Università' degli studi di Udine - Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Cattedra di Internet of Things, prof. I. Scagnetto

Relazione Progetto "Serratura ad Apertura Bluetooth"

Autore: Federico Pierobon - 144399 - 144399@spes.uniud.it

Idea

Al giorno d'oggi sempre più persone cercano la comodità, ad esempio muovendosi con meno cose possibili.
Inoltre crescono sempre di più le persone che possiedono dispositivi collegabili, via Bluetooth, al telefono: cuffie, smartwatch, occhiali smart.
Uno degli oggetti che si porta più spesso in giro sono le chiavi fisiche per aprire la porta di casa e, con l'avanzare della tecnologia, sono diventate probabilmente le cose più ingombranti.
Ad esempio una persona che va a correre preferirebbe girare senza le chiavi, invece si porterebbe lo smartphone o lo smartwatch, così da poter ascoltare musica.
Allora l'idea è nata per combinare entrambe le cose: realizzare una porta, con una serratura Bluetooth, in grado di aprirsi dopo aver rilevato il MAC Address del dispositivo del proprietario.
Ovviamente, per ragioni di sicurezza, la porta dovrà lo stesso supportare l'apertura fisica mediante chiavi, altrimenti in caso di problemi sarebbe impossibile accedere nell'immobile.
Un esempio potrebbe essere per un mancamento dell'elettricità.
Il prototipo è stato pensato per realizzare la serratura via Bluetooth e, successivamente, caricare in un Database le informazioni relative agli accessi, così che un'eventuale applicazione futura potesse far visualizzare alle persone chi è entrato.

Obiettivi

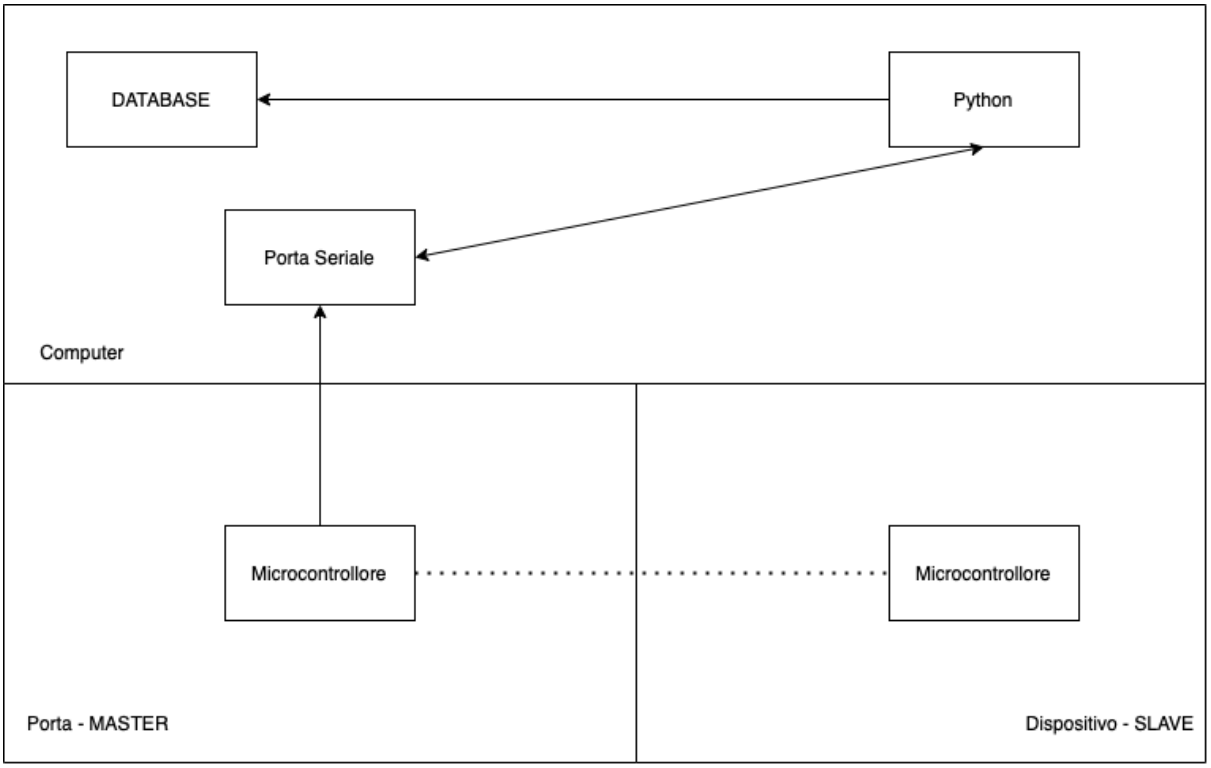
Il progetto realizzato consiste in una serratura ad apertura automatica.
L'automazione è presente grazie all'utilizzo di un modulo bluetooth, che verifica la presenza dei dispositivi riconosciuti come safe.
Per safe si intende qualunque dispositivo con MAC address riconosciuto come appartenente ad un utente possessore della porta.
Il primo obiettivo è stato quello di ricercare i vari moduli:

- modulo bluetooth HM-10 BLE;
- modulo motorino stepper;
- modulo driver per motorino stepper;

Il secondo obiettivo, dopo aver ricercato i vari componenti, è stato quello di studiare i relativi DataSheet.
Dopo aver appreso le potenzialità dei singoli moduli, si sono fatti dei test per ognuno.
Quello più interessante è stato il modulo bluetooth, con i suoi comandi AT.
Alla fine si sono messi insieme i vari componenti e, dopo una prima fase di testing, si è passati a realizzare il prototipo della porta in una tavola di compensato.
Successivamente si è realizzato un database locale con PostGresSQL, dove poter memorizzare i vari accessi.
Il passaggio da Arduino al DBMS è stato reso possibile grazie all'utilizzo di Python e interrogazioni SQL.

Descrizione

Il prototipo è stato pensato come sistema IoT di livello 1, siccome in locale, con 1 nodo e con una bassa mole di dati.
Ovviamente in caso si dovessero sviluppare versioni future, si passerebbe ad un livello 2, dove l'applicazione è cloud-based.
Una versione ancora più futura, dove ci possa essere la possibilità di avere più nodi che interrogano il cloud, potrebbe portare il sistema ad un livello 4.
Lo schema logico generale del sistema implementato è il seguente:

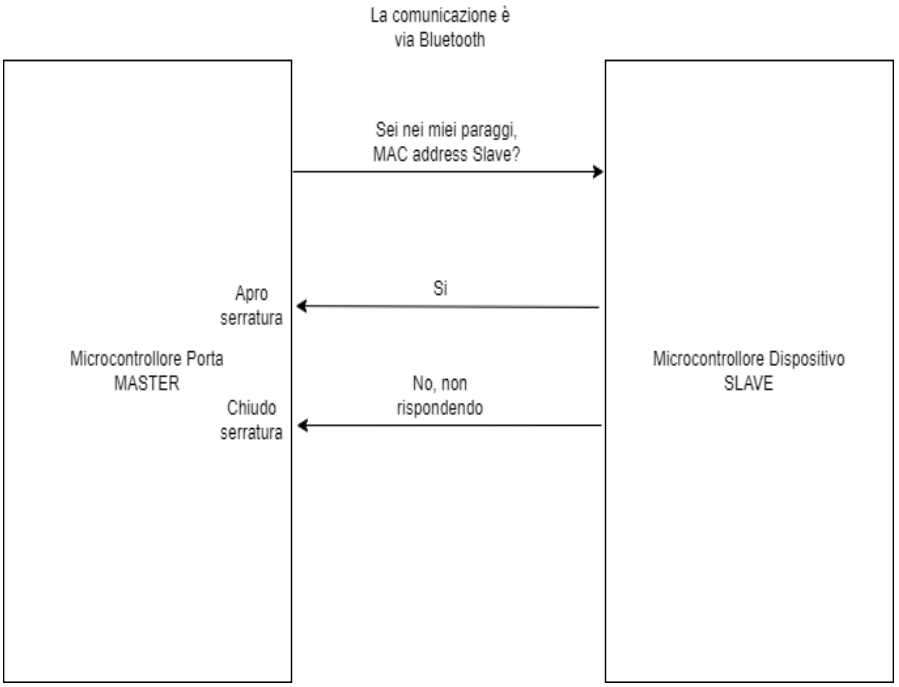


Dove la comunicazione via Bluetooth è rappresentata da un segmento tratteggiato.

