Projet PUUCE : Cartes magnétiques

Introduction

Le projet PUUCE a été initié en 2017 par pnahoum et a pour objectif d'utiliser le tag Mifare (tag magnétique) des cartes étudiantes à différentes fins au service des associations et de l'école. En effet, chaque carte étudiante dispose d'un tag qui contient un identifiant unique, on peut utiliser cet identifiant pour proposer différents contrôles, qu'ils soient d'accès ou d'usage.

L'objectif du projet PUUCE est de proposer un design ouvert de lecteur de carte qui puisse facilement être utilisé pour divers projets basés sur Arduino/RasPI. Cette page présente la documentation du système de contrôle d'accès mise en place sur la porte du GINFO telle qu'elle a été conçue pour le projet S7 PUUCE. Le projet est cependant assez “ouvert” pour être adapté à de nombreuses autres applications, sous réserve de la compréhension de la dite documentation ci dessous.

Le système : vue d'ensemble

Le premier objectif est de créer un lecteur de carte qui puisse lire les cartes de type **MiFare Ultralight C** (type des cartes étudiants), et qui puisse ensuite envoyer les informations de la carte (à savoir son UID, identifiant unique de chaque carte) à un Raspberry PI qui pourra les traiter et y donner différentes applications (ouverture de porte par exemple).

Le système est composé de plusieurs élements interconnectés :

→ Le lecteur de carte : positionné à l'extérieur du ginfo. Il permet de lire les cartes de type Mifare Ultralight C (étudiant) et il envoie via un cable DB-9 les informations au boitier de traitement. Il est composé d'un boitier plastique dans lequel sont disposés un lecteur PN532 et un buzzer.

→ Le boitier de traitement : composé d'une entrée d'alimentation, d'une entrée/sortie pour le contrôle d'un appareil par relais, une entrée DB-9 pour interagir avec le lecteur, ainsi que nappe IDE pour échanger les informations avec le Raspberry PI

→ La gâche électrique : connectée au boitier de traitement, celle ci s'ouvre et laisse la porte s'ouvrir uniquement si la Raspberry Pi le lui ordonne. (i.e : quand une carte étudiant est autorisée à pénétrer dans le local du GInfo)

→ Le raspberry PI : il est connecté au système via la nappe IDE et connecté en Ethernet au réseau du GINFO.

→ Un élement “virtuel” : l'application PUUCE (connectée à l'API PUUCE) accessible via l'URL http://www.puuce.ginfo qui permet de dire à la raspberry PI si oui ou non une personne est autorisée à activer le module. Cette application permet aussi de rajouter des personnes autorisées ou non.

Chacune des parties de la suivante documentation détaille le fonctionnement de chaque unité de manière séparée. Une note finale explique comment interconnecter les différents éléments pour avoir un système fonctionnel.

Le lecteur de carte

Le lecteur de carte est composé de plusieurs éléments :

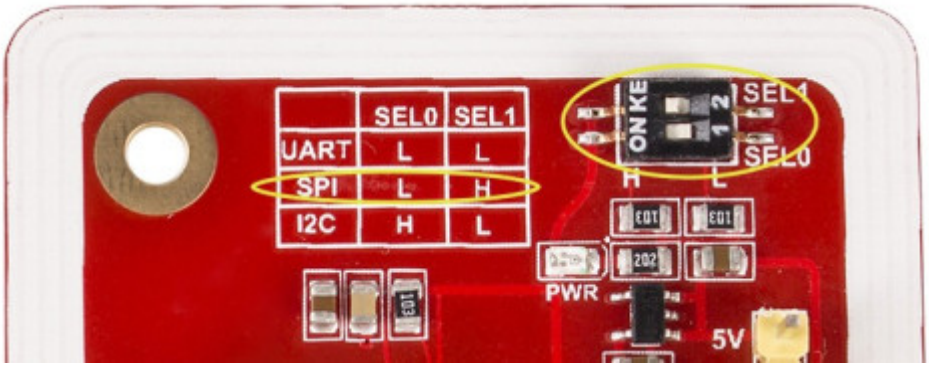
- Un boitier interrupteur plastique vidé de son électronique (pour y placer la notre) (ref : MAIFO TOUCH SWITCH)
- Un module de lecteur MIFARE PN532 miniaturisé (cf : image ci dessous ou lien ICI). Attention, certains modules PN532 de ce genre ne lisent pas les Mifare Ultralight C car ils sont de mauvaise qualité.
- Un buzzer piezzo
- Un connecteur de type DB-9 femelle (qui fera le lien avec note boitier de traitement)

