

Raspberry Pi & Smart Card

COME REALIZZARE UN SISTEMA DI ACCESSO



ANTONIO MUSARRA

HOEPLI
INFORMATICA

Table of Contents

Informazioni sulla Guida	1
1. Introduzione	2
2. Descrizione dello scenario	3
3. Dettagli sulle carte MIFARE Classic 1K	6
4. I misteriosi pacchetti APDU	9
5. Cos'è l'Answer to reset o ATR	13
6. Requisiti hardware	16
7. Requisiti software	18
8. Schema elettrico della soluzione	19
9. Progettare il software	21
10. Implementare il software	25
10.1. Cos'è Pyscard	26
10.2. Struttura del progetto	28
11. Deploy & Test	33
12. Conclusioni	41
13. Riferimenti	42

Informazioni sulla Guida

Antonio Musarra, Raspberry Pi e Smart Card Mifare Classic 1K - Guida pratica per realizzare un sistema di autenticazione con smart card contactless su Raspberry Pi

Copertina e impaginazione di Antonio Musarra

Edizione digitale Luglio 2024 (v1.0.2)

Serie: Elettronica&Informatica (#elettronica-informatica)

Promosso da: Antonio Musarra's Blog (<https://www.dontesta.it>)

Profilo LinkedIn <https://www.linkedin.com/in/amusarra/>

Il progetto di esempio realizzato per quest'opera è disponibile sul repository GitHub all'indirizzo <https://github.com/amusarra/smardcard-contactless-raspberry-pi>

Quest'opera è stata realizzata usando l'approccio doc-as-code e il testo è stato scritto in formato AsciiDoc.

Nel caso di errori, segnalazioni o suggerimenti, si prega di aprire una issue o discussione sul repository GitHub.

Progetto Smart Card Contactless Raspberry Pi

1. Apertura issue <https://github.com/amusarra/smardcard-contactless-raspberry-pi/issues>
2. Apertura discussione <https://github.com/amusarra/smardcard-contactless-raspberry-pi/discussions>

Nell'ambito del social coding e del contributo alla comunità, è possibile contribuire al progetto con una pull request per migliorare questa guida e il progetto di esempio.

Note sul Copyright

Tutti i diritti d'autore e connessi sulla presente opera appartengono all'autore Antonio Musarra. Per volontà dell'autore quest'opera è rilasciata nei termini della licenza Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International il cui testo integrale è disponibile alla pagina web <https://creativecommons.org/licenses/by-nc-sa/4.0/deed.it>

Tutti i marchi riportati appartengono ai legittimi proprietari; marchi di terzi, nomi di prodotti, nomi commerciali, nomi corporativi e società citati possono essere marchi di proprietà dei rispettivi titolari o marchi registrati d'altre società e sono stati utilizzati a puro scopo esplicativo e a beneficio del possessore, senza alcun fine di violazione dei diritti di copyright vigenti.

1. Introduzione

Le [Smart Card](#) fanno parte ormai da tempo del nostro quotidiano: dalla [SIM \(Subscriber Identity Module\)](#) del cellulare, alla carta di credito, come sistema di fidelizzazione, per accedere ai locali e come mezzo per fruire dei servizi della pubblica amministrazione.

Integrare sistemi di autenticazione basati su Smart Card nei propri sistemi e applicazioni, credo che possa essere un'attività molto interessante per chi programma e l'aspetto a mio avviso più coinvolgente è la vicinanza con quelli che amo definire bonariamente "pezzi di ferro".

Quasi due anni fa ho scritto l'articolo [Raspberry Pi – Un esempio di applicazione della TS-CNS](#), dove mostravo come poter creare un sistema di accesso utilizzando la propria TS-CNS; in questo articolo vedremo invece come mettere insieme: [Raspberry Pi](#), modulo da quattro relè, lettore di Smart Card e [MIFARE Classic 1K](#) contactless Smart Card per poi armonizzare il tutto, sviluppando il software necessario allo scopo di realizzare un sistema di accesso.

Per lo sviluppo del progetto è necessario toccare un numero considerevole di argomenti e cercherò di trattare i principali con il giusto livello di profondità, in caso contrario l'articolo assumerebbe delle dimensioni notevoli. Mi impegnerò a lasciare tutti i riferimenti utili per ogni vostro approfondimento.

A questo punto direi d'iniziare; mettetevi comodi e munitevi della giusta concentrazione perché la lettura di questo articolo sarà abbastanza impegnativa ma spero interessante.

2. Descrizione dello scenario

Credo d'intuire quale potrebbe essere il vostro pensiero in questo momento. **Che tipo di Sistema di Accesso andremo a realizzare?**

Vorrei implementare qualcosa che tutti almeno una volta abbiamo usato. Un comunissimo scenario di **Sistema di Accesso** lo abbiamo vissuto tutti accedendo alla nostra camera di albergo grazie alla Smart Card (o chiave elettronica) consegnata al momento dell'accoglienza.

La figura a seguire mostra lo schema hardware della soluzione che andremo a realizzare.

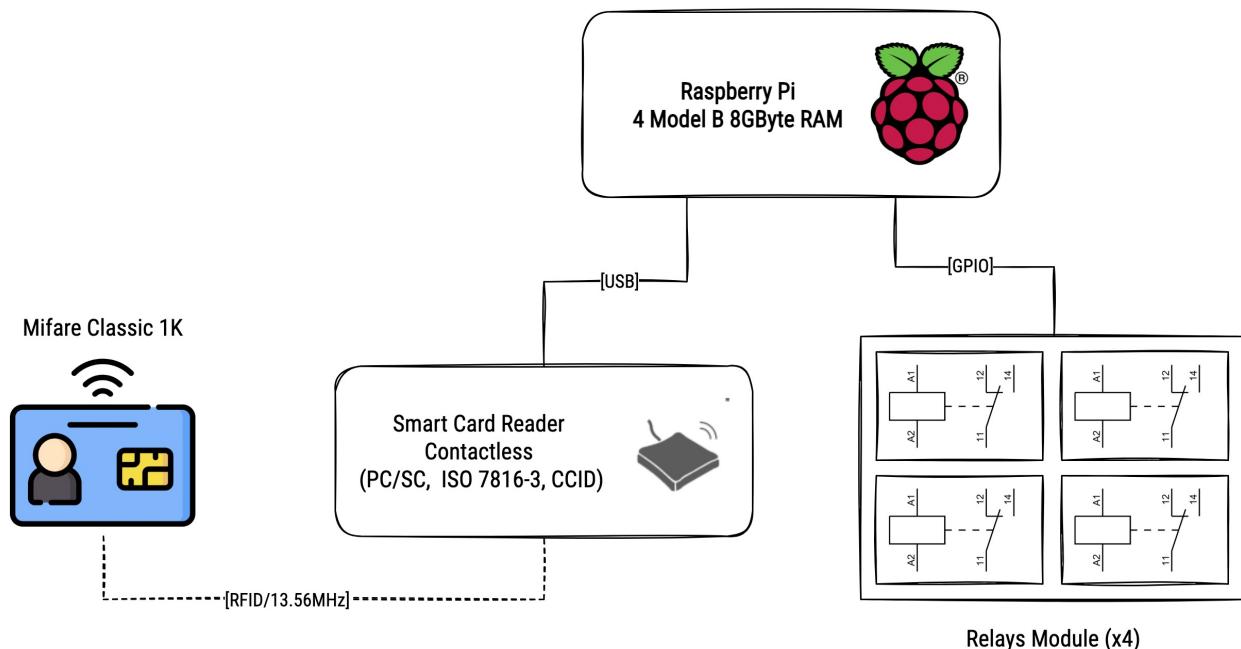


Figura 1 - Schema hardware della soluzione di accesso (Smart Card icon da <https://www.smartcardfocus.com/>)

La **MIFARE Classic® 1K contactless** si basa su NXP MF1 IC S50. Questa tipologia di carta è una buona scelta per applicazioni classiche come la biglietteria dei trasporti pubblici, servizi di fidelizzazione e può essere utilizzata anche per molte altre applicazioni come sistemi di apertura porte e simili.

Il lettore di Smart Card deve essere conforme a standard indicati nello schema di figura 1. Nel mio caso ho utilizzato il lettore **CIE Bit4id miniLector** collegato al Raspberry Pi tramite la porta USB. Il nome tecnico del lettore che ho utilizzato è: BIT4ID miniLector AIR NFC v3.

Al Raspberry Pi è collegato un modulo di quattro relè gestito tramite le porte **GPIO**. Nel nostro scenario, i relè rappresentano gli attuatori necessari per aprire le porte dell'hotel dove siamo ospiti.

Appurato che il nostro scenario è quello di un sistema di accesso per consentire agli ospiti di un hotel di accedere alle proprie stanze attraverso una Smart Card, vediamo quali sono i due processi che portano al raggiungimento di questo obiettivo.

La figura 2 illustra il processo semplificato (in notazione **BPMN**) di ciò che accade quando un ospite viene ricevuto dal personale dell'hotel. Dell'intero processo, solo il **Service Task** (indicato in rosso) sarà l'oggetto dell'implementazione del software.

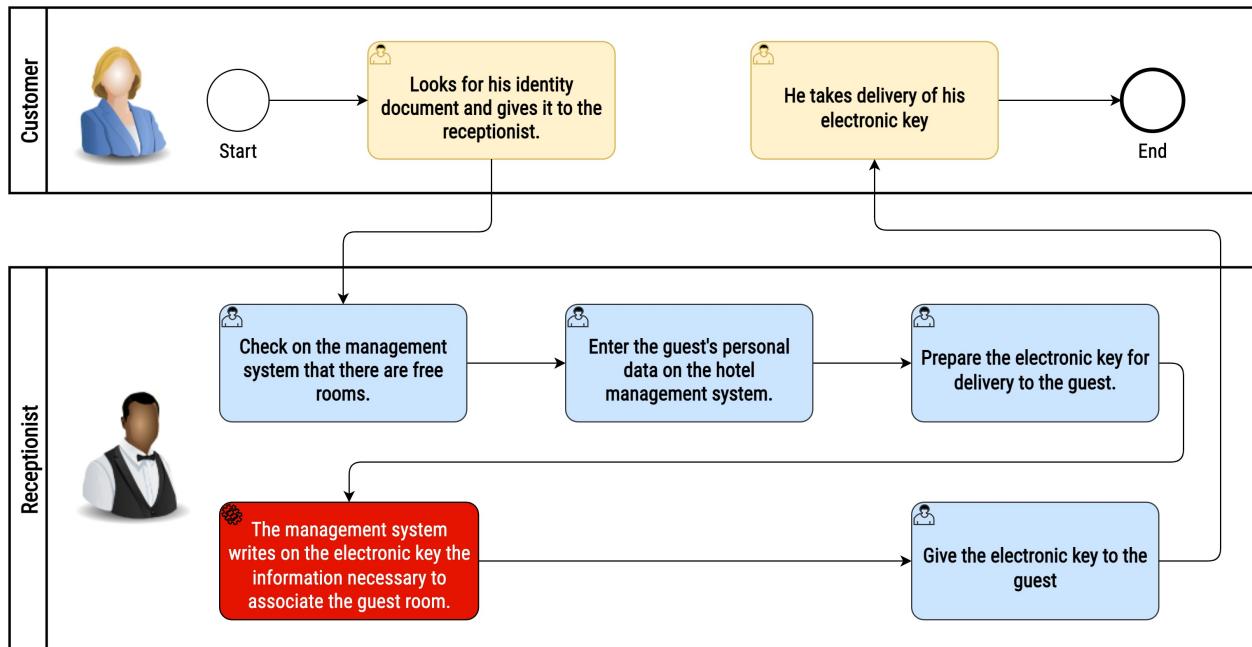


Figura 2 - Processo semplificato di accoglienza dell'ospite in hotel

Il processo di figura 3 mostra invece cosa succede quando l'ospite chiede di entrare nella sua camera tramite l'uso della chiave elettronica appoggiandola sul lettore. Dell'intero processo, solo i Service Task (indicati in rosso) saranno oggetto dell'implementazione del software.

