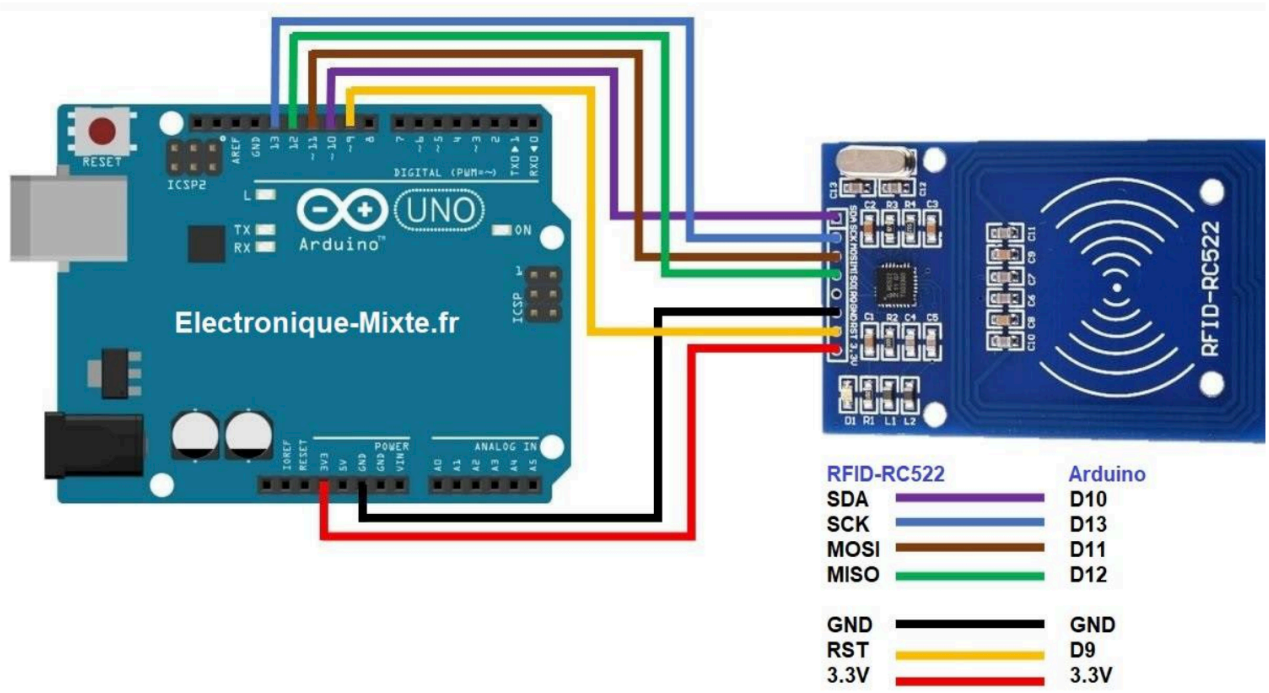
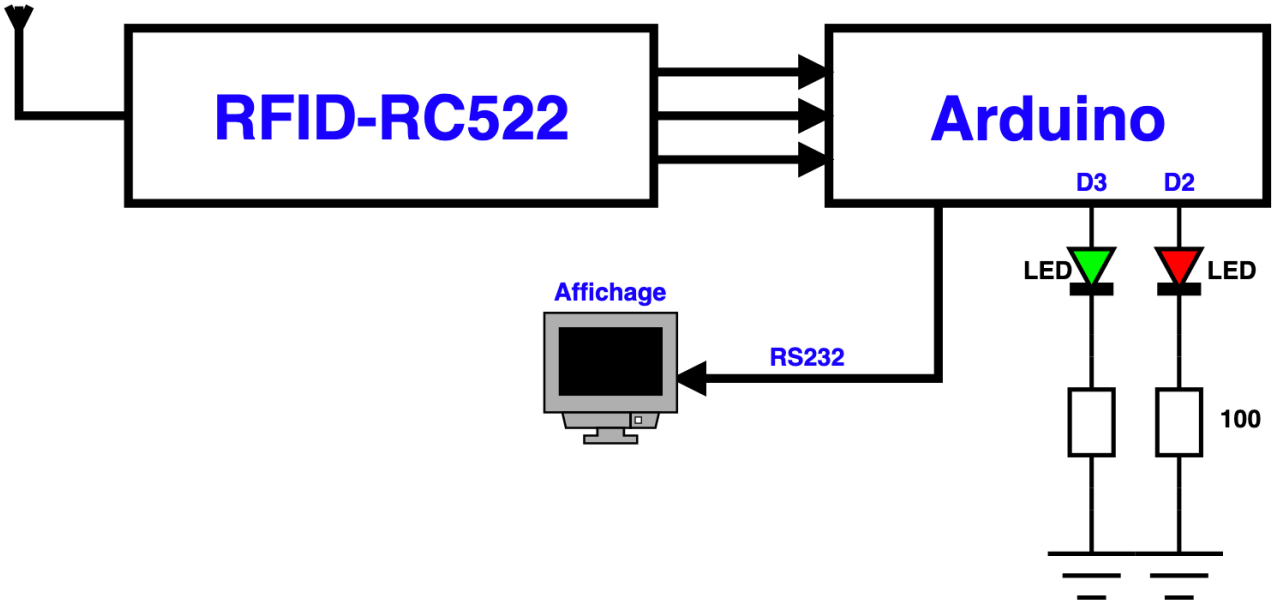
Partie 2 :