Prima di analizzare le singole comunicazioni fra le diverse entità del sistema, è bene capire come funziona il tutto.
Il microcontrollore Slave è colui che simula un dispositivo bluetooth, e per la rappresentazione fisica sarà necessario un modulo Bluetooth e una Scheda Arduino Uno.
Il suo compito è quello di farsi "notare" dal Master.
Il microcontrollore Master ha il dovere di simulare la porta, quindi rimanere perennemente alla ricerca dello Slave e, non appena rilevata la sua presenza, provare a connettersi.
Se connesso aprirà la serratura e comunicherà, via Porta Seriale, che lo Slave si è connesso.
Python farà da interfaccia tra Porta Seriale e Database, prelevando le informazioni dalla porta e interrogando il Database.
Il Database sarà in locale.

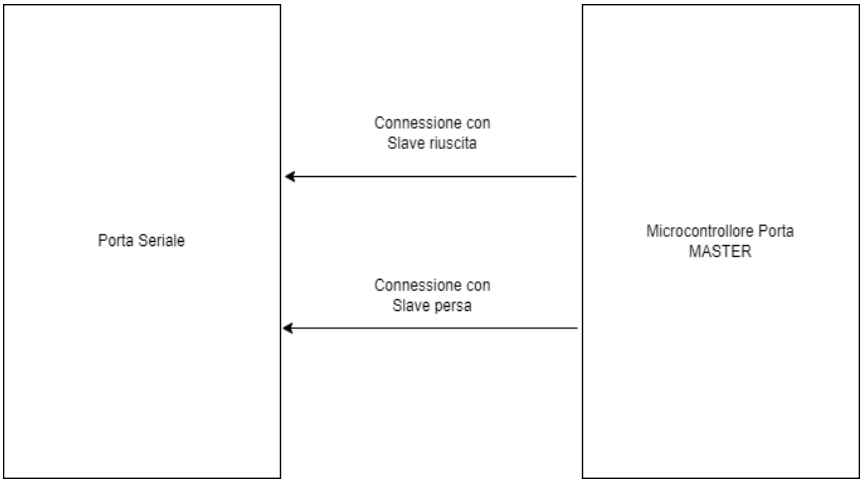
Analizzando i singoli casi

Master - Slave



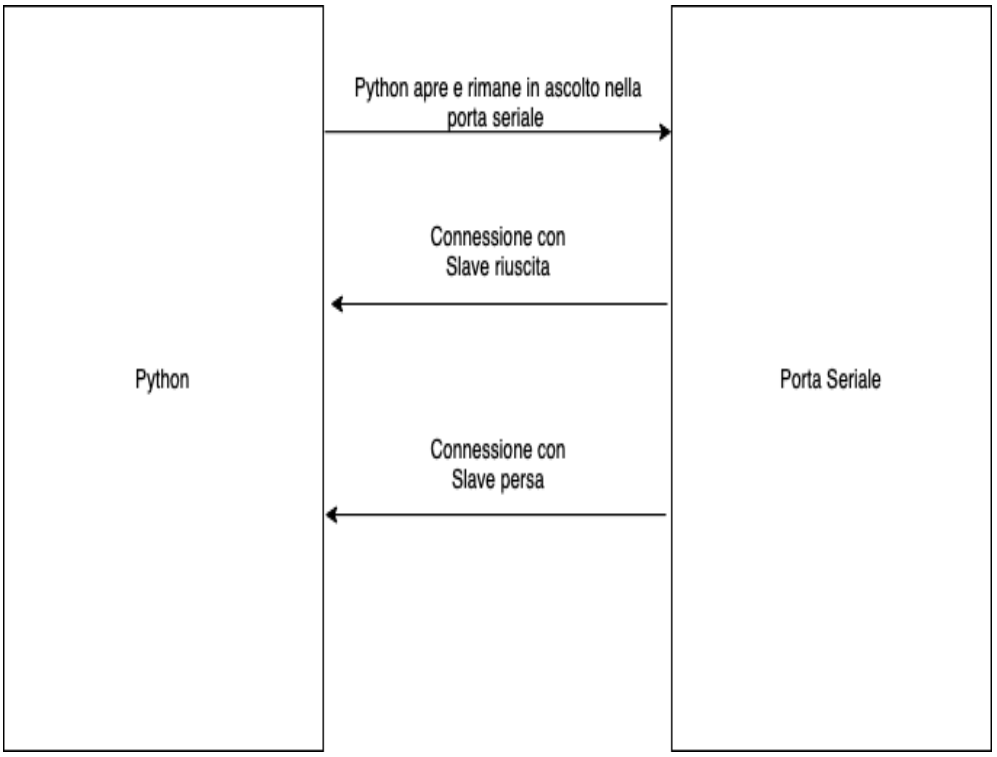
Ogni 20 ms, il Master cerca di riuscire a connettersi con lo Slave.
Lo slave quando rileva tale richiesta accetta sempre la connessione, quindi si ha la sicurezza che, in caso nessuno accetti, il dispositivo Slave sia lontano o spento.
Se la risposta è positiva, quindi lo Slave è nei paraggi, allora apre la serratura, altrimenti chiude la serratura.

Porta Seriale - Master



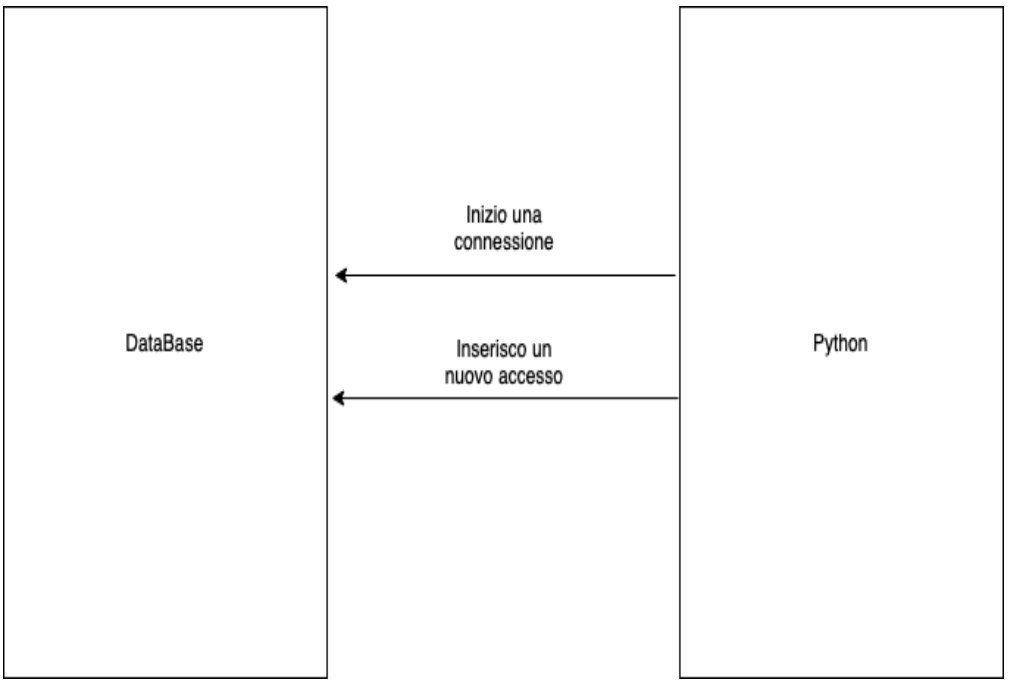
Ogni notifica, relativa allo stato della connessione tra master e slave, viene comunicata alla porta seriale. Ci sono 2 tipi di notifiche, una riguarda l'esito positivo della comunicazione, mentre l'altro comunica l'esito negativo, in caso in cui la comunicazione sia stata persa o non sia stato rilevato lo slave.