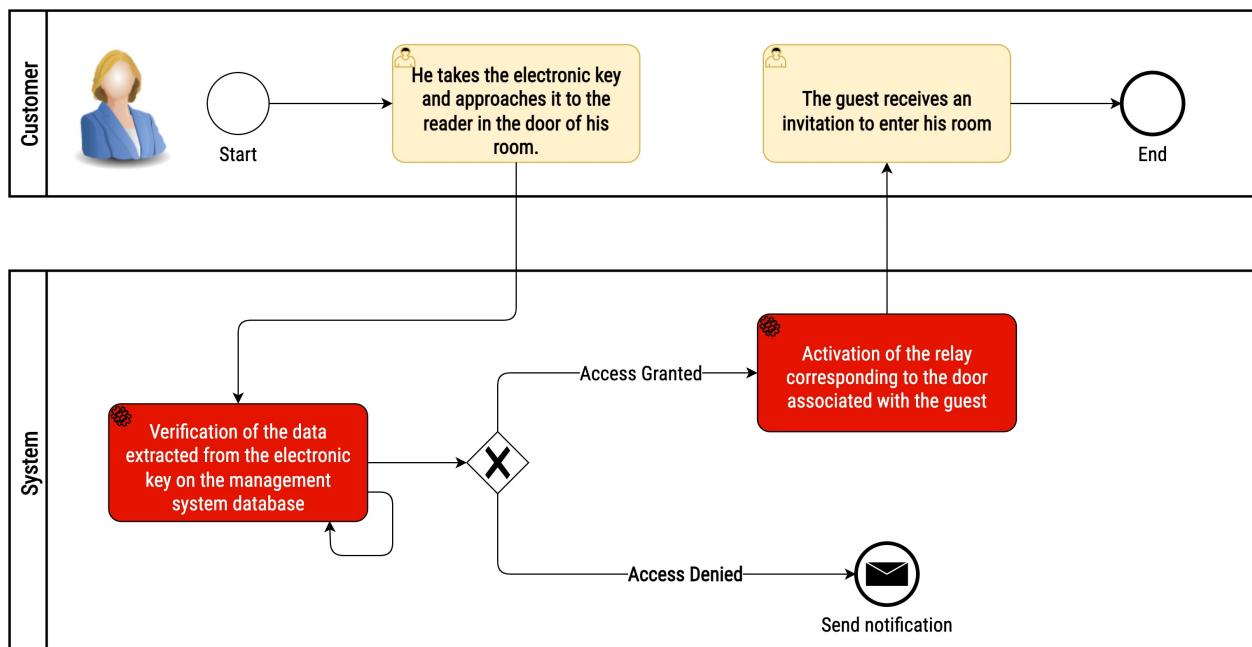


Figura 3 - Processo di accesso alla stanza

Credo che tutti i processi illustrati (in figura 2 e figura 3) in notazione BPMN siano abbastanza esplicativi da non richiedere ulteriori approfondimenti. I Service Task sono gli elementi che saranno oggetto di nostro interesse per l'implementazione del software che fin da questo momento possiamo dividere in due macro componenti le cui responsabilità devono essere:

1. il setup della Smart Card (o chiave elettronica) in fase di registrazione dell'ospite presso la struttura alberghiera. All'interno della Smart Card sarà memorizzato l'identificativo di un documento di riconoscimento dell'ospite, sul database del sistema di gestione dell'albergo saranno invece memorizzati i dati anagrafici insieme ad altri dati necessari per associare la chiave elettronica all'ospite e alla stanza a lui assegnata;
2. la verifica che la chiave elettronica sia abilitata e associata all'ospite e alla stanza a cui consentire l'accesso.

3. Dettagli sulle carte MIFARE Classic 1K

MIFARE è un marchio registrato di [NXP Semiconductors](#). I prodotti MIFARE sono circuiti integrati ampiamente utilizzati per l'utilizzo in Smart Card senza contatto e molte altre applicazioni in tutto il mondo. La gamma MIFARE comprende circuiti integrati per la realizzazione di tessere contactless e lettori per la comunicazione con esse.

Il nome MIFARE racchiude diversi tipi di Smart Card senza contatto, quella utilizzata in questo scenario rientra nel tipo **Classic**. Si tratta di schede di memoria a logica cablata che solo parzialmente sono conformi allo standard [ISO/IEC 14443A](#) (caratteristiche fisiche, potenza e interfaccia del segnale radio, inizializzazione e anticollisione), poiché utilizzano un set di comandi proprietari invece del protocollo [ISO/IEC 14443-4](#) di alto livello e non sono conformi al formato frame [ISO/IEC 14443-3](#) e nelle comunicazioni crittografate usano un protocollo di sicurezza proprietario [NXP \(CRYPTO1\)](#) per l'autenticazione e la crittografia che è stato rotto nel 2008.

Dopo l'ultima frase immagino che vi stiate domandando: **perché scrivere allora un articolo su questa Smart Card?**

La risposta è abbastanza semplice. Questa è una delle [Smart Card a sola memoria](#) più diffuse e semplici da usare, e per il tipo di scenario qui presentato, la possibilità di scoprire "facilmente" la chiave di autenticazione è secondario, inoltre, questo è articolo prettamente didattico.

La MIFARE Classic 1K dispone di 1024 byte di memoria ([EEPROM](#)) suddivisa in 16 segmenti; ogni settore è protetto da due chiavi che vengono chiamate A e B. In tutti i tipi di carta, 16 byte a settore sono riservati alle chiavi e alle condizioni d'accesso e non possono essere utilizzati per i dati dell'utente; inoltre, i primi 16 byte contengono il numero di serie univoco e di sola lettura. In questo modo la memoria disponibile si riduce a 752 byte.

Data l'altissima diffusione di questa tipologia di Smart Card e le versioni "cinesi" esistenti, dubito che non ci siano numeri di serie duplicati, visto anche che i byte dedicati alla memorizzazione del numero di serie sono quattro.

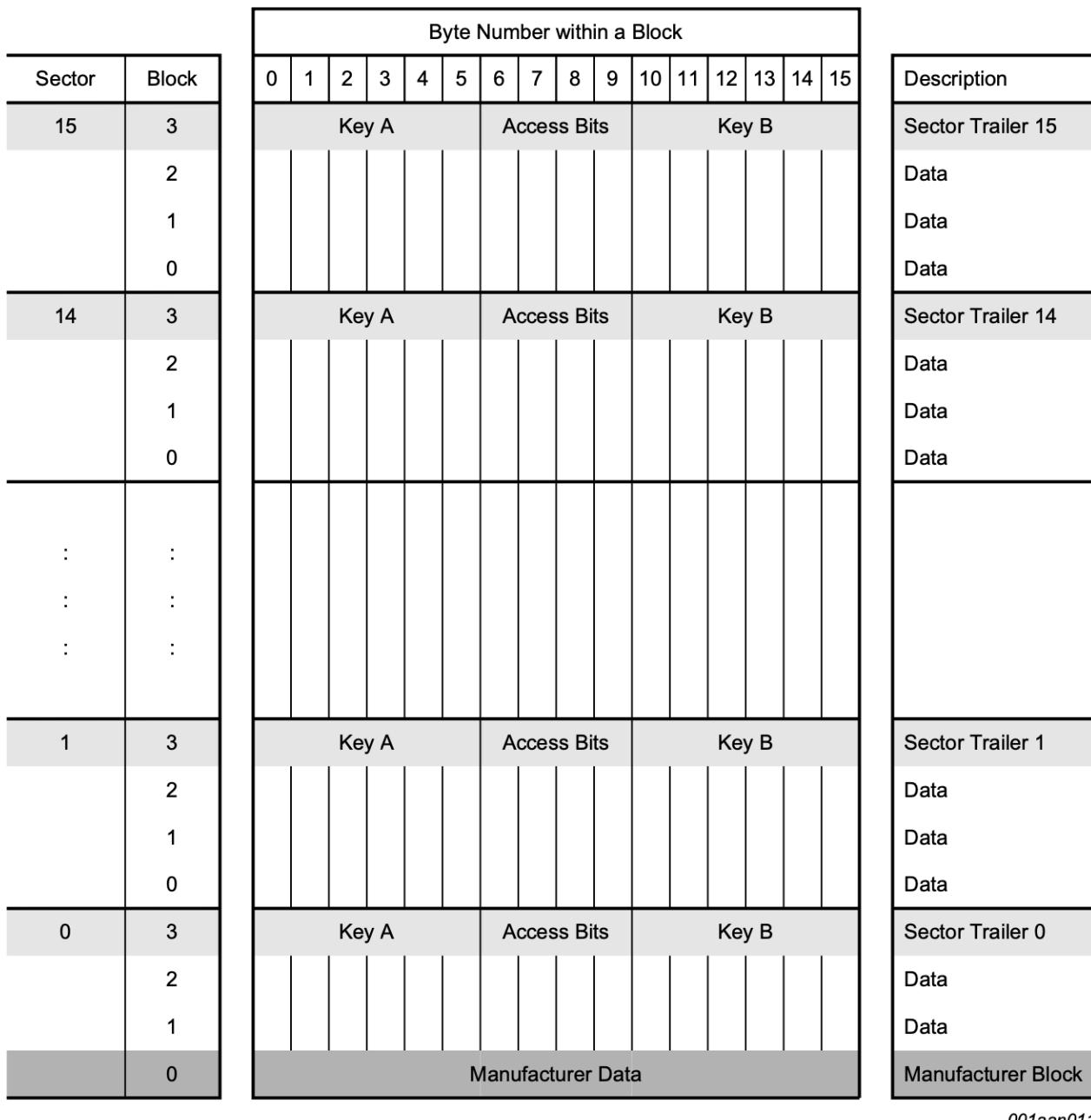


Figura 4 - Struttura della memoria della Mifare Classic 1K (dal datasheet NXP MIFARE Classic EV1 1K
https://www.nxp.com/docs/en/data-sheet/MF1S50YYX_V1.pdf

Prima di poter compiere operazioni di lettura o scrittura sui blocchi di memoria, è necessario eseguire prima un'autenticazione tramite la chiave del settore del blocco. In genere tutte le chiavi (A e B) sono impostate su valore **FFFFFFFFFFFFh** in fase di produzione del chip (e in ogni caso la documentazione fornita al momento dell'acquisto fornisce indicazioni sulle chiavi di accesso e su come eseguire la modifica).

La tabella mostrata in figura 5 rappresenta la mappa della memoria con in evidenza il range degli indirizzi per i **Data Blocks** e i **Trailer Block** per ogni settore. Questa mappa sarà utile nel momento in cui dovremo leggere e scrivere sui data blocks.

MIFARE Classic 1K Memory Map		
Sectors (Total 16 sectors. Each sector consists of 4 consecutive blocks)	Data Blocks (3 blocks, 16 bytes per block)	Trailer Block (1 block, 16 bytes)
Sector 0	00h ~ 02h	03h
Sector 1	04h ~ 06h	07h
...		
Sector 14	38h ~ 0Ah	3Bh
Sector 15	3Ch ~ 3Eh	3Fh

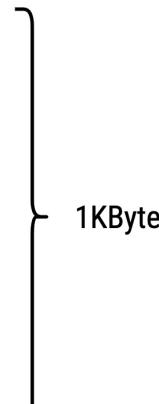


Figura 5 - Memory Map della MIFARE Classic 1K completa di range d'indirizzi dei blocchi

Dopo questa sorvolata sulle caratteristiche principali della MIFARE Classic 1K che sono di nostro interesse per il nostro obiettivo, possiamo andare avanti ed esplorare il modo su come comunicare con la carta per le operazioni di autenticazione, lettura e scrittura.

4. I misteriosi pacchetti APDU

La comunicazione delle informazioni tra la carta e il lettore è resa possibile grazie allo scambio di pacchetti detti **APDU** (**A**pplication **P**rotocol **D**ata **U**nity); essi rappresentano l'unità di comunicazione tra il lettore e la carta e la loro struttura è ben definita da [ISO/IEC 7816-4 Organization, security and commands for interchange](#).

Esistono due tipi di APDU: quelli di comando e quelli di risposta. I primi richiedono un set di attributi attraverso cui il lettore è in grado di sapere quali operazioni compiere e quali dati inviare, i secondi contengono il risultato dell'operazione richiesta con in coda l'esito dell'operazione.

La tabella di figura 6 mostra la struttura dei comandi APDU che attraverso il lettore di Smart Card possiamo inviare alla carta MIFARE.

Struttura dei Comandi APDU (Application Protocol Data Unit)			
Nome attributo	Lunghezza (bytes)	Descrizione	
CLA	1	Classe dell'istruzione. Indica il tipo di comando.	
INS	1	Codice dell'istruzione. Indica il comando specifico (esempio: lettura blocco dati).	
P1-P2	2	Parametri dell'istruzione (esempio: la locazione della chiave di autenticazione)	
L _(c)	0,1 o 3	Codifica il numero N _(c) di byte di dati del comando da eseguire <ul style="list-style-type: none"> • 0 byte denota N_(c)=0 • 1 byte con un valore da 1 a 255 denota N_(c) con la stessa lunghezza • 3 byte, il primo dei quali deve essere 0, denota N_(c) nell'intervallo da 1 a 65 535 (tutti e tre i byte potrebbero non essere zero) 	
		Command Data N _(c) byte dei dati	
		Codifica il numero massimo N _(e) di byte di risposta previsti <ul style="list-style-type: none"> • 0 byte denota N_(e)=0 • 1 byte nell'intervallo da 1 a 255 denota quel valore di N_(e), o 0 denota N_(e)=256 • 2 byte (se nel comando era presente L_(c) esteso) nell'intervallo da 1 a 65 535 denota N_(e) di quel valore, o due zero byte denotano 65 536 • 3 byte (se L_(c) non era presente nel comando), il primo dei quali deve essere 0, denotano N_(e) allo stesso modo di L_(e) a due byte 	
Leggenda e/o note			
L _(e) = expected length (bytes). Indica quanti byte sono previsti (00h = numero di byte massimo) come risposta.			
L _(c) = exact length (bytes). Indica la lunghezza in byte di N _(c) . Per esempio nel caso dell'APDU per caricare la chiave di autenticazione, questo attributo varrà 06h perché la lunghezza della chiave è di sei byte.			