L'application consiste en l'ouverture d'une porte en utilisant un badge. Le lecteur RFID couplé à la carte Arduino permet de détecter un badge enregistré ou non. Lorsque l'utilisateur est reconnu, le système déclenche l'ouverture de la porte (ou une alarme). L'utilisateur à droit à trois tentatives. Le nombre de tentatives est ajustable par le programme Arduino. Ci-dessous les éléments constituant le projet ainsi leurs fonctionnements.

LED verte : Voyant indiquant l'ouverture de la porte. La LED s'allume pendant une seconde lorsqu'un badge reconnu est détecté. Elle reste éteinte dans le cas contraire

LED rouge : Voyant indiquant la détection d'un Faux badge (identifiant non reconnu du badge). La LED rouge s'allume pendant une seconde puis s'éteint pour chaque fausse détection. Lorsque le nombre de tentatives est atteint, la LED rouge clignote en boucle infinie en état d'alarme. Aucun moyen n'est possible pour réactiver le système à part la réinitialisation de la carte Arduino.

RS232 : Le système communique l'état de la porte ou la présence de l'alarme par la liaison RS232. On affiche « Ouverture de la porte » lorsque la LED verte est allumée. Et « Alarme » en boucle lorsque le nombre de tentatives est atteint.

RFID-ARDUINO:

Signal	RFID-RC522	ARDUINO
RST/Reset	RST	D9
SPI SS	SDA(SS)	D10
SPI MOSI	MOSI	D11
SPI MISO	MISO	D12
SPI SCK	SCK	D13

Le module RFID est accompagné de deux badges de formes différentes : l'un se forme d'une carte et l'autre d'une clé. Pour l'instant on ne connaît pas les identifiants de chacun d'entre eux. La première étape consiste à reconnaitre les ID pour les opérations à venir. On a besoin de télécharger la librairie RFID (disponible ICI). On considère le même schéma de câblage. Ci-dessous les étapes importantes de déclaration, initialisation, lecture et affichage de l'ID d'un badge.

1- Déclaration

```
#include <SPI.h>      // SPI
#include <MFRC522.h>  // RFID

#define SS_PIN 10
#define RST_PIN 9

// Déclaration
MFRC522 rfid(SS_PIN, RST_PIN);

// Tableau contentent l'ID
byte nuidPICC[4];
```

2- Initialisation

```
void setup() {
  // Init RS232
  Serial.begin(9600);

  // Init SPI bus
  SPI.begin();

  // Init MFRC522
  rfid.PCD_Init();
}
```

3- Attente d'un nouveau badge

```
void loop() {
  // Initialiser la boucle si aucun badge n'est présent
  if (!rfid.PICC_IsNewCardPresent())
    return;

  // Vérifier la présence d'un nouveau badge
  if (!rfid.PICC_ReadCardSerial())
    return;
  …
}
```