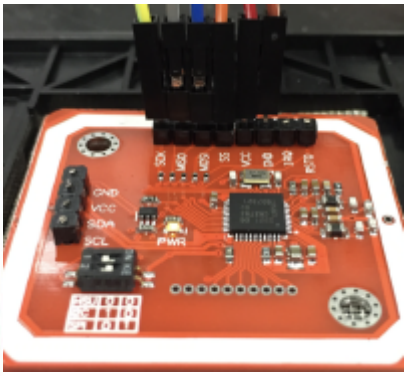
Ci dessous vous trouverez les images de ces 4 éléments nécessaires à la réalisation du boitier



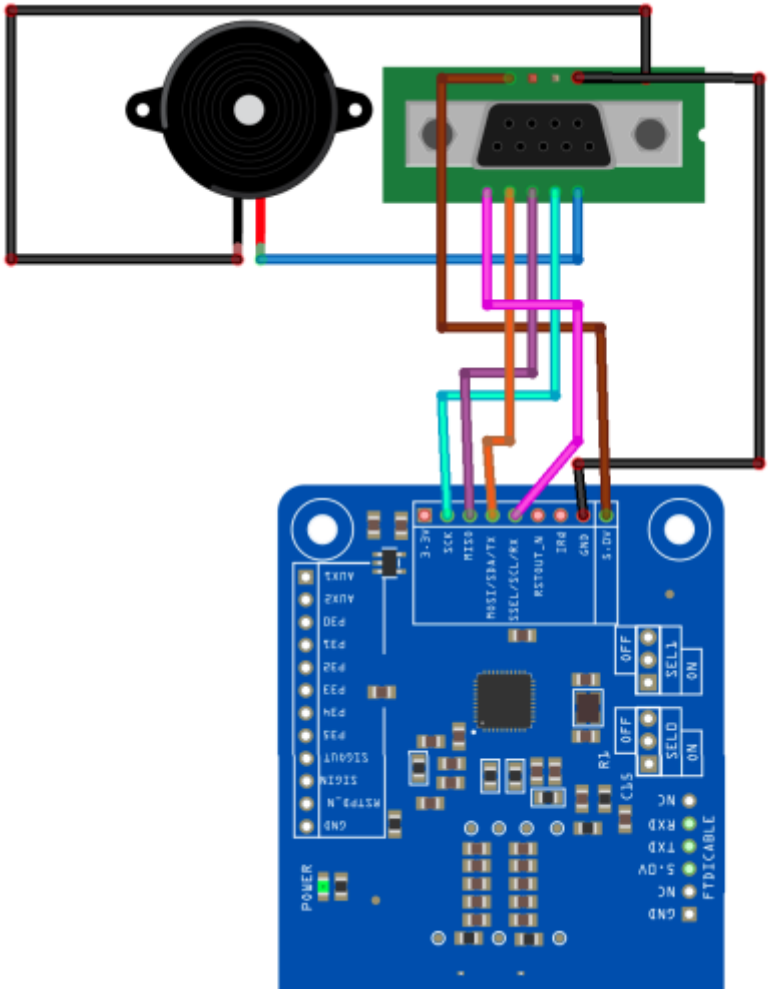

Le principe du lecteur est assez simple. Il doit envoyer au boitier de traitement (via le connecteur DB9) l'UID de la carte lue. Si la carte est lue, le boitier de traitement confirmera la lecture et l'accès (ou non) via le buzzer intégré.

Le transfert de données est effectué via le Protocole SPI vers le boitier de traitement. Pour cela, il faut brancher correctement le module et activer le module SPI en tournant le pin de réglage au bon endroit sur le module. Ci dessous vous trouverez les images des branchements sur le module lors de la phase de prototypage.





Enfin, il faut tout faire rentrer dans l'arrière du boitier et souder les sorties SPI du lecteur, l'alimentation du lecteur et le buzzer sur le connecteur DB9. Vous trouverez ci dessous un schéma détaillé. (notez qu'ici le lecteur est bleu, mais les connexions sont bonnes et identiques à celui du PN532 rouge)



Le tout est collé à la colle chaude sur l'arrière du boitier, fermé par le module PN532 de sorte à ce qu'on puisse retirer facilement le panneau avant en cas de problème.