Figura 6 - Struttura dei comandi APDU

La tabella di figura 7 mostra un esempio di comando APDU per ottenere l'UID della MIFARE Classic 1K, la cui lunghezza è pari a 4 byte (vedi attributo Le). A fronte dei comandi inviati, riceviamo sempre una risposta che ha la struttura indicata dalla tabella di figura 8.

Comando APDU per richiedere l'UID					
Comando	Classe	INS	P1	P2	Le
Ottenere l'UID	FFh	CAh	00h	00h	04h

Figura 7 - Comando per ottenere l'UID della Smart Card MIFARE Classic 1K

Struttura Risposta APDU (Application Protocol Data Unit)		
Nome attributo	Lunghezza (bytes)	Descrizione
Response	$N_{(r)}$ (al massimo $N_{(e)}$)	Dati di risposta al comando richiesto
SW1-SW2 (Response Trailer)	2	Risultato del comando in esadecimale (esempio: 90 00 indica il successo dell'esecuzione del comando).

Figura 8 - Struttura di risposta APDU

La tabella di figura 9 mostra la risposta ottenuta a fronte del comando APDU per richiedere l'UID (vedi figura 7) la cui lunghezza dell'UID è pari a 4 byte più 2 byte che segnalano l'esito del comando. L'UID è restituito con il byte meno significativo ([LSB](#)) a sinistra e quello più significativo ([MSB](#)) a destra (quindi usa il sistema [little-endian](#)).

Risposta del comando APDU per richiedere l'UID						
Risposta	Dati					
Risultato	UID (LSB)	UID (MSB)	SW1	SW2
-Leggenda e/o note-						

LSB = *Least Significant Byte* indica il byte meno significativo di un numero composto da più byte.

MSB = *Most Significant Byte* indica il byte più significativo di un numero composto da più byte.

In questo caso per l'UID è usato il cosiddetto sistema **little-endian** per l'ordinamento dei byte, dove si inizia dal byte meno significativo (estremità più piccola) per finire col più significativo.

Figura 9 - Risposta del comando APDU per ottenere l'UID della MIFARE Classic 1K

Confrontando i valori di **SW1** e **SW2** (**Status Word**) con quelli riportati dalla tabella mostrata in figura 10 è possibile evincere se l'esecuzione del comando è andato a buon fine.



Attenzione. I valori di SW1 e SW2 fanno riferimento al comando per ottenere l'UID della carta; questi valori potrebbero essere diversi per altre classi di comando, in particolare nei casi di errore.

Risposta APDU significato SW1 e SW2			
Risultato	SW1	SW2	Descrizione
Successo	90h	00h	Operazione conclusa con successo
Errore	63h	00h	Operazione fallita
Errore	6Ah	81h	Funzione non supportata

Figura 10 - Codici di risposta APDU comando UID

Per maggiori dettagli sui comandi ed eventuali estensioni, consiglio sempre di consultare il datasheet della [MIFARE Classic 1K](#). I comandi APDU di nostro interesse per implementare lo scenario descritto nella parte introduttiva dell'articolo, sono:

- comando per ottenere l'UID della carta;
- comando per leggere i dati da un determinato settore e blocco;
- comando per scrivere dati su un determinato settore e blocco della carta.

5. Cos'è l'Answer to reset o ATR

La prima risposta di una Smart Card inserita in un lettore si chiama **ATR (Answer to reset)**. Lo scopo dell'ATR è descrivere i parametri di comunicazione supportati, la natura e lo stato della carta.

L'ottenimento di un ATR viene spesso utilizzato come prima indicazione che questa sia operativa, e il suo contenuto viene esaminato come prima prova che sia del tipo appropriato per un determinato utilizzo. Il lettore di Smart Card, il driver del lettore e il sistema operativo utilizzeranno questi parametri per stabilire una comunicazione con la scheda.

L'ATR è descritto dallo standard [ISO/IEC 7816-3](#). I primi byte dell'ATR descrivono i parametri elettrici, seguiti da byte che descrivono le interfacce di comunicazione disponibili e i rispettivi parametri. Questi byte di interfaccia sono quindi seguiti da byte storici che non sono standardizzati e sono utili per trasmettere informazioni proprietarie come il tipo di scheda, la versione del software integrato o lo stato della scheda. Infine questi byte storici sono eventualmente seguiti da un byte di checksum.

Potremmo riassumere che l'ATR contiene "un sacco di dati" che ci dicono vita morte e miracoli della Smart Card.

Per esempio, scopriamo di più sull'ATR **3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6A** utilizzando il tool [Smart card ATR parsing](#) sviluppato da [Ludovic Rousseau](#). La figura a seguire mostra le informazioni estratte alcune delle quali:

- tipo di Smart Card e in questo caso si tratta della MIFARE Classic 1K;
- produttore della Smart Card e in questo caso NXP;
- standard tecnologici;
- protocolli di comunicazione e in questo caso T=0 (orientato al byte, che costituisce l'unità minima di informazione scambiata) e T=1 (orientato al blocco di byte, grazie al quale la velocità di accesso è maggiore), protocollo solitamente utilizzato di default quando disponibile e supportato anche dal lettore.

Results

Parsing ATR:

```
3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6A
```

TS = 0x3B	Direct Convention
T0 = 0x8F	Y(1): b1000, K: 15 (historical bytes)
TD(1) = 0x80	Y(i+1) = b1000, Protocol T=0

TD(2) = 0x01	Y(i+1) = b0000, Protocol T=1

Historical bytes	80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00
Category indicator byte: 0x80	(compact TLV data object) Tag: 4, Len: 15 (initial access data) Initial access data: 0C A0 00 00 03 06 03 00 01 00 00 00 00 "....."
TCK = 0x6A	correct checksum

Possibly identified card:

```
3B 8F 80 01 80 4F 0C A0 00 00 03 06 .. 00 01 00 00 00 00 ..
```

MIFARE Classic 1K (as per PCSC std part3)

```
3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 ... 00 00 00 00 ..
```

RFID - ISO 14443 Type A Part 3 (as per PCSC std part3)

```
3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 6A
```

NXP/Philips MIFARE Classic 1K (as per PCSC std part3)

[http://www.nxp.com/#/pip/pip=\[pfp=41863\]|pp=\[t=pfp,i=41863\]](http://www.nxp.com/#/pip/pip=[pfp=41863]|pp=[t=pfp,i=41863])

Oyster card - Transport for London (first-gen)

https://en.wikipedia.org/wiki/Oyster_card

ACOS5/1k Mirfare

vivotech ViVOcard Contactless Test Card

Bangkok BTS Sky SmartPass

Mifare Classic 1K (block 0 re-writeable)

Electric vehicle charging card of the German Telekom, acting as EMSP GetCharge

Electric vehicle charging card of the EMSP Stadtwerke Muenchen (SWM), ladenetz.de, Germany

Electric vehicle charging card of the EMSP EinfachStromLaden of Maingau-Energie, Germany

Scouter carsharing customer card in Germany

<https://scouter.de/>

Figura 11 - Dettagli estratti sull'ATR della Smart Card MIFARE Classic 1K

Ludovic Rousseau ha fatto un ottimo lavoro tracciando dal 2002 un [gran numero di ATR](#) costruendo un vero e proprio database. Così facendo è possibile identificare una Smart Card dato il suo ATR. All'interno della lista sono presenti anche le "nostrane Smart Card" come la **TS-CNS (Tessera Sanitaria - Carta Nazionale Servizi)** e **CIE (Carta d'Identità Elettronica)**.

È possibile utilizzare il comando `pcsc_scan` per ottenere informazioni di dettaglio sulla Smart Card, le stesse illustrate in figura 11. La figura 12 mostra l'output del comando menzionato da cui è possibile dedurre che la Smart Card analizzata è una CIE.



The image shows a screenshot of a terminal window on a Mac OS X desktop. The terminal is running the command `pcsc_scan`. The output shows the reader is a BIT4ID minilector AIR NFC v3. It scans for readers and finds one at address 0. The card state is shown as 'Card removed' at 22:26:00 on March 7, 2022. The card was inserted at 22:26:09. The ATR (Application Tracing Reference) is listed as `3B 8F 80 01 80 31 80 65 B0 85 04 00 11 12 0F FF 82 90 00 8A`. The output then provides detailed analysis of this ATR, identifying it as a Direct Convention (TS = 3B), historical bytes (T0 = 8F), and protocol parameters (TD(1) = 80, TD(2) = 01). It also lists historical bytes (80 31 80 65 B0 85 04 00 11 12 0F FF 82 90 00 80), category indicator byte (80), and various card service data bytes (Tag: 3, 6, 8, etc.). Finally, it identifies the card as a Dutch driver licence (eID) and provides a link to its Wikipedia page: https://en.wikipedia.org/wiki/Italian_electronic_identity_card.

```

Using reader plugin play mechanism ~ (-zsh)
Scanning present readers...
0: BIT4ID minilector AIR NFC v3 [minilector AIR DI v3 CLESS] 00 00

Mon Mar 7 22:26:00 2022
Reader 0: BIT4ID minilector AIR NFC v3 [minilector AIR DI v3 CLESS] 00 00
Event number: 0
Card state: Card removed

Mon Mar 7 22:26:09 2022
Reader 0: BIT4ID minilector AIR NFC v3 [minilector AIR DI v3 CLESS] 00 00
Event number: 1
Card state: Card inserted,
ATR: 3B 8F 80 01 80 31 80 65 B0 85 04 00 11 12 0F FF 82 90 00 8A
+ TS = 3B --> Direct Convention
+ T0 = 8F, Y(1): 1000, K: 15 (historical bytes)
TD(1) = 80 --> Y(1+1) = 1000, Protocol T = 0
TD(2) = 01 --> Y(1+1) = 0000, Protocol T = 1

+ Historical bytes: 80 31 80 65 B0 85 04 00 11 12 0F FF 82 90 00
Category indicator byte: 80 (compact TLV data object)
Tag: 3, len: 1 (card service data byte)
Card service data byte: 80
- Application selection: by full DF name
- EF.DIR and EF.ATR access services: by GET RECORD(s) command
- Card with MF
Tag: 6, len: 5 (pre-issuing data)
Data: B0 85 04 00 11
Tag: 1, len: 2 (country code, ISO 3166-1)
Country code: 0F FF
Tag: 8, len: 2 (status indicator)
SW: 9000
+ TCK = 8A (correct checksum)

Possibly identified card (using /usr/share/pcsc/smartcard.list.txt):
3B 8F 80 01 80 31 80 65 B0 85 04 00 11 12 0F FF 82 90 00 8A
Dutch driver licence (eID)
Italian electronic identity card (eID)
https://en.wikipedia.org/wiki/Italian\_electronic\_identity\_card

```

Figura 12 - Esempio di output del comando `pcsc_scan` che mostra le informazioni estratte dalla Smart Card, in questo caso CIE

6. Requisiti hardware

Realizzare lo scenario descritto richiede il recupero di una serie di "pezzi di ferro".

1. [Raspberry Pi 4 Model B 8GByte RAM](#)
2. [MicroSD Card \(min 8GByte\)](#)
3. [Elegoo 4 Channel DC 5V Modulo Relay](#)
4. [Jumper Wire Cable Cavo F2F female to female](#)
5. [Bit4id miniLector CIE](#)
6. [Mifare Classic 1K](#)
7. [Mifare Classic 1K RFID Tag \(opzionale\)](#)

Per quanto riguarda il Raspberry Pi è possibile optare per la versione con 4GByte di RAM ma è importante installare come sistema operativo l'ultima versione di [Raspberry Pi OS](#) affinché sia supportato il lettore di Smart Card Bit4id miniLector CIE e questo grazie alla versione 1.4.34-1 della libreria libccid (check via `apt info libccid`).

```
amusarra@raspberrypi-amusarra:~ $ apt info libccid
Package: libccid
Version: 1.4.34-1
Priority: optional
Section: libs
Source: ccid
Maintainer: Ludovic Rousseau <rousseau@debian.org>
Installed-Size: 553 kB
Provides: pcsc-ifd-handler
Depends: libc6 (>= 2.17), libusb-1.0-0 (>= 2:1.0.9)
Suggests: pcmciautils
Homepage: https://ccid.apdu.fr/
Tag: role::shared-lib
Download-Size: 331 kB
APT-Manual-Installed: no
APT-Sources: http://deb.debian.org/debian bullseye/main arm64 Packages
Description: PC/SC driver for USB CCID smart card readers
This library provides a PC/SC IFD handler implementation for the USB smart
card drivers compliant to the CCID protocol.
.
This package is needed to communicate with the CCID smartcard readers through
the PC/SC Lite resource manager (pcscd).
.
For an exhaustive list of supported reader see
http://pcslite.alioth.debian.org/section.html
.
This driver also supports the GemPC Twin connected to a serial port and
the GemPC Card (PCMCIA, through the suggested pcmciautils package) and
Gemplus GemPC Express (Express54 card).

amusarra@raspberrypi-amusarra:~ $
```

Figura 13 - Dettaglio del driver libccid necessario per interoperabilità tra Sistema Operativo e lettore di Smart Card

Nel caso in cui abbiate già un lettore di Smart Card o vogliate acquistarne un diverso modello, consiglio di consultare la [lista dei lettori di Smart Card supportati dal driver libccid](#).