Porta Seriale - Python



Come prima cosa viene aperta una porta seriale e periodicamente, il programma verifica se è presente o meno un messaggio. Se il contenuto dichiara l'avvenuta apertura della serratura, allora si caricano le informazioni sull'accesso nel database.

Python - Database



Come detto sopra, ogni volta che si verifica un accesso, il programma lo inserisce all'interno del database. Prima però instaura una connessione.

Realizzazione

Essendo un prototipo e avendo utilizzato Arduino, è stato difficile riuscire a implementare come dispositivo slave lo smartphone.
Il problema principale è stata la non possibilità di instaurare una connessione tra iPhone e modulo bluetooth, ma successivamente verrà spiegata meglio l'incompatibilità.

Prima Fase - Studio e Ricerca dei moduli

Modulo Motorino Stepper e Relativo Driver

Denominato anche come motore Passo-Passo, è un motorino elettrico che lavora con corrente continua. Viene controllato grazie ad una serie di passi che ne compiono la rotazione. Il Driver utilizzato è l'ULN2003, con 4 fasi comandabili dai comandi inviati al driver, impostando i valori HIGH o LOW.
Il motore Stepper è connesso al Driver grazie a 4 cavi, mentre il Driver ha bisogno di:

- 4 PIN per le 4 fasi;
- 2 PIN di corrente:
    - Ground.
    - Positivo: dai 5V ai 12V.

Per semplificare l'approccio si è utilizzata la libreria di arduino Stepper.h.

Modulo Bluetooth

Dopo una prima ricerca sui vari dispositivi bluetooth proposti dal mercato, è stato scelto un modulo con versione Bluetooth 4.0 in su che supportassero il BLE, ovvero Low Energy.
Il Bluetooth 4.0 è stato rivoluzionario in quanto al posto di utilizzare il classico Bluetooth con un alto consumo di energia e una discreta mole dati inviati al secondo, si dimezza il consumo energetico, aumentando però i dati inviati al secondo.
Tabella con le caratteristiche prima e dopo l'avvento della versione 4.0:

Caratteristiche	Bluetooth Standard	Bluetooth BLE
Consumo di Energia	meno di 30 mA	meno di 15 mA
Velocità	700 Kbps	1 Mbps
Range	meno di 30 m	50 m

Quindi si è optato per un modulo Bluetooth BLE, in particolar modo il dispositivo scelto è stato l'HM-10.
E' importante sottolineare che il modello Low Energy viene utilizzato per tutte le tecnologie dell'IoT, poiché permette un risparmio energetico notevole.

Come tutti i moduli è presente un DataSheet, dove si trova la documentazione che riassume le caratteristiche di un componente.

Dopo averlo studiato, si sono usati i seguenti comandi AT:

- AT+ADDR?: il modulo risponde con il proprio MAC Address.
- AT+ROLE1: il modulo viene impostato come Master.
- AT+ROLE0: il modulo viene impostato come Slave.
- AT+CON'MAC-Address': il modulo utilizzato per la richiesta deve essere il Master e il Mac Address specificato dello slave. Il comando prova a connettere i due moduli, e le risposte possono essere:
    - OK+CONNA: abbinamento riuscito.
    - OK+CONNE: errore nell'abbinamento.
    - OK+CONNF: abbinamento fallito.
- AT+NOTI1: vengono mandate notifiche via porta seriale riguardo lo stato del modulo:
    - OK+CONN: moduli abbinati con successo.
    - OK+LOST: comunicazione persa.

L'HM-10 ha un totale di 4 PIN:

- 2 PIN per l'energia: uno per il Ground e l'altro per il positivo che lavora dai 3.6V ai 6V.
- 2 per la comunicazione: RX e TX.

Durante la fase di progettazione si era pensato di utilizzare un modulo Bluetooth come master e come slave uno smartphone, nello specifico un iPhone.
Il problema di lavorare via Bluetooth con i telefoni Apple è l'essere molto limitati con i programmi sviluppabili. L'azienda permette solo ai produttori accreditati MFI l'accesso alle specifiche tecniche, e alle risorse necessarie per creare accessori che comunicano con i loro dispositivi.
Si è deciso quindi di utilizzare 2 shield HM-10, uno come master e uno come slave, che simulasse lo smartphone.

Per utilizzarli è stato necessario importare la libreria di Arduino SoftwareSerial.h.

Seconda Fase - Testing dei Singoli Moduli

Testing HM-10

In questa fase si sono fatti 2 test:

- riuscire a interagire, attraverso l'uso dei comandi AT, dal monitor seriale al modulo bluetooth.
- instaurare una connessione fra il master e slave.

Entrambe le fasi sono state realizzate utilizzando la libreria SoftwareSerial.h.

Prima Fase

Come prima fase si è cercato di interagire con il modulo, sfruttando il Serial Monitor.
Innanzitutto bisogna istanziare la libreria, inserendo i PIN con la quale ci si è connessi al modulo Bluetooth. Come primo valore va inserito RX e come secondo valore TX, anche se sono da invertire: l'RX inserito corrisponde al TX del modulo e viceversa.
RX e TX stanno per Receive e Transmit.
Lo Shield lavora ad una frequenza di 9600 Baud, ragion per cui, bisogna settare la frequenza della porta seriale e il data rate del HM-10 a 9600.
Nel metodo loop, si continua a cercare se è presente qualche messaggio in ingresso (BTSerial.available) o in uscita (Serial.available).
Ogni porta seriale software possiede un buffer in ricezione e in invio, per rilevare la presenza o meno di messaggi non letti, bisogna controllare con i comandi .available() se è vuota o meno la coda in ricezione o in invio.
Se ho un dato che il modulo bluetooth ha inviato alla porta seriale, e non è ancora stato letto / prelevato dal buffer, allora lo si scrive nella porta seriale, quindi nel monitor, viceversa, se ho un dato da dover inviare al modulo, lo scrivo nella sua porta seriale.