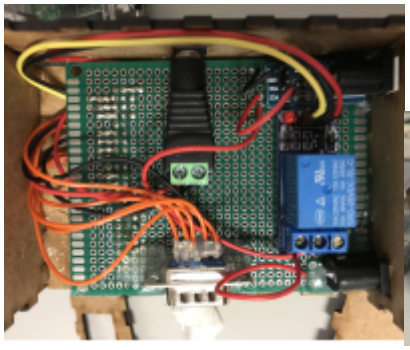
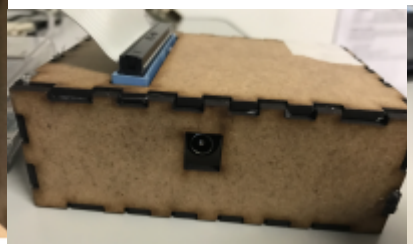
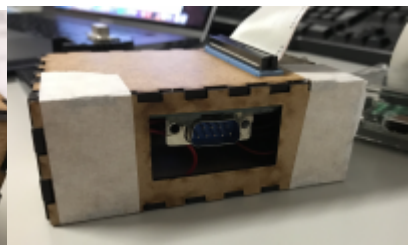
A noter qu'une amélioration future possible serait l'ajout d'une LED tricolore, la vitre du lecteur étant transparente.

Le boitier de traitement

Le boitier de traitement est le coeur du système. Il permet d'alimenter le système (Gâche électrique, Raspberry PI...), et il contient le relais qui va contrôler la gâche (ou l'appareil que l'on veut controller).

Il dispose de plusieurs entrées :

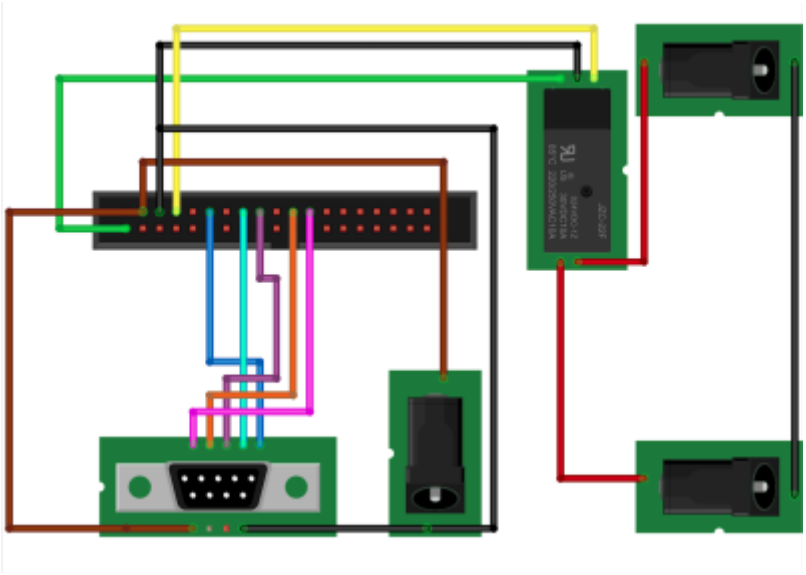
- Une entrée d'alimentation 5V (prise circulaire 2,1mm) pour alimenter le relais et le raspberry PI
- Une entrée d'alimentation (prise circulaire 2,1mm) pour l'appareil dont on veut contrôler l'usage (ici la gâche)
- Une sortie (prise circulaire 2,1mm) pour connecter l'appareil que l'on veut contrôler (ici la gâche)
- Un port DB-9 pour connecter le lecteur de carte placé à l'extérieur
- Un port IDE male pour transférer les données du lecteurs au Raspberry PI, et alimenter le Raspberry PI

Il faut savoir que tout ce qui arrive sur le port DB-9 est directement redirigé vers le port IDE male, le boitier de traitement n'effectuant pas de traitement en somme (il n'y a pas de microcontrolleur dans ce boitier), il est juste une sorte de boitier “agrégateur” sur lequel on branche tous les éléments du système pour qu'ils puissent communiquer entre eux.

Le tout est placé dans un boitier en bois dont le design n'est pas définitif.