7. Requisiti software

Così come abbiamo bisogno dell'hardware, è necessario che siano soddisfatti una serie di requisiti software, quali:

1. [Raspberry Pi OS \(64bit\)](#)
2. [Python 3.9.x](#) (distribuito e installato di default con Raspberry Pi OS)
3. [Docker 20.10.12](#)
4. [Development Tools \(make, gcc\) \(install or update via sudo apt install build-essential\)](#)

Per questo genere di scenari non è assolutamente necessario provvedere all'installazione del sistema operativo in versione Desktop, consiglio pertanto di preparare e usare l'immagine di [Raspberry Pi OS Lite \(64bit\)](#). Per coloro che avessero bisogno di una guida su come installare questo sistema operativo, consiglio di seguire la guida ufficiale [Installing the Operating System](#).

L'installazione di Docker potrebbe essere anche opzionale; personalmente preferisco installare il database in forma di container. Più in avanti vedremo quale database ho scelto per questa soluzione.

Per approfondimenti sul tema Docker, consiglio la lettura del libro [Docker: Sviluppare e rilasciare software tramite container](#) di [Serena Sensini](#) e la visione delle [Pillole di Docker](#) sul canale YouTube di [Mauro Cicolella](#).

8. Schema elettrico della soluzione

Una volta ottenuto l'hardware (indicato in precedenza), possiamo procedere con il collegamento del modulo dei quattro relè all'interfaccia GPIO con l'ausilio dei jumper femmina-femmina.

La figura 14 mostra lo schema elettrico di collegamento tra il modulo a quattro relè e il Raspberry Pi. Ricordo che il lettore di Smart Card è collegato via USB al Raspberry Pi. Utilizzando i jumper e seguendo lo schema, il risultato sarà assicurato. È preferibile eseguire l'operazione di collegamento lasciando il proprio Raspberry Pi spento e scollegato dalla fonte di alimentazione.

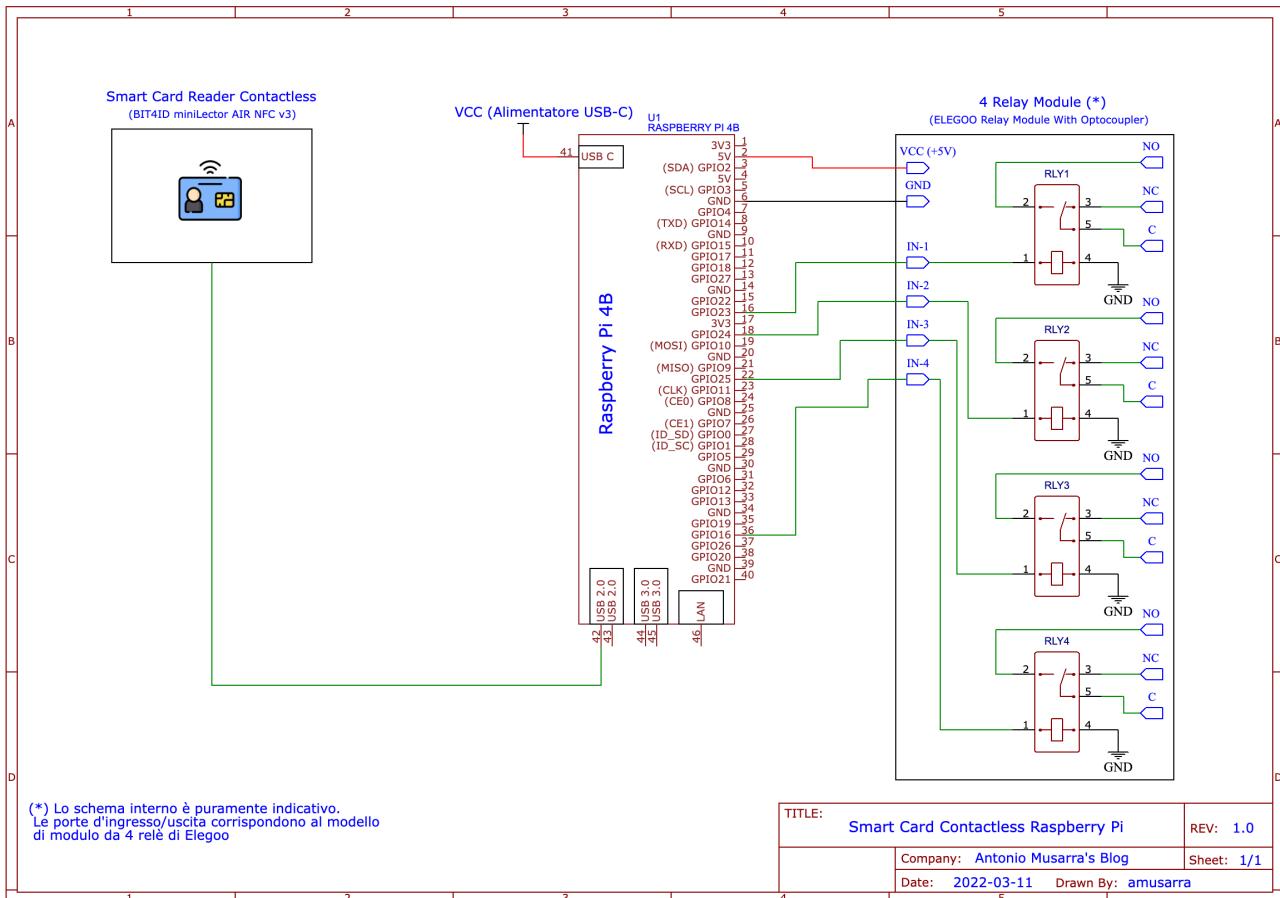


Figura 14 - Schema elettrico di collegamento tra il Raspberry Pi e il modulo da quattro relè

Per ogni dubbio sulla disposizione dei pin del connettore J8 del Raspberry, consultare l'output del comando `pinout` (vedi figura 15) prima di procedere con l'operazione di collegamento, oppure, puntate il vostro browser su pinout.xyz.

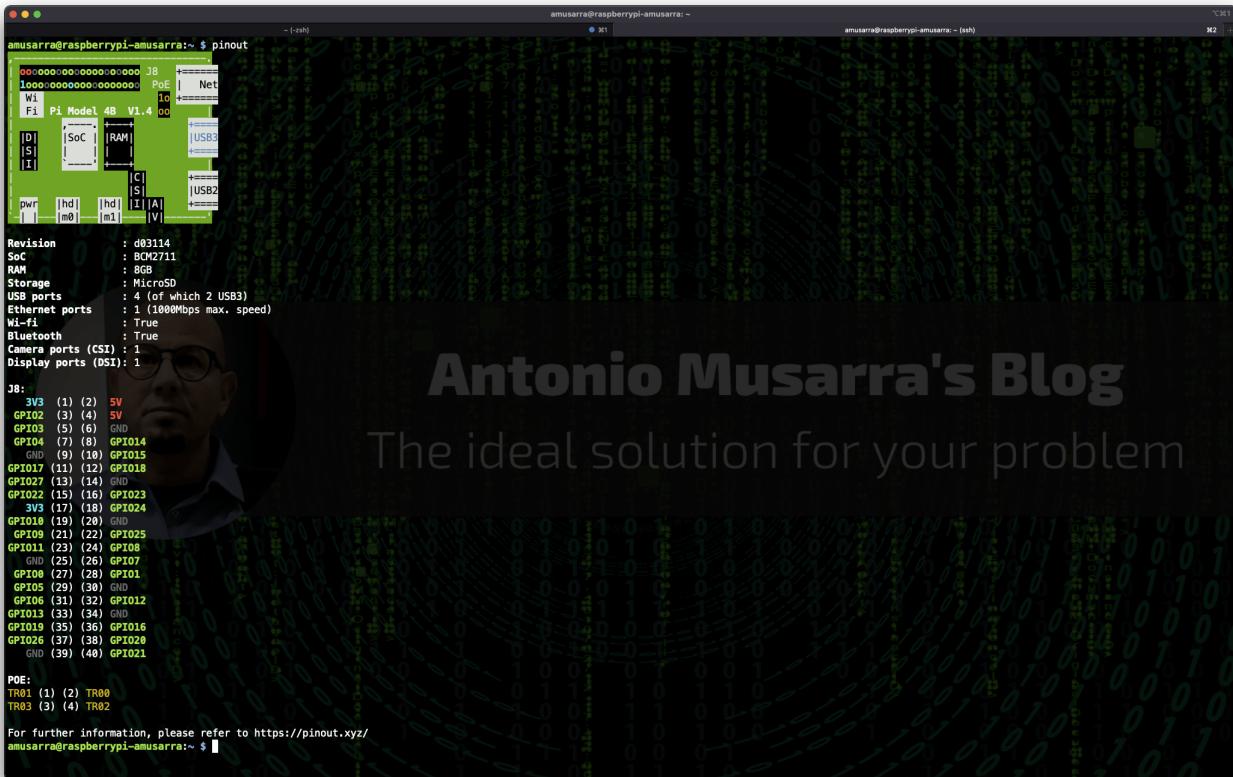


Figura 15 - Output del comando pinout, utile per verificare la piedinatura del GPIO e altre informazioni sul layout hardware e componenti

9. Progettare il software

È arrivato il momento di saltare dall'hardware al software cercando di delineare ciò che dovremo implementare con l'obiettivo di soddisfare i requisiti espressi in forma di diagrammi BPMN (vedi figura 2 e figura 3). Ricordo che solo quanto espresso dai Service Task deve essere realizzato in forma di software.

Cerchiamo di identificare il **cosa** dovrà essere soddisfatto dal punto di vista funzionale per poi passare al **come** gli aspetti funzionali dovranno essere implementati. La tabella mostrata in figura 16 (trasposizione di una classica Mind Map), descrive gli aspetti funzionali che il software che andremo a realizzare deve implementare.

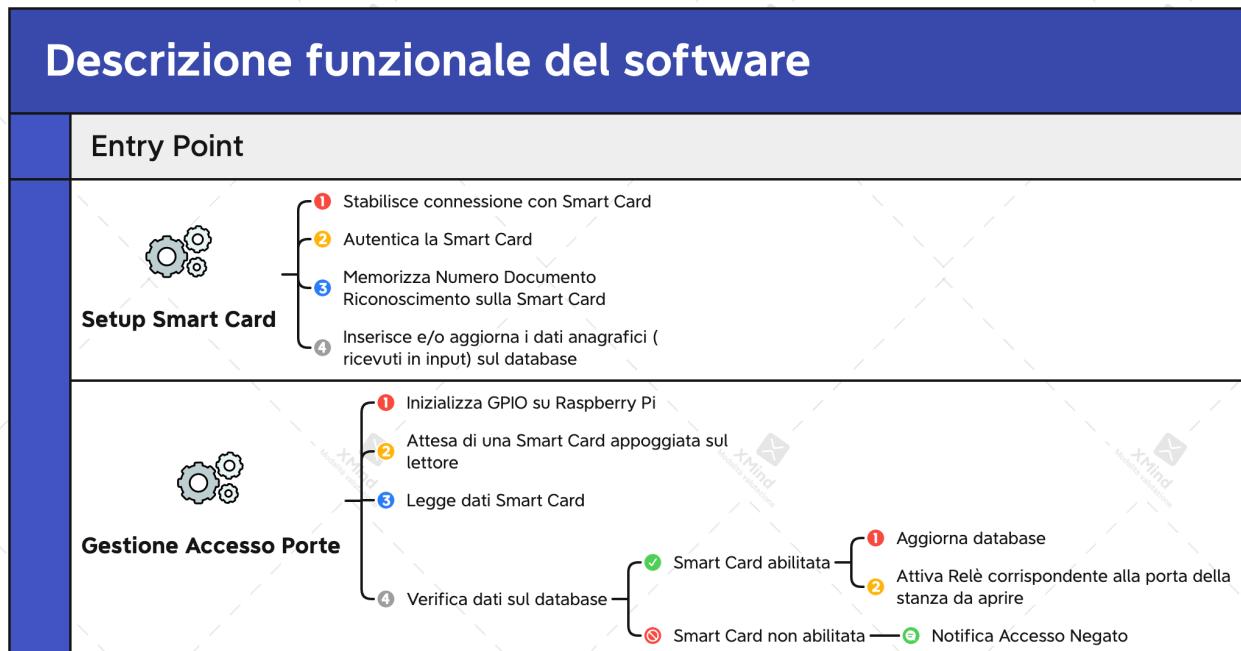


Figura 16 - Mind Map sugli aspetti funzionali che il software dovrà implementare

Il software deve prevedere due entry point, ed esattamente quelli indicati all'interno della tabella di figura 16, le cui responsabilità devo essere:

- **Setup Smart Card**: entry point la cui principale responsabilità è quella di "formattare" la Smart Card (o chiave elettronica) con i dati personali dell'ospite e in questo caso il numero del documento d'identità, inoltre, creare l'associazione tra chiave elettronica, ospite e stanza a lui assegnata. Una volta eseguiti i task di questo entry point (vedi figura 16), la Smart Card potrà essere consegnata all'ospite.
- **Gestione Accesso Porte** entry point la cui principale responsabilità è quella di garantire l'accesso (apertura della porta della stanza) solo a coloro che hanno in possesso una Smart Card valida, aggiornando i dati di accesso sul database. La Mind Map di figura 16 mostra i singoli task che devono essere eseguiti da questo specifico entry point.