Codice:

```
#include <SoftwareSerial.h>

SoftwareSerial BTSerial(2,3);

void setup(){
  Serial.begin(9600);
  BTSerial.begin(9600);
}

void loop(){
  if(BTSerial.available()){
    Serial.write(BTSerial.read());
  }
  delay(100);
  if(Serial.available()){
    BTSerial.write(Serial.read());
  }
}
```

Successivamente, grazie ai comandi AT, si sono ricavati e impostati dei valori nei singoli moduli:

1. Per ognuno dei due shield si è dapprima verificato in che 'Ruolo' fossero di default, ovvero Slave, e per uno dei due si è impostato AT+ROLE1, ovvero Master.
   Uno dei due è quindi Slave, mentre l'altro Master.
2. I due moduli si identificano univocamente grazie al MAC-Address, un indirizzo di 48 bit utilizzato nei protocolli che lavorano a livello 2 dello stack ISO/OSI.
   Per scoprire l'identità dello slave si è digitato AT+ADDR?, e la risposta è stata:
   AT+ADDR:0035FFE4E97C;
3. Siccome si lavora sulle risposte inviate dal Master al Serial Monitor, si è attivata la ricezione di notifiche: AT+NOTI1.

Seconda Fase

Dopo aver impostato i due dispositivi, si è lavorato principalmente sul Master con lo scopo di far cercare continuamente lo Slave e, al momento della connessione con esso, restituire una stringa nel Monitor Seriale.

Codice:

```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(2,3);
String command;
char c = " ";
bool state = false;
bool ciclo = false;

void setup() {
  Serial.begin(9600);
  mySerial.begin(9600);
}
void loop() {
```

Se non connesso, quindi stato = false, si manda al bluetooth il comando per provare a connettersi, digitando il comando AT+CONMacAddressSlave.

```
  if(state == false){
    mySerial.write("AT+CON0035FFE4E97C");
  }
```

Scopo: concatenare una serie di caratteri per ricavarne una stringa con la notifica.
Output: stringa con il comando.
Motivo: è stato necessario introdurre questo ciclo perché la comunicazione avviene passando singoli char e non string. Siccome dopo ci sarebbe stato un confronto tra string e string per capire la notifica arrivata, si è utilizzata la concatenazione per formare una stringa dai singoli char.

```
  while(mySerial.available() && ciclo == false){
    c = mySerial.read();
    command.concat(c);
    if(c == 'T' || c == 'A' || c == 'E' || c == 'F' || c == '#'){
        Serial.println(command);
        Serial.write("\n");
        ciclo = true;     }  }

//si sono connessi
  if(command == "OK+CONN" ||command == "OK+CONNA"){
      state = true;
      Serial.println("Connesso");
  }
//si è persa la connessione
  if(command =="OK+CONNE"||command =="OK+CONNF"||command == "OK+LOST"){
    if(state != false){
      Serial.println("Non Connesso");
    }
    state = false;
  }
//si resettano i parametri
  command = "";
  ciclo = false;
  delay(100);
}
```

Grazie a questo codice, si è riuscita a instaurare una connessione in caso il master rilevasse nei paraggi il dispositivo Slave, mentre, non appena spento o allontanato, la connessione viene persa.
Il tutto viene comunicato alla porta seriale grazie alle continue stampe.

Per quanto riguarda il modulo Slave, non si è aggiunto nulla al codice del primo test poiché il suo scopo è solo quello di rimanere in ascolto e, non appena il master comunica la sua volontà nel connettersi, accettare.

Testing Motore Stepper e Rispettivo Driver

Come prima cosa si è ricercata una libreria che semplificasse i comandi da mandare al Driver, e si è scelta la Stepper.h.
Si sono riscontrati dei problemi con la libreria e dopo una ricerca, modificando il codice di essa, si sono risolti.
La difficoltà stava nella non possibilità di ruotare in modo antiorario il motorino, probabilmente per incompatibilità delle fasi.
Dopo aver riscritto una piccola parte di codice della libreria, si è proseguito con il test.
Il test effettuato è stato quello di far muovere ripetutamente di 360° il motore, in senso antiorario e in senso orario.

Codice:

```
#include <Stepper.h>
```

La variabile step_mode identifica quanto giro far fare al motorino ogni volta, se impostata a 2 allora metà, 180°, mentre se 4 un pieno giro di 360°.

```
int step_mode = 4;
```

stepsPerRevolution è un valore intero che indica quanto dovrà girare, e il parametro che lo fa variare è step_mode.

```
int stepsPerRevolution = 2048 * (step_mode/4) ;
int mysteps = 1 * stepsPerRevolution;
```

Si inizializza la libreria.

```
Stepper myStepper(stepsPerRevolution,8,9,10,11,step_mode);

void setup() {
```

Si imposta la velocità di movimento a 60 rpm.

```
myStepper.setSpeed(60);
Serial.begin(9600);
}
```

Nel loop() si fa andare in senso antiorario e orario il motorino.

```
void loop() {

myStepper.step(-mysteps);
delay(500);

myStepper.step(mysteps);
delay(500);
}
```