Le schéma est donné ci dessous :

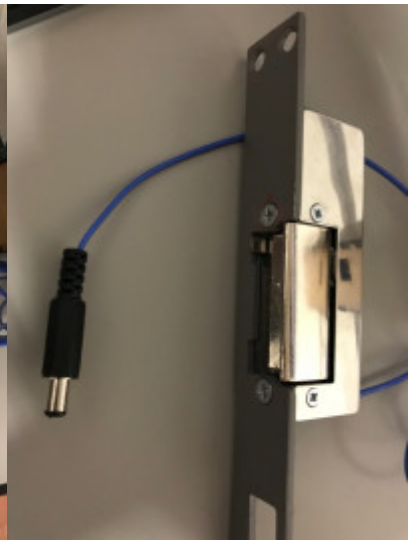


La Gâche Electrique

La gâche électrique est très simple. Elle s'active quand un courant de 12V / 500mA lui est appliqué. Elle est donc directement connectée au boitier de traitement, et le relais active le passage du courant quand le raspberry PI lui en donne la consigne. (HIGH → LOW)

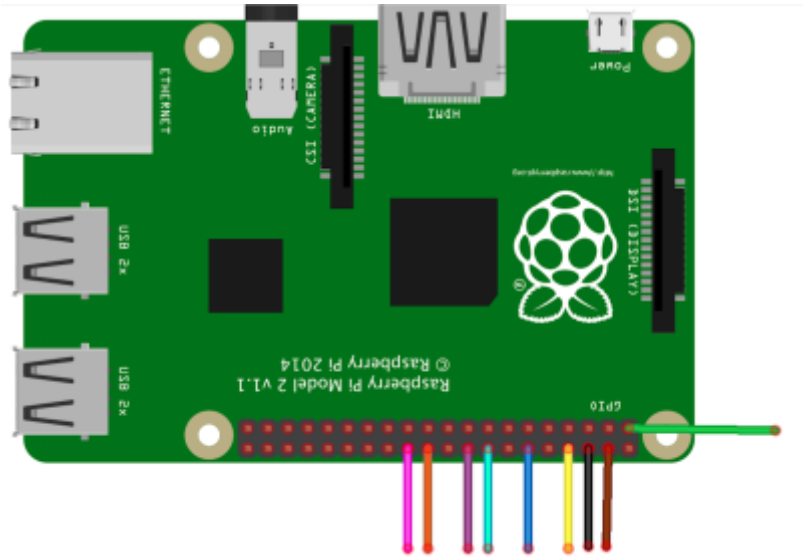
Ce gâche est encastrée dans le dormant de la porte et est connectée à un cable dont la sorte est circulaire 2,1mm.






La Raspberry PI

Nous supposons pour commencer que vous avez une Raspberry PI avec Raspbian d'installé et connectée au réseau du GInfo via Ethernet, avec une IP FIXE Dans le cas de la porte du GINFO, **l'IP du Raspberry PI est 10.61.16.70**



La raspberry pi est alimentée et interagit avec le système via son port de 2×20 pins, il faut un cable IDE (Parallel ATA) pour le relier avec le reste du système.

Installation de python et des dépendances

On se place ou l'on veut et on installe

On commence par installer la bibliothèque PN532

```
sudo apt-get update
sudo apt-get install build-essential python-dev git
git clone https://github.com/adafruit/Adafruit_Python_PN532.git
cd Adafruit_Python_PN532
sudo python setup.py install
cd ..
rm -R Adafruit_Python_PN532
```

On installe ensuite d'autres dépendances Python

```
sudo apt-get install python-pip
pip install pipenv
pipenv install requests
```

Enfin, on charge le code du gestionnaire de fichier dans /home/puuce.py

Ce code est temporaire, il ne prend pas en compte certaines exceptions qui peuvent mener le module à être redémarré pour re-fonctionner en cas de problème. (ex : déconnexion subite du lecteur)