Più volte è stato fatto riferimento all'esistenza di un database, bene, adesso cerchiamo di capire quali sono i dati da esso trattati e le responsabilità dei due entry point in merito al trattamento di questi dati.

La tabella di figura 17 mostra la struttura dati del documento che descrive l'associazione ospite, Smart Card e stanza assegnata e traccia i dati degli eventi d'inizializzazione e accesso.

Guest Access Document			
createDate	Scope		setup_smart_card (w)
	Tipo		Date Time + TimeZone (formato ISO)
	Descrizione		Data/ora di quando il documento è stato creato
modifiedDate	Scope		access_via_smart_card (w)
	Tipo		Date Time + TimeZone (formato ISO)
	Descrizione		Data/ora di quando il documento è stato creato e/o aggiornato
firstAccess	Scope		access_via_smart_card (w)
	Tipo		Date Time + TimeZone (formato ISO)
	Descrizione		Data/ora di quando è stato eseguito il primo accesso alla stanza assegnata (aperta la porta)
lastAccess	Scope		access_via_smart_card (w)
	Tipo		Date Time + TimeZone (formato ISO)
	Descrizione		Data/ora di quando è stato eseguito l'ultimo accesso alla stanza assegnata.
smartCardInitDate	Scope		setup_smart_card (w)
	Tipo		Date Time + TimeZone (formato ISO)
	Descrizione		Data/ora di quando la Smart Card è stata formattata o inizializzata
smartCardId	Scope		setup_smart_card (w)
			access_via_smart_card (r)
	Tipo		Stringa
	Descrizione		UID (4 byte) della Smart Card. Valore imposto dal produttore della carta
smartCardEnabled	Scope		setup_smart_card (w)
			access_via_smart_card (r)
	Tipo		Boolean
	Descrizione		Indica se la Smart Card è abilitata (true - apre porta) o non abilita (false - nega accesso).
documentId	Scope		setup_smart_card (w)
			access_via_smart_card (r)
	Tipo		String
	Descrizione		Numero del documento d'identità dell'ospite a cui è assegnata la Smart Card (o chiave elettronica)
roomNumber	Scope		setup_smart_card (w)
			access_via_smart_card (r)
	Tipo		Intero
	Descrizione		Numero della stanza assegnata all'ospite
countAccess	Scope		setup_smart_card (w)
			access_via_smart_card (rw)
	Tipo		Intero
	Descrizione		Numero degli accessi eseguiti con successo (numero di volte per cui la porta è stata aperta)
firstName	Scope		setup_smart_card (w)
	Tipo		Stringa
	Descrizione		Nome dell'ospite
lastName	Scope		setup_smart_card (w)
	Tipo		Stringa
	Descrizione		Cognome dell'ospite

Figura 17 - Struttura dati che descrive l'associazione ospite, smart card e stanza assegnata e traccia i dati gli eventi di inizializzazione e accesso

La tabella mostra un'informazione importante per ogni attributo del documento, ovvero, lo scope, quindi l'entry point, con evidenza del diritto di accesso: (r -read) lettura o (w - write) scrittura.

10. Implementare il software

Adesso che il **cosa** è chiaro, vediamo il **come**, rispondendo alle domande a seguire:

1. Quale linguaggio usare per implementare lo scenario che abbiamo descritto all'inizio dell'articolo? **Python**
2. Esiste qualche libreria Python per la costruzione di applicazioni basate su Smart Card? [Pyscard](#)
3. Quale tipo di database usare per la memorizzazione dei dati? [MongoDB](#) ([NoSQL](#) database)
4. Esiste qualche libreria Python per operare con MongoDB? [PyMongo](#)
5. Esiste qualche libreria Python per interagire con l'interfaccia GPIO del Raspberry Pi? [RPi.GPIO](#)

Da questa lista le cose da fare sono parecchie e lo spazio per vederle tutte nel dettaglio in questo articolo non c'è. Direi quindi di focalizzare la nostra attenzione sui componenti da realizzare rimanendo ad alto livello, senza scendere nello specifico del codice.



Non vi allarmate! Il progetto è già stato sviluppato prima di scrivere questo articolo e disponibile sul mio repository GitHub [Smart Card Contactless Raspberry Pi](#).



Nota su MongoDB. È necessario preparare un'istanza di MongoDB sul proprio Raspberry Pi affinché la soluzione funzioni nel modo corretto. Le scelte sono due: container docker o installazione classica direttamente sul sistema operativo. Personalmente ho scelto la strada di Docker (vedi 6. Requisiti Software), per cui, una volta [installato Docker sul proprio Raspberry Pi OS](#), i passi per tirare su un'istanza MongoDB 4.4.12 sono quelli indicati a seguire.

Console 1 - Pull dell'immagine MongoDB 4.4.12 e run del container

```
# 1. Docker pull e run MongoDB 4.4.12
docker pull mongo:4.4.12
docker run -d -p 27017-27019:27017-27019 --name mongodb mongo:4.4.12

# 2. Verifica che l'istanza sia su e accesso alla console bash (task opzionale)
docker exec -it mongodb bash

# 3. Avvio della console MongoDB (task opzionale)
root@0d21da235b0d:/# mongo
```

Nel caso in cui preferiate perseguiere la seconda strada, il blog di MongoDB riporta la procedura completa sull'articolo [Install & Configure MongoDB on the Raspberry Pi](#).

10.1. Cos'è Pyscard

Pyscard, Python Smart Card library, è un modulo Python che aggiunge il supporto per le Smart Card facilitando la costruzione di applicazioni che hanno la necessità di utilizzare la tecnologia delle Smart Card.

Pyscard supporta la piattaforma Microsoft Windows utilizzando i componenti **Microsoft Smart Card Base**, **Linux** e **macOS** utilizzando **PCSC-lite**. Il diagramma di figura 18 mostra l'architettura Pyscard (box in verde). Il modulo <https://pyscard.sourceforge.io/epydoc/smartcard.scard.html> [smartcard.scard] è il wrapper di WinSCard API (**smart card base components**). Il modulo **smartcard** è un vero e proprio framework costruito su PC/SC API.

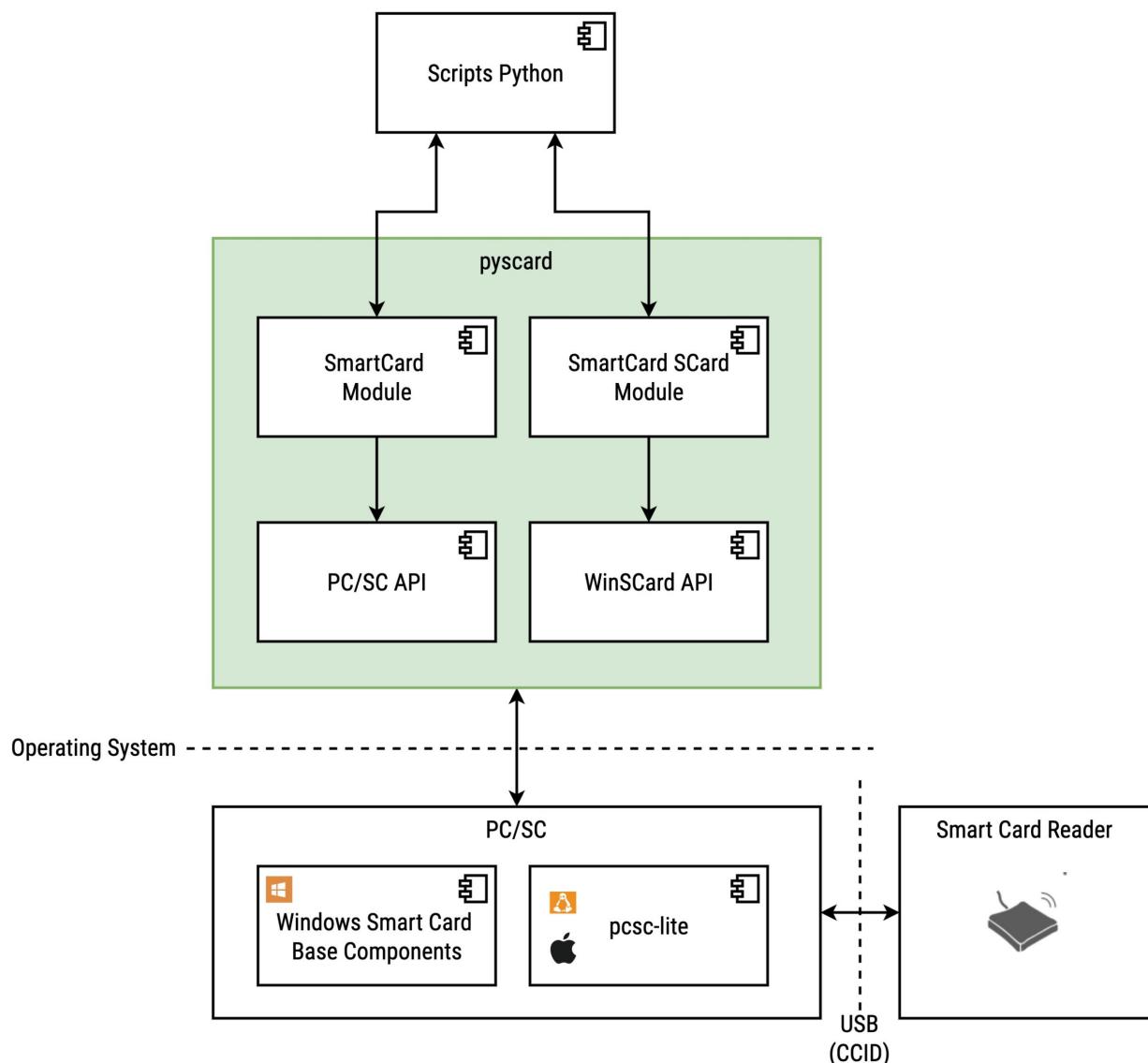


Figura 18 - Architettura di Pyscard

Pyscard sarà di aiuto per:

1. Eseguire la connessione al lettore di Smart Card e alla MIFARE Classic 1K
2. Filtrare le connessioni alle sole MIFARE Classic 1K tramite l'ATR
3. Inviare i comandi APDU
4. Ricevere le risposte ai comandi APDU
5. Intercettare gli eventi di aggiunta e rimozione Smart Card dal lettore

10.2. Struttura del progetto

Adesso che abbiamo visto dalla superficie cos'è Pyscard e per cosa sarà utile, cerchiamo di disegnare la struttura del nostro progetto che implementerà lo scenario introdotto a inizio articolo.

Il diagramma (in notazione UML) di figura 19 mostra package, componenti e classi utilizzati per l'implementazione dello scenario che abbiamo descritto a inizio articolo. Gli elementi del diagramma del nostro progetto sono quelli evidenziati con il colore giallo ocra e in particolare:

Classi

- [ManageRelay](#) (manage_relay.py). Classe che gestisce attraverso l'interfaccia GPIO l'attivazione e disattivazione dei quattro relè, nonché l'inizializzazione della stessa interfaccia GPIO.
- [SmartCardAccessCrud](#) (smart_card_access_crud.py). Classe che gestisce le operazioni CRUD sul database NoSQL MongoDB.
- [MifareClassicInterface](#) (mifare_interface.py). Classe che gestisce la comunicazione con il lettore di Smart Card e di conseguenza d'interagire con la Smart Card connessa. Le operazioni gestite sono: autenticazione, lettura ATR, lettura UID, lettura dalla memoria della carta e scrittura sulla memoria della carta.

Componenti

- [Setup Smart Card](#) (setup_smart_card.py). Script Python che rappresenta l'entry point responsabile del setup della Smart Card e memorizzazione dei dati anagrafici dell'ospite sul database MongoDB. Questo script deve ricevere in input i parametri necessari per eseguire il setup della Smart Card.
- [Access via Smart Card](#) (access_via_smart_card.py). Script Python che rappresenta l'entry point responsabile di governare l'apertura delle porte (attraverso l'attivazione dei relè) sulla base della validazione dei dati estratti dalla Smart Card. Questo script deve ricevere in input la chiave di autenticazione della Smart Card.