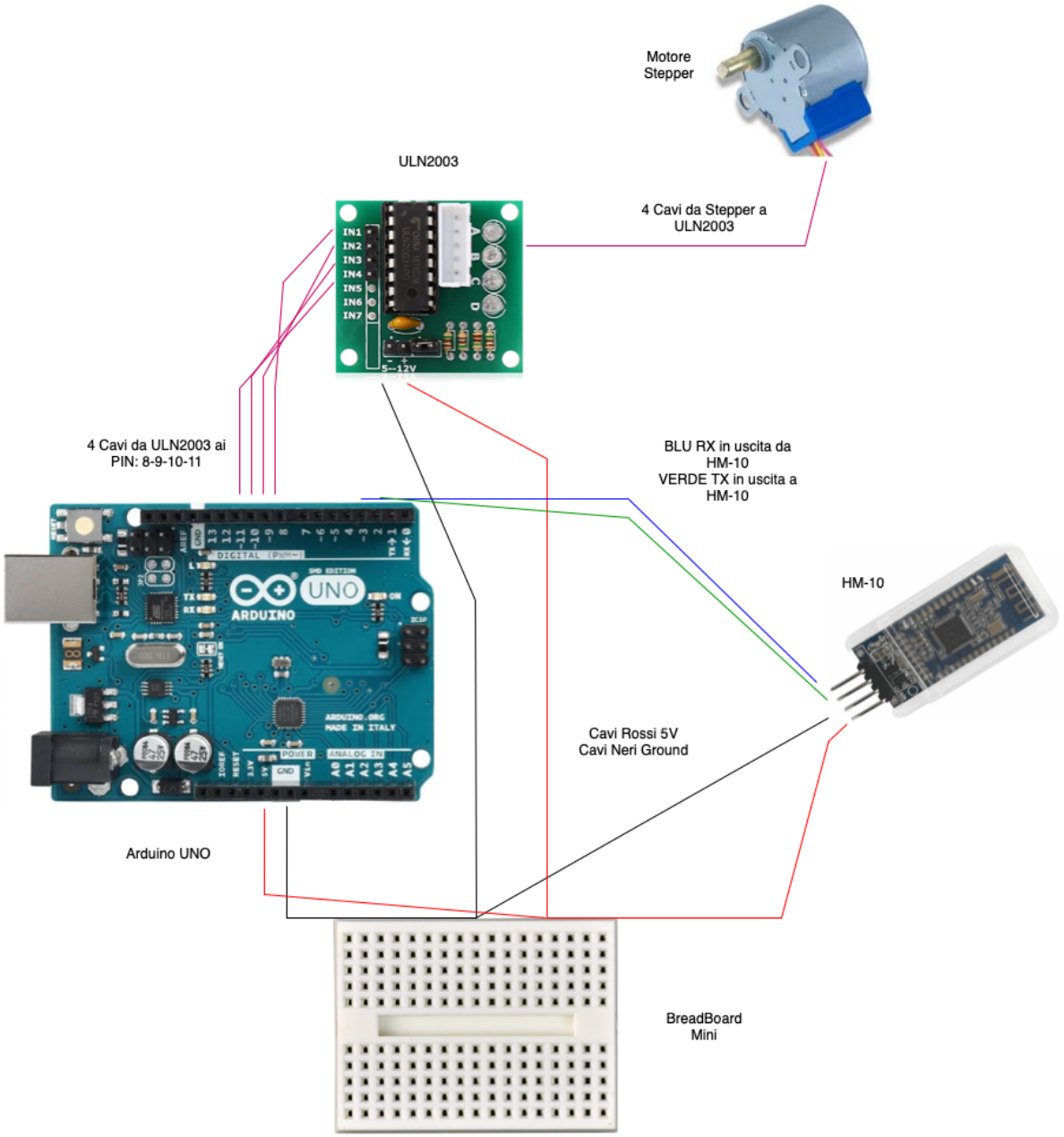
Terza Fase - Testing Prototipo

Finiti i 3 test si è iniziato a pensare come poter realizzare fisicamente una porta.
Dopo una prima fase di progettazione si è deciso di prendere:

- 1 lastra di compensato;
- 2 cerniere;
- 1 ‘serratura’, successivamente riadattata.

Una volta completati i vari lavori di falegnameria e bricolage, si sono attaccati i pezzi prendendo dapprima le misure, successivamente bucando il compensato e applicando delle fascette.

Lo schema logico Finale:



Il device è stato simulato utilizzando Arduino Uno e il modulo Bluetooth.

Si è poi preso lo scheletro del codice della seconda fase del test dei moduli bluetooth e, dove serviva che si aprisse la serratura, si è inserito il codice per far girare il motorino.

Codice:

```
#include <SoftwareSerial.h>
#include <Stepper.h>

SoftwareSerial mySerial(2,3);
String command;
char c = " ";
```

L'unica cosa modificata qua è stata quella di far compiere al motorino metà giro, e non un giro completo, quindi step_mode = 2.

```
int step_mode = 2;
int stepsPerRevolution = 2048* (step_mode/4) ;
int mysteps = 1 * stepsPerRevolution;
Stepper myStepper(stepsPerRevolution,8,9,10,11,step_mode);

bool state = false;
bool ciclo = false;

void setup() {
  Serial.begin(9600);
  mySerial.begin(9600);
  myStepper.setSpeed(60);
}

void loop() {
  if(state == false){
    mySerial.write("AT+CON0035FFE4E97C");
  }

  while(mySerial.available() && ciclo == false){
    c = mySerial.read();
    command.concat(c);
    if(c == 'T' || c == 'A' || c == 'E' || c == 'F' || c == '#'){
        ciclo = true;
    }
  }
```

Se connesso allora apre la serratura.

```
  if(command == "OK+CONN" ||command == "OK+CONNA"){
      myStepper.step(-mysteps);
      state = true;
      Serial.println("connesso");
  }
```

Se disconnesso allora chiude la serratura.

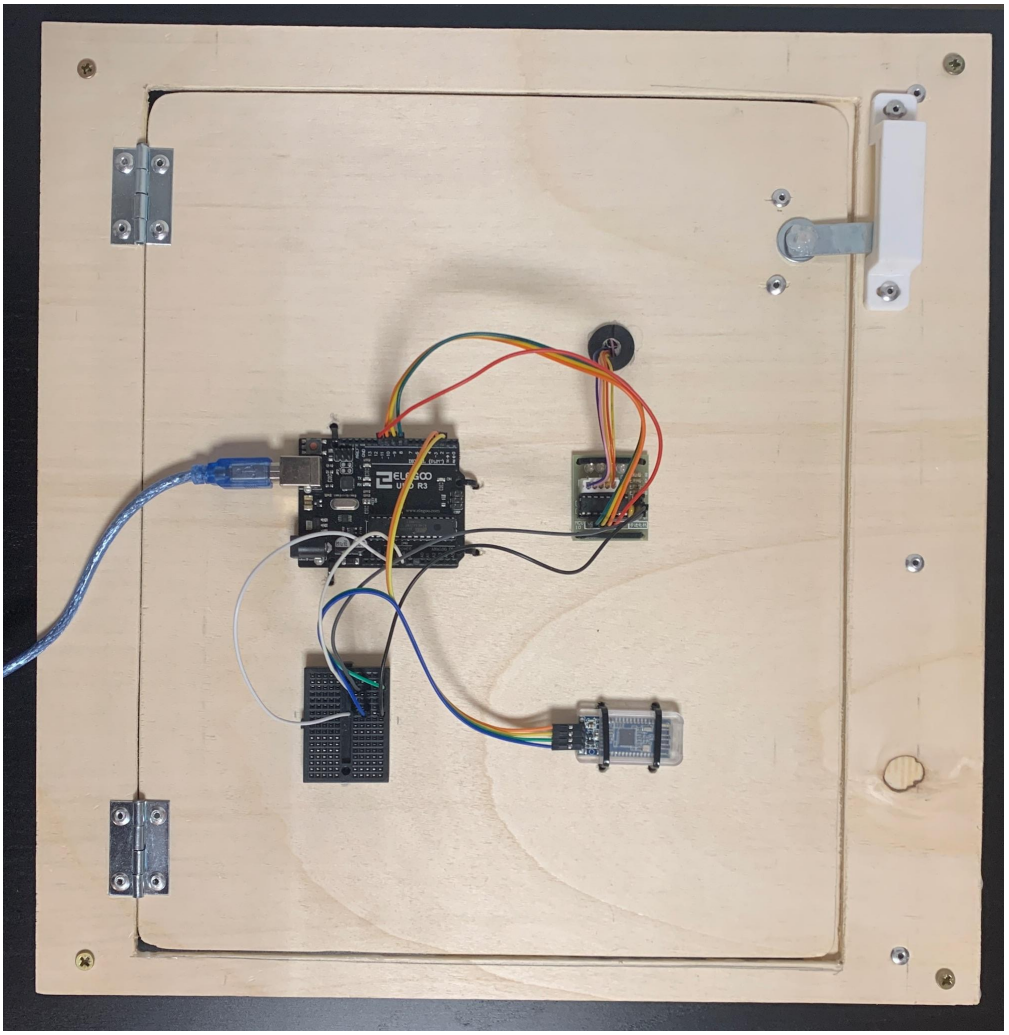
```
 if(command =="OK+CONNE"||command =="OK+CONNF"||command == "OK+LOST"){
   if(state != false){
     myStepper.step(mysteps);
   }
   state = false;
   Serial.println("non connesso");
 }

 command = "";
 ciclo = false;
 delay(200);
}
```

Come detto in precedenza, oltre che chiudere o aprire grazie al motorino la serratura, il programma scrive continuamente nella porta seriale:

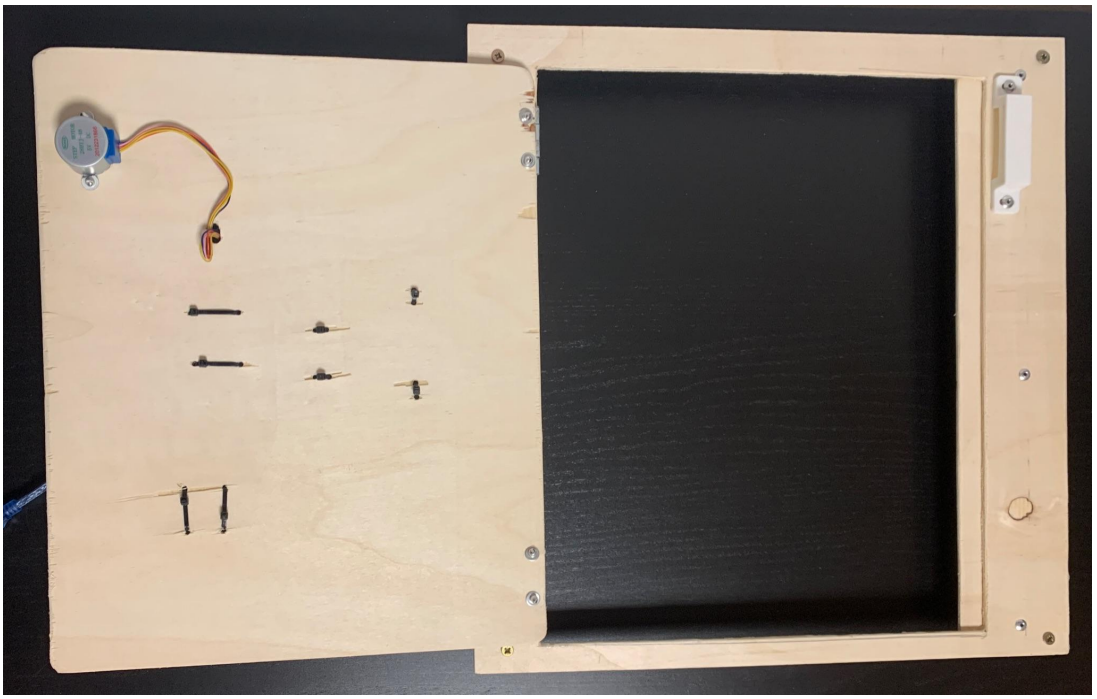
- connesso: apertura della serratura;
- non connesso: chiusura della serratura;

Queste stampe serviranno successivamente durante la scrittura del codice Python, per rilevare lo stato della porta.

Il Risultato Finale con porta chiusa:

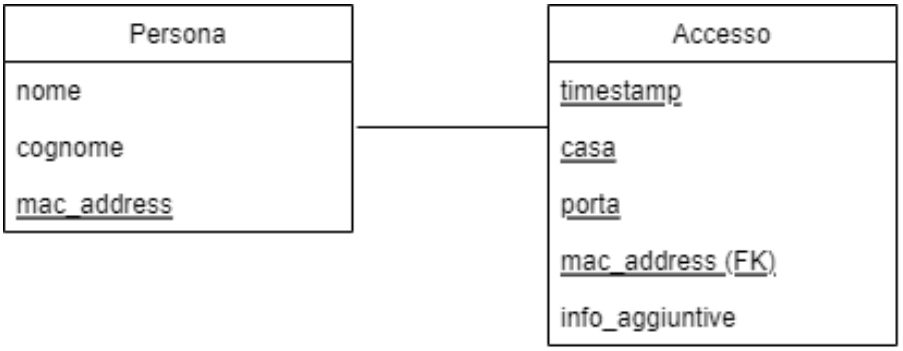


Il Risultato Finale con porta aperta:



Quarta Fase - Creazione Database

Il Database è stato creato in un DBMS locale, grazie all'applicazione PostGresSQL.
Lo schema logico del Database è il seguente:



Mentre le operazioni SQL usate per crearlo sono state:

```
CREATE DATABASE ProgettoIoT;

CREATE TABLE persona(
    nome VARCHAR(50) not null,
    cognome VARCHAR(50) not null,
    mac_address VARCHAR(48),
    Primary key (mac_address)
);

CREATE TABLE accesso(
   timestamp TIMESTAMP,
   casa VARCHAR(50),
   porta VARCHAR(50),
   info_aggiuntive VARCHAR(100),
   mac_address VARCHAR(48),
   Foreign key (mac_address) References persona(mac_address),
   Primary key(mac_address, timestamp, casa, porta)
);
```

La chiave primaria della tabella persona è composta dall'attributo mac_address.
La chiave primaria della tabella accesso, è formata dalla tupla:
mac_address, timestamp, casa, porta.
La chiave esterna nella tabella accesso è mac_address e, grazie ad essa, è possibile ricavare i dati dell'utente che ha effettuato l'accesso.
Casa identifica l'abitazione di dove si è eseguito l'accesso, mentre Porta la rispettiva serratura aperta.

Prima di continuare con la progettazione, si è inserito il seguente record:

```
insert into persona(nome, cognome, mac_address) values
('Federico','Pierobon','0035FFE4E97C');
```

Che verrà utilizzato in seguito.

Quinta Fase - Codice Python

In questa fase si è creata un'interfaccia che gestisce sia i dati in arrivo dalla porta seriale, sia le interrogazioni al DBMS.
Ogni qualvolta la porta seriale ha dei dati, il programma li prende e li esamina.
Se i valori ricevuti segnalano un'apertura della porta, viene inserito un record nel DBMS per segnalare un accesso.
Essendo in fase di prova la query inserisce valori statici.

Codice:

Si importano le librerie.

```
import psycopg2
import time
import serial
from datetime import datetime
```

Si dichiara la porta seriale da aprire, il baud rate e il timeout, ovvero quel tempo in secondi che viene usato quando non ci sono abbastanza dati da leggere e, la sua scadenza forza il restituire tutti i byte letti fino a quel momento.
Successivamente si aspetta 2 secondi.

```
ser = serial.Serial('/dev/cu.usbmodem146401', 9600, timeout=0.5)
time.sleep(2)
```

Cicli infinito per continuare a prelevare valori dalla porta seriale.

```
while True:
```

Si legge il valore sulla porta seriale, e si converte prima in utf-8, successivamente si rimuovono tutti i caratteri / spazi indesiderati.

```
  data = ser.readline().decode('utf-8').rstrip()
```

Se nella variabile data sono presenti dei valori, allora passa la condizione e come prima cosa viene stampata sulla console. Questa stampa si è usata soprattutto in fase di testing per capire quando fosse stato inserito un nuovo record nella tabella.
Difatti successivamente si verifica se la stringa sia uguale a 'connesso' o 'OK+CONN', due valori che identificano la corretta apertura della serratura.

```
  if data:
     print(data)
     if (data == 'connesso' or data == 'OK+CONN'):
```

Passata la condizione si vuole inserire un nuovo record nella tabella accesso.
Come prima cosa viene calcolato il timestamp, e per farlo viene prima calcolata l'ora e il giorno in secondi, successivamente si trasforma così da avere il formato SQL:
Anno - Mese - Giorno - Ora - minuti - secondi.

```
ts = time.time()
timestamp = datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
```

Si effettua la connessione al DBMS, dichiarando il nome del database, il nome dell'utente con la quale effettuare il login, relativa password, l'indirizzo al quale connettersi e la relativa porta.

```
conn = psycopg2.connect(
        database="progettoiot", user='federicopierobon', password='', host='localhost',
        port= '8080'
)
```

Si apre un cursore che servirà per effettuare le interrogazioni.

```
x = conn.cursor()
```

Successivamente si è realizzato un costrutto try - catch.
Prima si prova a inserire all'interno di accesso e in caso di successo si esegue il commit.
In caso venga rilevato un errore, parte l'eccezione che esegue una rollback, ovvero un abort della transazione appena effettuata.
Alla fine si chiude la connessione con il DBMS.

```
try:
   x.execute("""insert into accesso(timestamp, casa, porta, mac_address)
values(%s,%s,%s,%s);""",(timestamp,'CasaTest1','PortaTest1','0035FFE4E97C'))
   conn.commit()
except:
   conn.rollback()
conn.close()
```

Il codice scritto non è strutturato bene perché, messo in questo modo, è più facile da commentare.
Il codice finale corretto è presente nell'allegato della cartella CodicePython.