/home/puuce.py

```
#Importation des dependances
```

```
import binascii
import sys
import Adafruit_PN532 as PN532
import time
import requests
import hashlib
import RPi.GPIO as GPIO
import time

# Configuration des pins de sorties pour la  Raspberry Pi (numero
GPIO):
SS   = 8
MOSI = 25
MISO = 24
SCK = 23
channel_buzzer = 18
channel_relais = 14

#Parametres pour accéder à l'API PUUCE (donnant la correspondance
pseudo Centrale/UID)
apikey="XXXXXXXXXXX" #APIKEY DE LECTURE
url = "http://puuce.ginfo/access" #ADRESSE DES REQUETES
uid_porte="uid1porteginfo"; #ID DE LA PORTE


#Initialisation des sorties RSPBRYPI
GPIO.cleanup()
GPIO.setmode(GPIO.BCM)
GPIO.setup(channel_relais,GPIO.OUT)
GPIO.output(channel_relais,GPIO.HIGH)

#Définition de la fonctionpour le buzz du lecteur
def buzz(frequency, duration, channel):
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(channel, GPIO.OUT)
    p = GPIO.PWM(channel, frequency)
    p.start(5)
    #ici, rapport_cyclique vaut entre 0.0 et 100.0
    time.sleep(duration)
    p.stop()

#Définition de la fonction pour ouvrir le relais
def openrelais(channel):
    GPIO.output(channel,GPIO.LOW)
        time.sleep(5)
    GPIO.output(channel,GPIO.HIGH)

# Initialisation du lecteur
pn532 = PN532.PN532(cs=SS, sclk=SCK, mosi=MOSI, miso=MISO)
pn532.begin()
```

```
# Récupération du firmware PN532 et affichage
ic, ver, rev, support = pn532.get_firmware_version()
print('Found PN532 with firmware version: {0}.{1}'.format(ver, rev))

#Configuration du lecteur
pn532.SAM_configuration()


#Boucle principale de lecture
print('En attente d\'une carte...')
while True:
    # Vérification si un carte est là
    uid = pn532.read_passive_target()
    # On recommence la boucle s'il n'y a pas de carte
    if uid is None:
        continue
    #Sinon, on a une carte et donc UID et on continue
    #On buzz donc la lecture
    buzz(1000,0.1, channel_buzzer);
    time.sleep(0.1);
    buzz(1000,0.1, channel_buzzer);
    time.sleep(0.1);
    buzz(1000,0.1, channel_buzzer);
    #On récupère l'UID sous forme hexa
    uid_encrypted = format(binascii.hexlify(uid));

    #On prépare la requête POST pour l'API
    payload={'apikey': apikey, 'uid_personne': uid_encrypted,
'uid_porte': uid_porte}
    r = requests.post(url, data=payload)
    #Si la requête POST est un succès
    if(r.status_code == 200):
    reponse = r.text[:300]
    resultat = reponse.split("_")
        #Si la personne est autorisée on ouvre la porte
    if(resultat[1] == "1"):
            print("Acces autorise");
        buzz(1500,1, channel_buzzer);
        openrelais(channel_relais);
        #Sinon on ouvre pas la porte
    else:
            print("Acces refuse");
            buzz(440,1, channel_buzzer);
    print(reponse)
    #Dans le cas ou la réponse n'est pas 200, on n'ouvre pas la porte
    else:
    print("pas de reponse serveur");
        print(r.status_code);
        print("Acces refuse");
        buzz(440,1, channel_buzzer);
```

```
    #A la fin : reinitialsation uid puis pause d lecture
    uid_encrypted = ""
    time.sleep(3) # pause 3 secondes
```

Utilisation de contrôle d'accès

Depuis les lecteurs, on va interroger **puuce.ginfo/access** avec une requête **POST**. Pour pouvoir utiliser les données de l'API il faut fournir une clé d'autorisation (ici nommé APIKEY)

Attribut	Description	Exemple
apikey	La clé de l'api pour lire	-
uid_personne	L'UID encodé en sha1 par la Raspberru-iPI	-
uid_porte	L'UID de la porte concernée	uidporteginfo1

L'application de gestion : PUUCE.GINFO

L'application de gestion des accès est hébergée par le GInfo, elle interagit avec l'API PUUCE et est accessible depuis le réseau interne de l'école via puuce.ginfo

L'application a été développée en PHP à l'aide du Framework Symfony et n'est pas du tout terminée, bien que totalement utilisable (elle fait le minimum). Pour rajouter des portes ou des groupes, il faut passer pas la DB directement.

Elle permet de :

→ Rajouter un utilisateur et l'associer à son UID dans la DB

→ Affecter un utilisateur à une porte pour qu'il y ai accès

→ Voir l'historique des accès autorisés et refusés

L'application est disponible en protégé sur le dépôt GIT de la forge de L'ECM

API

L'API web est documentée sur la page Fonctionnement de l'API PUUCE.