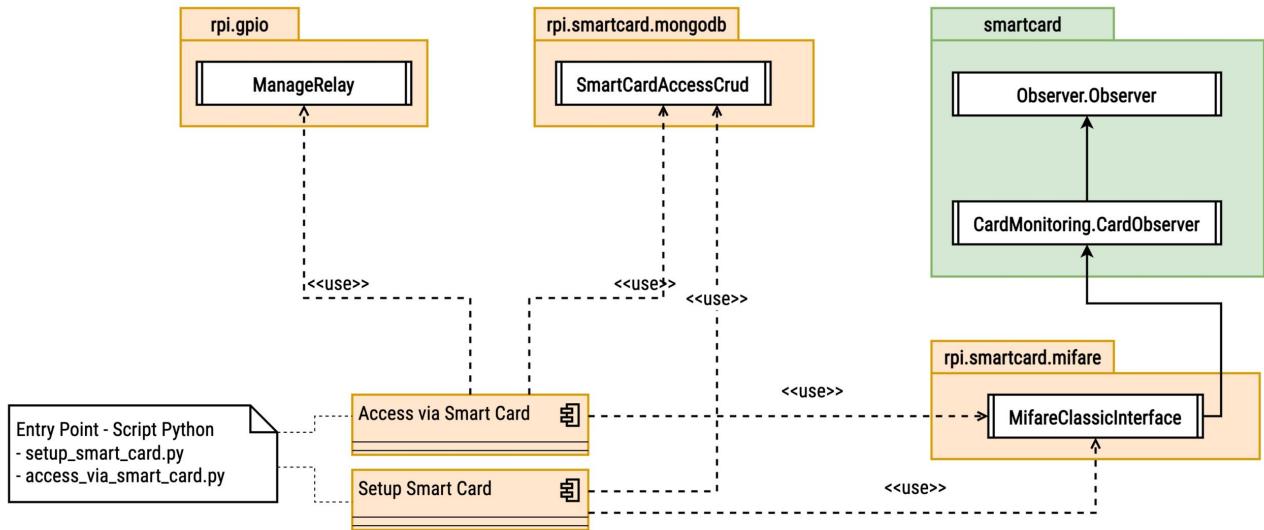


Figura 19 - Package/Component/Class diagram del progetto Smart Card Contactless Raspberry Pi

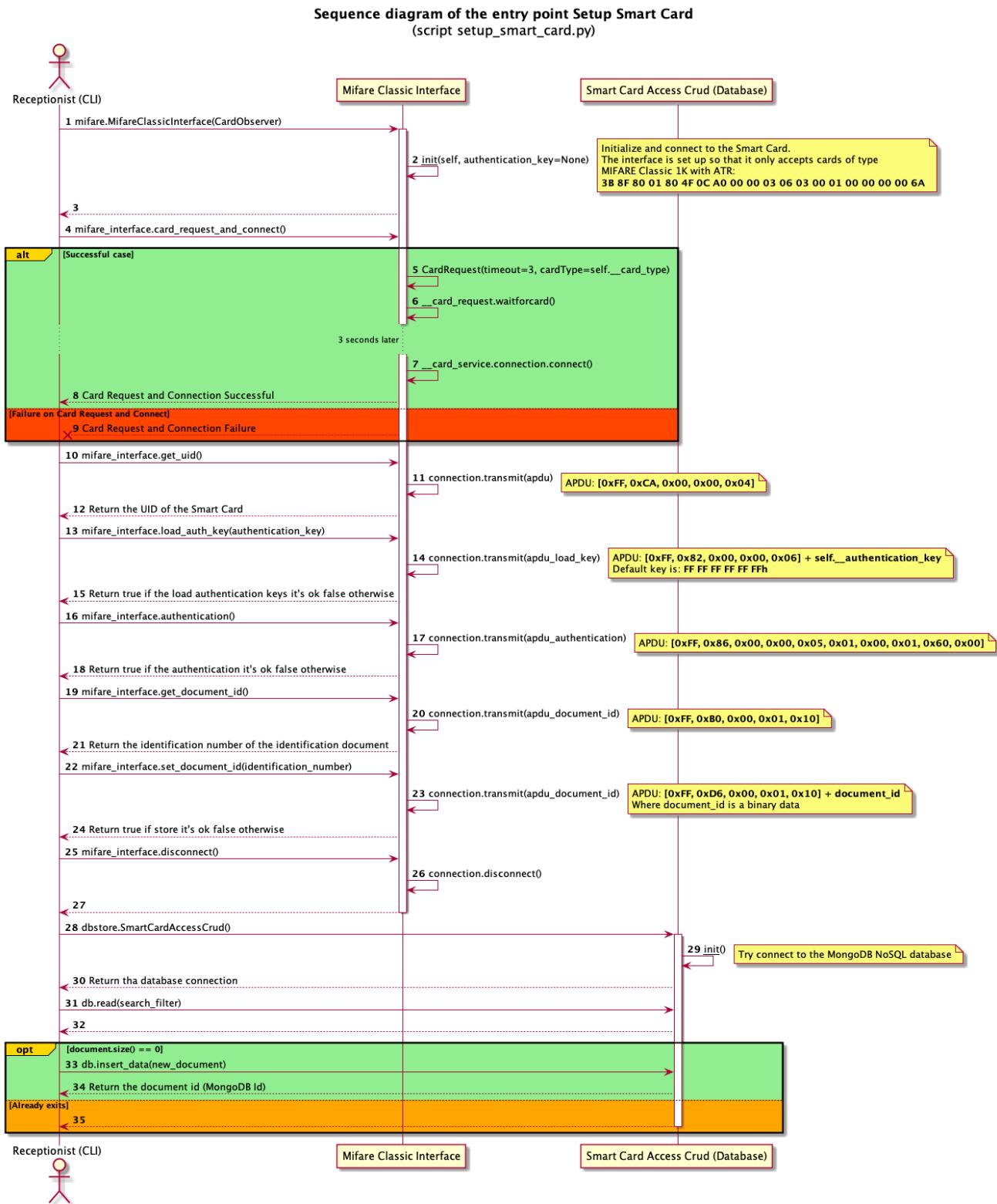
Il package **smartcard** (in verde) fa parte di Pyscard, e i componenti all'interno sono utilizzati per sfruttare il monitoraggio delle Smart Card. Utilizzando l'interfaccia **CardObserver** siamo in grado di poter sapere quando la Smart Card viene aggiunta (appoggiata sul lettore) o rimossa (dal raggio di azione del segnale RF).

Il sequence diagram di figura 20 descrive lo scenario del setup della Smart Card (vedi Figura 2 - Processo semplificato di accoglienza dell'ospite in hotel), mostrando le azioni principali e gli eventuali flussi alternativi. Sempre da questo diagramma possiamo vedere le relazioni che intercorrono, in termini di messaggi, tra i vari attori, quest'ultimi sono stati descritti in precedenza e mostrati nel diagramma di figura 19.

Le azioni decisamente più importanti sono quelle che riguardano l'invio dei comandi APDU, che riassumendo sono:

1. [Load Authentication Key sulla memoria volatile](#)
2. [Autenticazione per accedere al primo blocco del settore uno della MIFARE Classic 1K](#)
3. [Richiesta UID](#)
4. [Lettura dell'eventuale precedente documentId \(numero identificativo del documento di riconoscimento\)](#)
5. [Salvataggio del documentId](#)

I link della precedente lista riportano direttamente sul metodo specifico della classe **Mifare Classic Interface** all'interno del quale è specificato il contenuto dell'APDU inviata.

*Figura 20 - Diagramma di sequenza dell'entry point Setup Smart Card*

Nella fase di setup della Smart Card, prima di inserire i dati anagrafici dell'ospite sul database, viene eseguita la query utilizzando il filtro `{"smartCardId": "f{uid}", "documentId": f"{identification_number}"}`, in modo che sia possibile verificare che non esista già un documento con l'associazione smartCardId e documentId (fare riferimento allo step 33 del sequence diagram di figura 20).

Il sequence diagram di figura 21 descrive lo scenario di Accesso alla stanza tramite Smart Card

(vedi Figura 3 - Processo di accesso alla stanza), mostrando le azioni principali e gli eventuali flussi alternativi. Sempre da questo diagramma possiamo vedere le relazioni che intercorrono, in termini di messaggi, tra i vari attori, quest'ultimi sono stati descritti in precedenza e mostrati nel diagramma di figura 19.

In questo diagramma, tra gli attori in gioco abbiamo anche il componente **Card Monitor** di Pyscard, che abbiamo utilizzato per aggiungere la nostra interfaccia **Mifare Classic Interface** come observable, in questo modo il Card Monitor richiamerà il metodo `update(self, observable, actions)` nel momento in cui si verificheranno gli eventi di aggiunta o rimozione della Smart Card. Il metodo update implementa quanto descritto in Figura 16 - Mind Map sugli aspetti funzionali che il software dovrà implementare, di cui le azioni sono evidenti sul sequence diagram a seguire.

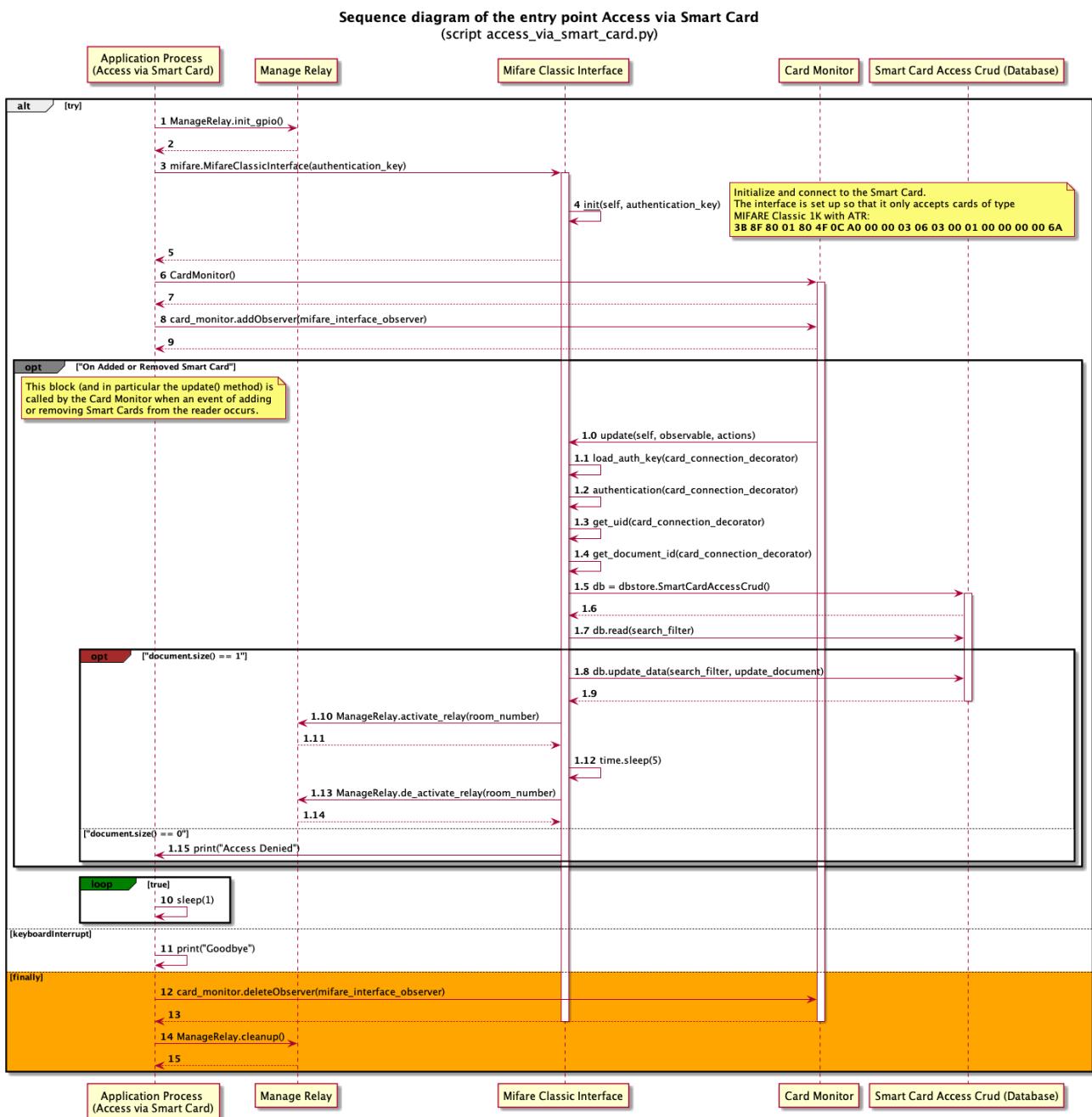


Figura 21 - Diagramma di sequenza per l'entry point Access via Smart Card

I dati sono estratti dal database MongoDB utilizzando il filtro `{"smartCardId": f"{{uid}}", "documentId":f{{identification_number}}", "smartCardEnabled": "true"}`, nel caso non ci siano documenti che rispettino questo filtro, il sistema negherà l'accesso, alternativamente, sarà aggiornato il documento su MongoDB e attivato il relè corrispondente al numero della stanza associato all'ospite (fare riferimento allo step 1.7 del sequence diagram di figura 21).

Molto bene! Una volta descritti i sequence diagram dei nostri due scenari, direi che potremmo passare all'azione, ovvero, eseguire il deploy dell'applicazione sul Raspberry Pi 4.