Sviluppi Futuri

Essendo un prototipo, tutto quello realizzato fino ad ora serve per far andare in modo basico tutto il sistema.
Ci sono diversi punti importanti da analizzare, prima di poter dichiarare che sia pronto per l'applicazione sul campo.

Sicurezza

La sicurezza è un aspetto importantissimo per tutte le applicazioni IoT, ancora di più se si parla della propria serratura di casa.
Possono effettuarsi diversi attacchi di tipo attivo.
Il meccanismo più semplice che si può implementare è quello di utilizzare una password, che il device invia alla serratura, per poterla aprire.
Ad oggi quasi tutti i sistemi ad apertura bluetooth utilizzano questa tecnica, e tanti di essi non passano i controlli di sicurezza o perché la password viene passata in chiaro, o perché un attaccante, sfruttando delle vulnerabilità del sistema, riesce a cambiare la chiave simmetrica.
Come si può notare, già implementando un robusto algoritmo di cifratura simmetrica si riuscirebbero a bloccare tanti tipi di attacco, ma non è abbastanza.

In seguito verranno analizzati alcuni tipi di attacco e le relative soluzioni:

- MAC spoofing: l'attaccante maschera il proprio indirizzo MAC con un altro.
- Jamming: tecnica in cui l'attaccante disturba, introducendo del rumore, le comunicazioni radio scambiate dai bluetooth in questo caso, creando quindi un Denial Of Service.
- Rubare il device e entrare "legalmente".

Come si può notare, tutti questi attacchi sono categorizzati come attivi perché alterano il funzionamento del sistema, mentre un attacco passivo potrebbe essere utilizzato per studiare il traffico di nascosto, ad esempio con il metodo del Traffic analysis, per osservare le informazioni della trasmissione cercando eventuali vulnerabilità nel sistema.

MAC Spoofing

Bisogna trovare un modo per impedire che un qualunque dispositivo riesca, modificando il proprio MAC con uno safe, ad essere accettato dal sistema.
Ci possono essere diverse tecniche di prevenzione, si è deciso di analizzarne due:

1. Una prima tecnica consiste nell'utilizzo di un'applicazione aggiuntiva connessa ad internet.
   L'apertura delle porte avviene solo quando si è accettati sia via Bluetooth, sia via Internet, come una doppia autenticazione.
   Fra le due tecniche è considerata la più sicura, ma comporta dei costi.
   Innanzitutto il device utilizzato dall'utente deve avere una connessione ad internet, e ciò non è scontato nell'insieme di device che implementano almeno Bluetooth.
   Ad esempio certi Apple watch non implementano tale funzionalità senza avere un telefono vicino.
   Dopodichè bisogna possedere una connessione stabile nell'abitazione siccome, ad ogni accesso, la porta deve poter comunicare con il Server.

2. La seconda tecnica sfrutta l'algoritmo di cifratura simmetrica, sfruttando l'idea del canale insicuro di Dolev Yao, dove le due parti saranno la porta e il device, mentre la terza parte sicura sarà il server.
   Denotiamo come A la porta e B come il device scelto per aprire la porta.
   Prima di tutto bisogna far sapere ad A che B è riconosciuto come dispositivo safe, successivamente ogni volta che B viene rilevato da A, A lo accetta solo se B riesce a superare una challenge che solo i dispositivi safe possono superare.
   Un possibile procedimento potrebbe essere:
   - La porta A viene installata e collegata ad internet;
   - In un eventuale pannello di controllo della porta A, l'utente ha la possibilità di avviare una ricerca dei dispositivi e selezionare il proprio dispositivo B come safe.
   - Ora A sa che B è safe, e lo salva nel Database.
   - Presupponiamo che B abbia una connessione ad Internet, e abbia scaricato l'applicazione e fatto il login.
   - Il Server invia a B una chiave simmetrica.
   - Ora, lo smartphone possiede la chiave nel locale, quindi non serve una connessione ad internet.
   - Ogni volta che A rileva B, li manda una sfida che solo lui può risolvere:
     $A \rightarrow B$ A, Key(messaggio)
     B grazie alla sua chiave decifra il messaggio e lo rinvia:
     $B \rightarrow A$ (messaggio)
     Se B riesce a decodificare il messaggio allora è sicuramente lui o un altro dispositivo safe.
     Nel primo messaggio viene scambiata anche l'identità di A, così B sa con quale chiave cifrare il messaggio siccome un device potrebbe essere connesso a più porte, e ognuna potrebbe utilizzare una chiave diversa.

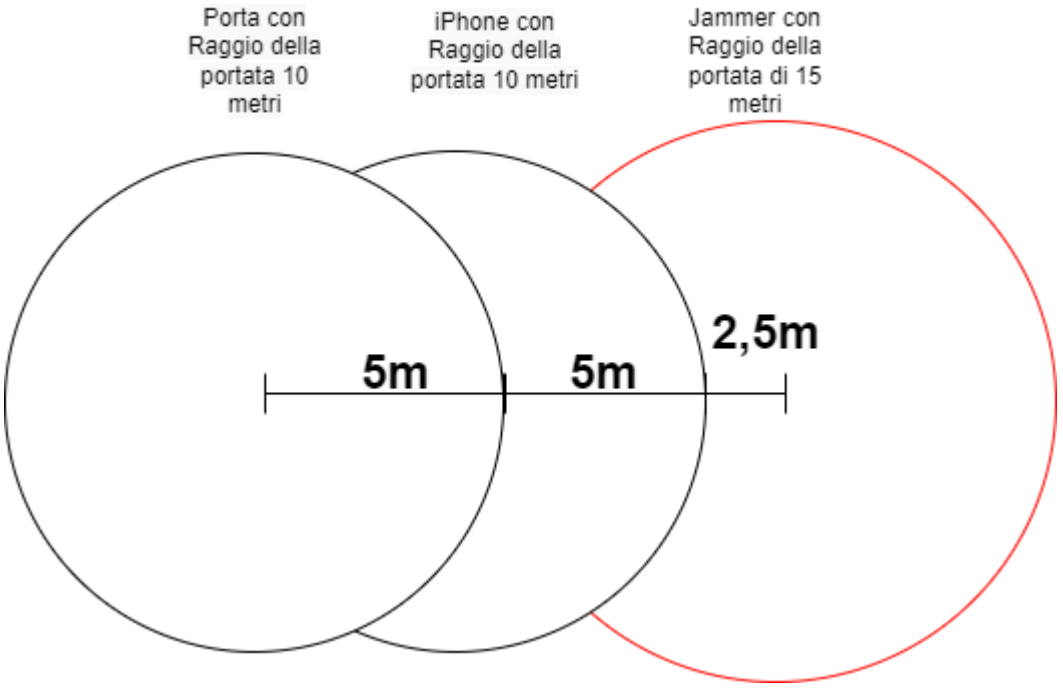
   Questa tecnica richiede che il messaggio sia generato ad ogni challenge, così da evitare attacchi di tipo replay (si rinviano più volte gli stessi messaggi).
   L'utilizzo di questa tecnica potrebbe essere fatto da tutti quei dispositivi che non possiedono una connessione internet propria, ad esempio quelli citati nella soluzione 1.
   Il contro è l'utilizzo massiccio dell'hardware, poiché la cifratura costa molto computazionalmente parlando, specialmente se si parla di dispositivi con una bassa potenza di calcolo come gli smartwatch.