11. Deploy & Test

Ci siamo! È arrivato il momento d'installare il progetto software sul Raspberry Pi e verificare che tutto funzioni così per com'è stato ideato. Il deployment diagram della figura a seguire mostra tutti i componenti del nostro sistema di accesso.

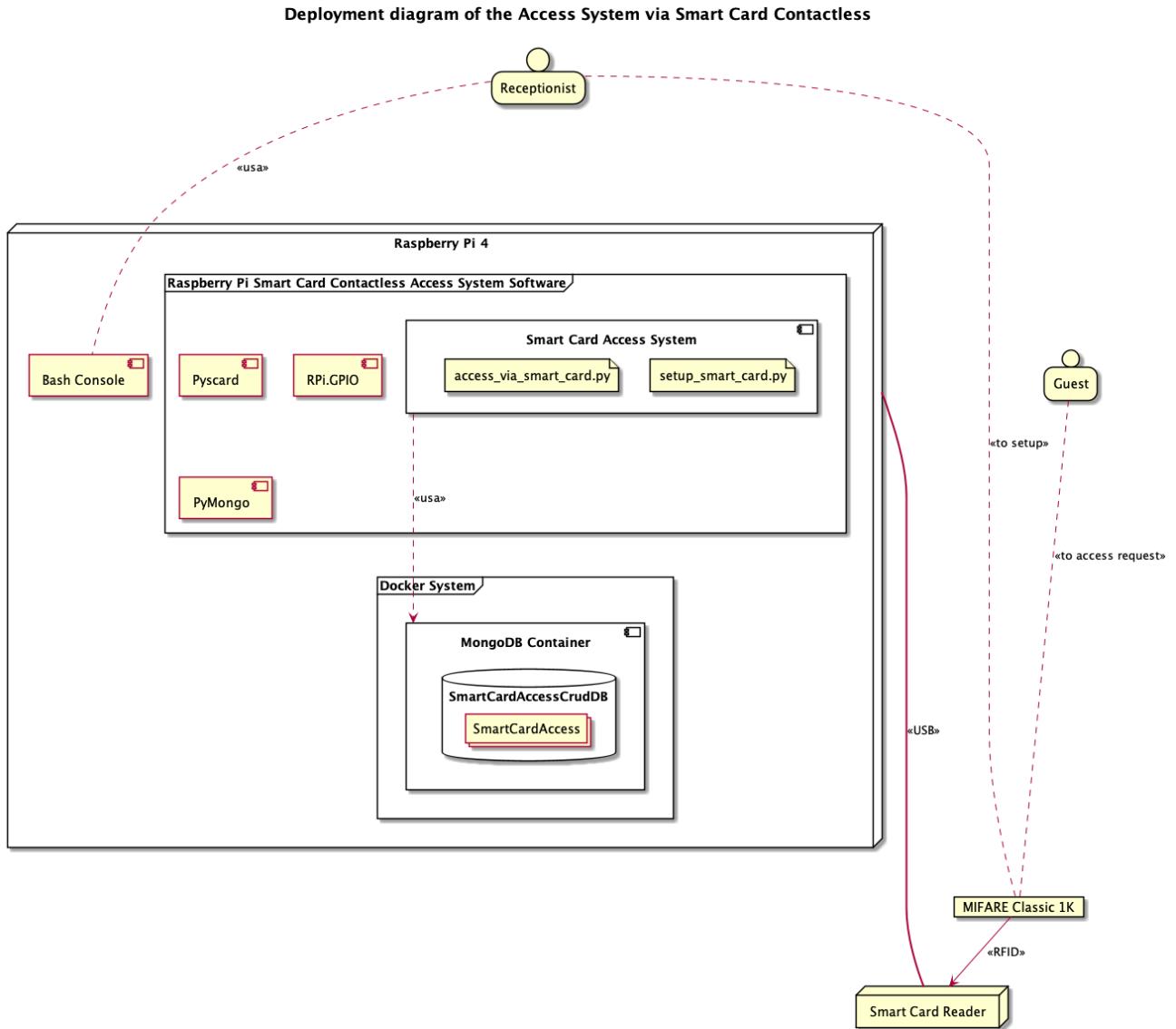


Figura 22 - Deployment diagram del sistema di accesso via Smart Card Contactless su Raspberry Pi

Assumiamo a questo punto che tutti i requisiti software indicati in precedenza siano tutti soddisfatti (fare riferimento a 6. Requisiti Software). Per installare il progetto software sul Raspberry Pi occorre seguire i seguenti passi:

1. accedere in ssh alla Raspberry Pi;
2. decidere una locazione dove installare il progetto software. Non ci sono restrizioni; nel mio caso ho preferito usare la home del mio account;
3. eseguire il clone del repository del progetto;
4. eseguire l'installazione delle dipendenze Python.

Per ambienti di sviluppo o test è possibile pensare di fare ricorso alla creazione di quella che viene definita nel mondo Python, [Virtual Environments](#).

Console 2 - Installazione del progetto software

```
# Accesso al Raspberry Pi via SSH
ssh amusarra@192.168.238.169

# Clone del repository GitHub del progetto
git clone https://github.com/amusarra/smartcard-contactless-raspberry-pi.git

# Installazione delle dipendenze Python
cd smartcard-contactless-raspberry-pi
make
```

Il comando `make` non fa altro che procedere con l'installazione delle dipendenze Python specificate sul file `requirements.txt` utilizzando `pip`. La figura 23 mostra il processo d'installazione delle dipendenze Python sul Raspberry Pi. Ultimata l'installazione, è possibile eseguire il test vero e proprio del software. Prima di eseguire il test occorre accertarsi che dal punto di vista hardware sia tutto regolare controllando tutti i collegamenti (vedi schema elettrico), compreso il collegamento del lettore di Smart Card via USB.

```
amusarra@raspberrypi-amusarra: ~/smartcard-contactless-raspberry-pi
~ (-zsh)          %1          amusarra@raspberrypi-amusarra: ~/smartcard-contactless-raspberry-pi (ssh)  %2 +
amusarra@raspberrypi-amusarra:~/smartcard-contactless-raspberry-pi $ make
pip install -r requirements.txt
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting pycard==2.0.2
  Using cached https://www.piwheels.org/simple/pycard/pycard-2.0.2-cp39-cp39-linux_aarch64.whl
Collecting art==5.4
  Using cached https://www.piwheels.org/simple/art/art-5.4-py2.py3-none-any.whl (561 kB)
Collecting pidfile==0.1.1
  Using cached https://www.piwheels.org/simple/pidfile/pidfile-0.1.1-py3-none-any.whl (2.8 kB)
Collecting pid==3.0.4
  Using cached https://www.piwheels.org/simple/pid/pid-3.0.4-py2.py3-none-any.whl (10 kB)
Collecting pymongo==4.0.1
  Using cached https://www.piwheels.org/simple/pymongo/pymongo-4.0.1-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (460 kB)
Collecting colorama==0.4.4
  Using cached https://www.piwheels.org/simple/colorama/colorama-0.4.4-py2.py3-none-any.whl (16 kB)
Collecting RPi.GPIO==0.7.1
  Using cached https://www.piwheels.org/simple/RPi.GPIO/RPi.GPIO-0.7.1-cp39-cp39-linux_aarch64.whl
Collecting fake-rpi==0.7.1
  Using cached https://www.piwheels.org/simple/fake-rpi/fake_rpi-0.7.1-py3-none-any.whl (9.5 kB)
Requirement already satisfied: numpy in /home/amusarra/.local/lib/python3.9/site-packages (from fake-rpi==0.7.1->--r requirements.txt (line 8)) (1.22.2)
Installing collected packages: RPi.GPIO, pycard, pymongo, pidfile, pid, fake-rpi, colorama, art
Successfully installed RPi.GPIO-0.7.1 art-5.4 colorama-0.4.4 fake-rpi-0.7.1 pid-3.0.4 pidfile-0.1.1 pymongo-4.0.1 pycard-2.0.2
amusarra@raspberrypi-amusarra:~/smartcard-contactless-raspberry-pi $
```

Figura 23 - Installazione delle dipendenze Python tramite il comando make

Ormai dovremmo sapere quali sono gli entry point da utilizzare, sia quello per il setup della Smart Card, sia quello che avvia il controllo degli accessi. Entrambi gli entry point, quindi gli script Python, devono essere avviati specificando una serie di parametri. Le due tabelle a seguire mostrano i parametri d'input dei due script: `setup_smart_card.py` e `access_via_smart_card.py`.

Script Setup Smart Card (parametri)	
-h --help	show this help message and exit
-a AUTHENTICATION_KEY --authentication-key AUTHENTICATION_KEY	Authentication Key of Mifare Classic 1K. The authentication keys are used to authenticate the particular sector of the MIFARE Classic 1K memory card. Volatile authentication key location is provided.
-i DOCUMENT_ID --document-id DOCUMENT_ID	Identity document number (example: identity card, driving license, social security number, passport number)
-s --store-on-database	In this way will associate the Smart Card with the specific user.
-f FIRSTNAME --firstname FIRSTNAME	Firstname of the person to assign the Smart Card
-l LASTNAME --lastname LASTNAME	Lastname of the person to assign the Smart Card
-r {1,2,3,4} --room-number {1,2,3,4}	The room number

Figura 24 - Tabella dei parametri d'input per lo script Python setup_smart_card.py

Script Access via Smart Card (parametri)	
-h --help	show this help message and exit
-a AUTHENTICATION_KEY --authentication-key AUTHENTICATION_KEY	Authentication Key of Mifare Classic 1K. The authentication keys are used to authenticate the particular sector of the MIFARE Classic 1K memory card. Volatile authentication key location is provided.

Figura 25 - Tabella dei parametri d'input per lo script Python access_via_smart_card.py

La figura 26 mostra un esempio di come si presenta l'help in linea dello script `setup_smart_card.py` attivato utilizzando l'opzione `--help` (o `-h`).

```

amusarra@raspberrypi-amusarra:~/smartcard-contactless-raspberry-pi
amusarra@raspberrypi-amusarra:~/smartcard-contactless-raspberry-pi$ ./setup_smart_card.py --help
usage: setup_smart_card.py [-h] -a AUTHENTICATION_KEY -i DOCUMENT_ID [-s] [-f FIRSTNAME] [-l LASTNAME] [-r {1,2,3,4}]

Smart Card initialization tool. This tool is valid for MIFARE Classic 1K cards. It must be used to initialize the card with the identification number of the identification document (example: identity card, driving license, social security number, passport number) of the person to whom the card is delivered.

optional arguments:
-h, --help            show this help message and exit
-a AUTHENTICATION_KEY, --authentication-key AUTHENTICATION_KEY
                      Authentication Key of Mifare Classic 1K. The authentication keys are used to authenticate the particular sector of the MIFARE Classic 1K memory card. Volatile authentication key location is provided.
-i DOCUMENT_ID, --document-id DOCUMENT_ID
                      Identity document number (example: identity card, driving license, social security number, passport number)
-s, --store-on-database
                      In this way will associate the Smart Card with the specific user.
-f FIRSTNAME, --firstname FIRSTNAME
                      Firsname of the person to assign the Smart Card
-l LASTNAME, --lastname LASTNAME
                      Lastname of the person to assign the Smart Card
-r {1,2,3,4}, --room-number {1,2,3,4}
                      The room numer
amusarra@raspberrypi-amusarra:~/smartcard-contactless-raspberry-pi$ 

```

Figura 26 - Come si presenta l'help in line dello script setup_smart_card.py

A questo punto siamo davvero pronti. Il primo step è la registrazione della Smart Card per il nuovo ospite Mario Rossi il cui documento d'identità ha il numero MU589876XD e al quale assegneremo la stanza numero due.

Prima di avviare la registrazione, prendiamo la Smart Card poggiandola sul lettore. Il comando da avviare per la registrazione è: `./setup_smart_card.py -a FFFFFFFFFF -i MU589876XD -s --firstname Mario --lastname Rossi -r 2`

Se tutto va per il verso giusto, l'output ottenuto in console dovrebbe essere quello mostrato dalla figura a seguire.



The ideal solution for your problem

```

amusarra@raspberrypi-amusarra:~/smartcard-contactless-raspberry-pi $ ./setup_smart_card.py -a FFFFFFFFFFFF -i MU589876XD -s --firstname Mario --lastname Rossi -r 2
Version: 1.0.0.dev0
Copyright (c) 2022 Antonio Musarra <antonio.musarra@gmail.com>
Antonio Musarra's Blog - https://www.dontesta.it
GitHub Project https://github.com/amusarra/smartcard-contactless-raspberry-pi
---
Waiting for the Mifare Classic 1K...
Smart Card Reader: BIT4ID miniLector AIR NFC v3 [miniLector AIR DI v3 CLESS] 00 00
Card UID: B4 90 90 1E

Authentication successful
Get previous Identification Number MSRNHTG56X.....
Try to store the new Identification Number MU589876XD into Smart Card...
Try to store the new Identification Number MU589876XD into Smart Card...[Stored]

Starting store data into MongoDB (Mario, Rossi, Room Number: 2)...
Searching entry on the MongoDB with filter {'smartCardId': 'B4 90 90 1E', 'documentId': 'MU589876XD'}...
Searching entry on the MongoDB with filter {'smartCardId': 'B4 90 90 1E', 'documentId': 'MU589876XD'}...[Not Found]
Starting store data into MongoDB (Mario, Rossi, Room Number: 2)...[Inserted with id: 622a736e47f0229a31e4e376]

-- Card initialized successfully and ready to use --
amusarra@raspberrypi-amusarra:~/smartcard-contactless-raspberry-pi $ 
```

Figura 27 - Registrazione Smart Card MIFARE Classic 1K con i dati dell'ospite

Dopo la registrazione della Smart Card e la consegna all'ospite, quest'ultimo può usare la Smart Card per accedere alla propria stanza assegnata in fase di registrazione, che ricordo essere la numero due.

```
> use SmartCardAccessCrudDB
switched to db SmartCardAccessCrudDB
> db.SmartCardAccess.find().pretty()
{
    "_id" : ObjectId("622a7120d360f5fa671d1c54"),
    "createDate" : "2022-03-10T21:44:00.051286Z",
    "firstname" : "Antonio",
    "lastname" : "Musarra",
    "documentId" : "MSRNHTG56X",
    "smartCardEnabled" : "true",
    "smartCardId" : "B4 90 90 1E",
    "smartCardInitDate" : "2022-03-10T21:44:00.051312Z",
    "roomNumber" : 1,
    "countAccess" : 4,
    "firstAccess" : "2022-03-10T21:46:25.812943Z",
    "lastAccess" : "2022-03-10T21:48:34.323634Z",
    "modifiedDate" : "2022-03-10T21:48:34.323675Z"
}
{
    "_id" : ObjectId("622a736e47f0229a31e4e376"),
    "createDate" : "2022-03-10T21:53:50.268461Z",
    "firstname" : "Mario",
    "lastname" : "Rossi",
    "documentId" : "MU589876XD",
    "smartCardEnabled" : "true",
    "smartCardId" : "B4 90 90 1E",
    "smartCardInitDate" : "2022-03-10T21:53:50.268487Z",
    "roomNumber" : 2,
    "countAccess" : 0
}
> □
```

Figura 28 - Documento registrato su MongoDB a fronte del processo di registrazione Smart Card

Avviamo adesso il programma del controllo degli accessi utilizzando il comando: `./access_via_smart_card.py -a FFFFFFFFFF`. Avviato il programma, questo resta in attesa di leggere la Smart Card. Poggiando la Smart Card registrata poc' anzi, l'ospite dovrebbe riuscire ad accedere alla sua stanza, così come mostra la figura a seguire.



amusarra@raspberrypi-amusarra:~/smartcard-contactless-raspberry-pi \$./access_via_smart_card.py -a FFFFFFFFFFFF

```

Version: 1.0.0.dev0
Copyright (c) 2022 Antonio Musarra <antonio.musarra@gmail.com>
Antonio Musarra's Blog - https://www.dontesta.it
GitHub Project https://github.com/amusarra/smartcard-contactless-raspberry-pi
---

Insert or remove a Smart Card in the reader.
Press ctrl+c to exit.

+Insert Card with UID B4 90 90 1E
Get Identification Number MU589876XD
Check if authorized to access...

Searching entry on the MongoDB with filter {'smartCardId': 'B4 90 90 1E', 'documentId': 'MU589876XD', 'smartCardEnabled': 'true'}...
Check if authorized to access... [Access Granted]

Activate the relay for the room number 2...
Activate the relay for the room number 2... [Activated]

-Removed Card ATR 3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6A
+Insert Card with UID 13 9E 55 73
Get Identification Number MU589876XD
Check if authorized to access...

Searching entry on the MongoDB with filter {'smartCardId': '13 9E 55 73', 'documentId': 'MU589876XD', 'smartCardEnabled': 'true'}...
Check if authorized to access... [Access Denied]

```

Figura 29 - Richiesta di accesso via Smart Card MIFARE Classic 1K

L'output mostrato dalla figura 29 evidenzia anche un accesso non riuscito perché in questo caso la Smart Card presentata non è registrata sul sistema. Gli screencast a seguire mostrano i due entry point in azione: setup_smart_card.py e access_via_smart_card.py.



```

In this way will associate the Smart Card with the specific user.
-f FIRSTNAME, --firstname FIRSTNAME
Firstname of the person to assign the Smart Card
-l LASTNAME, --lastname LASTNAME
Lastname of the person to assign the Smart Card
-r {1,2,3,4}, --room-number {1,2,3,4}
The room numer

amusarra@raspberrypi-amusarra:~/smartcard-contactless-raspberry-pi $ ./setup_smart_card.py -a FFFFFFFFFFFF -i MSRNHTG56X -s -f Antonio -l Musar
ra -r 1

Version: 1.0.0.dev0
Copyright (c) 2022 Antonio Musarra <antonio.musarra@gmail.com>
Antonio Musarra's Blog - https://www.dontesta.it
GitHub Project https://github.com/amusarra/smartcard-contactless-rasp
---

Waiting for the Mifare Classic 1K...
Smart Card Reader: BIT4ID miniLector AIR NFC v3 [miniLector AIR DI v3 CLESS] 00 00
Card UID: B4 90 90 1E

Authentication successful

Get previous Identification Number MSRN769BDC.....
Try to store the new Identification Number MSRNHTG56X into Smart Card...
Try to store the new Identification Number MSRNHTG56X into Smart Card... [Stored]

Starting store data into MongoDB (Antonio, Musarra, Room Number: 1)...
Searching entry on the MongoDB with filter {'smartCardId': 'B4 90 90 1E', 'documentId': 'MSRNHTG56X'}...
Searching entry on the MongoDB with filter {'smartCardId': 'B4 90 90 1E', 'documentId': 'MSRNHTG56X'}... [Not Found]
Starting store data into MongoDB (Antonio, Musarra, Room Number: 1)... [Inserted with id: 622a6f24dfac86ccb94e16bf]

-- Card initialized successfully and ready to use --

```

Screencast 1 - Processo di registrazione Smart Card in azione - <https://asciinema.org/a/475795>

```
Copyright (c) 2022 Antonio Musarra <antonio.musarra@gmail.com>
Antonio Musarra's Blog - https://www.dontesta.it
GitHub Project https://github.com/amusarra/smardcard-contactless-raspberry-pi
---

Insert or remove a Smart Card in the reader.
Press ctrl+c to exit

+Insert Card with UID B4 90 90 1E
Get Identification Number MSRNHTG56X
Check if authorized to access...

Searching entry on the MongoDB with filter {'smartCardId': 'B4 90 90 1E', 'documentId': 'MSRNHTG56X', 'smartCardEnabled': 'true'}...
Check if authorized to access...[Access Granted]

Activate the relay for the room number 1...
Activate the relay for the room number 1...[Activated]

-Removed Card ATR 3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6A
+Insert Card with UID 13 9E 55 73
Get Identification Number MU589876XD
Check if authorized to access...

Searching entry on the MongoDB with filter {'smartCardId': '13 9E 55 73', 'documentId': 'MU589876XD', 'smartCardEnabled': 'true'}...
Check if authorized to access...[Access Denied]

-Removed Card ATR 3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6A
+Insert Card with UID B4 90 90 1E
Get Identification Number MSRNHTG56X
Check if authorized to access...

Searching entry on the MongoDB with filter {'smartCardId': 'B4 90 90 1E', 'documentId': 'MSRNHTG56X', 'smartCardEnabled': 'true'}...
Check if authorized to access...[Access Granted]

Activate the relay for the room number 1...
Activate the relay for the room number 1...[Activated]

^CGoodbye
amusarra@raspberrypi-amusarra:~/smardcard-contactless-raspberry-pi $ # Che
```

Screencast 2 - Processo di accesso alla stanza via Smart Card in azione - <https://asciinema.org/a/475797>

Dopo i due screencast che mostrano il sistema di accesso in azione, possiamo affermare che il nostro lavoro di analisi, progettazione e implementazione sia arrivato al termine, raggiungendo anche l'obiettivo prefissato.

12. Conclusioni

Ringrazio tutti voi per essere arrivati “incolumi” alla fine di questo lungo percorso sperando di non essere stato noioso e di essere riuscito nell'intento di rendere interessanti gli argomenti trattati oltre che a spingere la vostra curiosità in avanti.

Potrei lasciarvi un compito per casa: **come aprire la porta di casa utilizzando la propria CIE o TS-CNS.**

Mi sarebbe piaciuto approfondire maggiormente alcuni degli argomenti trattati, come per esempio il framework Pyscard e alcune sezioni del codice Python sviluppato. Nell'attesa di pubblicare altri articoli di approfondimento, vi chiedo di scrivere le vostre impressioni, esperienze o altro di cui vorreste qualche approfondimento utilizzando i commenti all'articolo oppure condividendo attraverso i classici canali social.

Sempre nell'ottica Open Source, il progetto [Smart Card Contactless Raspberry Pi](#) pubblicato sul mio repository GitHub, contiene all'interno della directory [docs](#) i seguenti elementi:

- il file `diagrams.drawio` contenente diagrammi e tavole. File che potete aprire con [Draw.io](#)
- il file `SCH_Smart_Card_Contactless_Raspberry_Pi_2022-03-11.json` dello schema elettrico che potete aprire con [EasyEDA](#)
- i file UML `*.puml` di [PlantUML](#) che contengono i sequence diagram

13. Riferimenti

Come di consueto lascio una serie di risorse che ritengo utili ai fini dell'approfondimento degli argomenti trattati nel corso di questo articolo.

- Attuatori per maker – <https://amzn.to/3slHmE9>
- Raspberry Pi. La guida completa – <https://amzn.to/2RpYZWh>
- Docker: Sviluppare e rilasciare software tramite container – <https://amzn.to/3tiyO1W> di [Serena Sensini](#)
- Valutiamo se continuare a usare Docker o passare a Podman – <https://www.theredcode.it/podman/what-is-podman>
- Pillole di Docker – <https://www.youtube.com/watch?v=wAyUdtQF05w> di [Mauro Cicolella](#)
- Raspberry PI GPIO – Tutti i segreti del pinout – <https://www.moreware.org/wp/blog/2021/04/09/raspberry-pi-gpio-tutti-i-segreti-del-pinout/>
- Smart cards – A short illustrated guide (Feb. 2022) - <https://www.thalesgroup.com/en/markets/digital-identity-and-security/technology/smart-cards-basics>
- Hacking Mifare Classic Cards - <https://www.blackhat.com/docs/sp-14/materials/arsenal/sp-14-Almeida-Hacking-MIFARE-Classic-Cards-Slides.pdf>
- How to Crack Mifare Classic Cards - <https://firefart.at/post/how-to-crack-mifare-classic-cards/>
- A Practical Attack on the MIFARE Classic - <https://arxiv.org/pdf/0803.2285.pdf>
- ACR122U Application Programming Interface - <https://www.acs.com.hk/download-manual/419/API-ACR122U-2.04.pdf>

Raspberry Pi e Smart Card Mifare Classic 1K: Realizzare un Sistema di Accesso

La rivoluzione digitale ha aperto le porte a infinite possibilità per automatizzare e semplificare le nostre vite quotidiane. Questo eBook vi guiderà passo dopo passo nella creazione di un sistema di accesso un Raspberry Pi e le smart card Mifare Classic 1K.

Cosa troverai in questo eBook

- Introduzione alla tecnologia Mifare Classic 1K: una panoramica sulle smart card Mifare, la loro struttura e i principi di funzionamento.
- Setup del Raspberry Pi: istruzioni dettagliate per configurare il tuo Raspberry Pi per la lettura delle smart card.
- Sviluppo del Software: guida completa per scrivere e implementare il codice necessario per interfacciarsi con le smart card.
- Applicazioni Pratiche: suggerimenti su come integrare il sistema in diversi scenari
- Troubleshooting: Soluzioni ai problemi comuni che potresti incontrare durante l'implementazione del sistema.

Questo eBook è perfetto per hobbisti, maker e professionisti dell'IT che desiderano esplorare le potenzialità dei sistemi di accesso basati su Raspberry Pi e smart card.

Ho iniziato il mio viaggio nel mondo dell'informatica da un Olivetti M24 dotato di un processore Intel 8086 acquistato da mio padre esclusivamente per il suo lavoro. Non ho mai posseduto console di nessun genere (Commodore, Amiga, etc...) e inizialmente quell'enorme scatola mi terrorizzava, terrore durato poco; giorno dopo giorno prendevo rapidamente il controllo fino a quando....



Ho sempre creduto che la condivisione della conoscenza sia un ottimo mezzo per la crescita personale e questo è stato uno dei principali motivi che mi ha spinto a creare il mio blog personale www.dontesta.it.

Dicono di me che sono bravo nell'analizzare e risolvere rapidamente i problemi complessi. La mia attività odierna è quella di consulente in progetti enterprise che utilizzano tecnologie Web Oriented come J2EE, Web Services, ESB, TIBCO, Microservices, Cloud Native Application, Kubernetes.



www.linkedin.com/in/amusarra



github.com/amusarra