Tra le due soluzioni non ce n'è una migliore:

1. E' possibile implementarla solo nei dispositivi che possiedono la capacità di collegarsi ad Internet ogni volta che si voglia aprire la porta. Costa meno in termini computazionali. Ideale per Smartphone.
2. Si può implementare in entrambi i tipi di dispositivi: connessi e non connessi ad internet. Durante la configurazione è necessario però una connessione, per potersi scambiare le chiavi. Costa molto in termini computazionali, per colpa della cifratura.

Jamming

Un jammer è un disturbatore delle frequenze e ad oggi nel mercato non ci sono tecniche anti-jamming.
Di per sé l'apertura della porta funziona solo se il dispositivo si avvicina di tot metri (ipotizziamo 10m).
Questa è l'unica sicurezza che si può avere ovvero, un attaccante per aver influenze sulla comunicazione device-serratura, deve posizionarsi relativamente vicino.
Prendiamo come esempio un iPhone, con la portata standard del bluetooth di 10 m.
La porta rimane con una portata di 10 m, mentre il jammer, che di solito ha una portata dai 10 m ai 30 m, ipotizziamo ne abbia 15 m.
Esempio:



Facendo due conti, utilizzando la porta come centro, il jammer per non fare interferenze deve porsi ad una distanza > 12,5 metri.
Come detto in precedenza, non esistono tecniche di prevenzione per un attacco jamming perché di per sé è un attacco che introduce del rumore nella comunicazione, facendo sfasare lo spettro delle frequenze.

Furto del dispositivo

In questo caso l'attaccante ruba il device considerato safe, quindi con la capacità di aprire legalmente, per il sistema, la serratura.
Una possibile soluzione potrebbe essere quella di revocare lo stato safe per il dispositivo rubato.
Un ragionamento uguale viene utilizzato in caso di smarrimento / furto della carta di credito e il conseguente blocco di essa, tramite l'applicazione della banca.
Quindi dentro l'applicazione potrebbe esserci una sezione dedicata alla revoca di un certo indirizzo MAC considerato valido.
Ovviamente il tutto deve essere possibile solo alle persone che condividono la serratura.
Mettiamo caso che A e B siano due persone che condividono una serratura, mentre C e D due persone che condividono un'altra serratura.
AMac, BMac, CMac e DMac sono i rispettivi indirizzi MAC.
A perde il telefono, quindi AMac è compromesso.
B ha l'autorizzazione di revocare AMac, mentre C e D no, perché non condividono l'accesso alla serratura di A e B.
Ora che AMac è non è più valido, l'attaccante non può aprire la porta perché quest'ultima non considera più AMac come safe.
Questo genere di cambiamento da stato safe a non safe può essere eseguito grazie ad un'interrogazione della query che rimuove il MAC address dell'utente A.
Questo meccanismo di sicurezza aggiunge una complicazione:
la porta deve connettersi ad internet ogni qualvolta rilevi un device, verificandone lo stato dell'indirizzo MAC.

Applicazione

Lo sviluppo di un'applicazione diventa indispensabile per garantire un futuro al progetto, garantendo prima, durante la fase di installazione del sistema, un meccanismo di registrazione e successivamente, durante la fase d'uso, un'aggiunta di diverse funzionalità e meccanismi di sicurezza.
Potrebbe presentare all'interno diverse features:

- Inserisci nuovo device;
- Blocca serratura;
- Rimuovi device;
- Verifica gli ultimi accessi;
- Verifica lo storico degli accessi;
- Notifiche varie sullo stato della porta;

Oltre alle funzionalità con cui l'utente può interagire, ce ne sono altre in background che servono per il corretto e sicuro collegamento con la serratura.
Ad esempio nel caso della prevenzione del MAC Spoofing, un'applicazione diventa fondamentale per parlare con il server.

Domotica

Sempre più sono le case intelligenti, ovvero che applicano un sistema domotico al proprio interno.
Una serratura ad apertura Bluetooth, con la relativa applicazione, potrebbe garantire un livello di sicurezza in più.
Mettiamo caso che un ladro provi a sfondare la porta, oppure provi ad aprirla applicando la tecnica del MAC Spoofing.
Il sistema che individua l'attacco potrebbe mandare prima una notifica all'utente, successivamente attivare le telecamere dell'abitazione e abbassare le tapparelle.
Possono essere diverse le funzionalità da aggiungere, il problema spesso è la compatibilità tra le diverse marche di prodotti.
Una possibile soluzione al problema è l'utilizzo di assistenti personali intelligenti come Alexa e Google Home.
Questi sistemi permettono di gestire facilmente, l'intero ecosistema domotico della casa.
Uno sviluppo del progetto potrebbe quindi essere il rendere compatibile la serratura e la relativa applicazione con questi sistemi.

Conclusioni

Il prototipo di serratura ad apertura automatica mediante bluetooth è uno scheletro e una base sulla quale poi si potrà sviluppare qualcosa di più concreto.
Al giorno d'oggi questi dispositivi fanno parte di un settore del mercato IoT che sta crescendo molto, ma molti modelli offerti presentano dei problemi per quanto riguarda la sicurezza.
Durante l'edizione del DEF CON del 2016, una coppia di programmatori esperti in sicurezza ha realizzato dei test su 16 serrature e 12 non sono riuscite a passare la prova.
La principale vulnerabilità è dovuta al fatto che tutte le comunicazioni bluetooth sono in chiaro, quindi un attaccante può conoscere la password di sblocco semplicemente sniffando il traffico.
Il prototipo realizzato ovviamente non può essere affidabile e sicuro, perchè presenta soprattutto dei problemi di sicurezza e non è ancora stato sviluppato per i dispositivi come smartphone, smartwatch, etc.
Si è cercato però di parlare delle possibili soluzioni per quanto riguarda l'affidabilità, la sicurezza e l'aggiunta di nuove funzionalità in una possibile versione futura.

Sitografia

DataSheet Motorino Stepper e Driver ULN2003:
http://eeshop.unl.edu/pdf/Stepper+Driver.pdf

DataSheet modulo Bluetooth HM-10:
https://people.ece.cornell.edu/land/courses/ece4760/PIC32/uart/HM10/DSD%20TECH%20HM-10%20datasheet.pdf

Libreria Stepper.h:
https://www.arduino.cc/reference/en/libraries/stepper/

Libreria SoftwareSerial.h:
https://www.arduino.cc/en/Reference/SoftwareSerial

Modifiche applicate alla libreria Stepper.h:
https://forum.arduino.cc/t/stepper-motor-doesnt-turn-counter-clockwise-wrong-steps-number-indication/139845/3

MFi Program Apple:
https://mfi.apple.com

Le serrature Bluetooth sono un colabrodo:
https://www.dday.it/redazione/20703/le-serrature-bluetooth-sono-un-colabrodo-meglio-la-cara-vecchia-chiave

Libreria psycopg2 py:
https://pypi.org/project/psycopg2/

Software usato per disegnare i vari schemi logici:
https://app.diagrams.net