4- Enregistrement de l'ID

```
…
// Enregistrer l'ID du badge (4 octets)
for (byte i = 0; i < 4; i++) {
  nuidPICC[i] = rfid.uid.uidByte[i];
}
…
```

5- Affichage de l'ID

```
…
// Affichage de l'ID
Serial.println("Un badge est détecté");
Serial.println(" L'UID du tag est:");
for (byte i = 0; i < 4; i++) {
  Serial.print(nuidPICC[i], HEX);
  Serial.print(" ");
}
Serial.println();
…
```

6- Re-Initialisation RFID

```
...
// Re-Init RFID
rfid.PICC_HaltA();        // Halt PICC
rfid.PCD_StopCrypto1();   // Stop encryption on PCD
...
```

7- Programme complet

```
#include <SPI.h>      // SPI
#include <MFRC522.h>  // RFID

#define SS_PIN 10
#define RST_PIN 9

// Déclaration
MFRC522 rfid(SS_PIN, RST_PIN);

// Tableau contentent l'ID
byte nuidPICC[4];

void setup() {
  // Init RS232
  Serial.begin(9600);

  // Init SPI bus
  SPI.begin();

  // Init MFRC522
  rfid.PCD_Init();
}

void loop() {
  // Initialiser la boucle si aucun badge n'est présent
  if (!rfid.PICC_IsNewCardPresent())
    return;

  // Vérifier la présence d'un nouveau badge
  if (!rfid.PICC_ReadCardSerial())
    return;

  // Enregistrer l'ID du badge (4 octets)
  for (byte i = 0; i < 4; i++) {
    nuidPICC[i] = rfid.uid.uidByte[i];
```

```
  }

  // Affichage de l'ID
  Serial.println("Un badge est détecté");
  Serial.println(" L'UID du tag est :");
  for (byte i = 0; i < 4; i++) {
    Serial.print(nuidPICC[i], HEX);
    Serial.print(" ");
  }
  Serial.println();

  // Re-Init RFID
  rfid.PICC_HaltA();       // Halt PICC
  rfid.PCD_StopCrypto1();  // Stop encryption on PCD
}
```

Programme du contrôle d'accès par un badge

On reprend les mêmes étapes du programme de lecture. En revanche, on garde en mémoire l'ID du badge {0x20, 0x12, 0x23, 0x2B}. Le programme compare en permanent le nouvel ID avec l'ID de base. Si les deux ID sont identiques, on déclenche l'ouverture de la porte, sinon l'alarme après trois tentatives. On fera des tests en utilisant le bon badge ayant le même ID enregistré et un nouveau badge ayant un ID différent.

On utilise une nouvelle fonction GetAccesState() qui prend en argument les deux ID (ID enregistré et le nouvel ID), puis elle retourne « 1 » si les deux ID sont identiques et « 0 » dans le cas contraire. Les variables à l'entrée sont deux tableaux de tailles 4 aux formats byte (la taille du code est égale à 4 octets). Ci-dessous la déclaration de la fonction :

```
byte GetAccesState(byte *CodeAcces, byte *NewCode) {
  byte StateAcces = 0;
  if ((CodeAcces[0] == NewCode[0]) && (CodeAcces[1] == NewCode[1]) &&
    (CodeAcces[2] == NewCode[2]) && (CodeAcces[3] == NewCode[3]))
    return StateAcces = 1;
  else
    return StateAcces = 0;
}
```

Programme principal

```
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 10
#define RST_PIN 9

#define AccesFlag_PIN 2
#define Gate_PIN 3
#define Max_Acces 3

byte Count_acces = 0;
byte CodeVerif = 0;
byte Code_Acces[4] = { 0x20, 0x12, 0x23, 0x2B };

MFRC522 rfid(SS_PIN, RST_PIN);  // Instance de la classe

// Tableau d'initialisation qui stockera le nouveau NUID
byte nuidPICC[4];

void setup() {
  // Init RS232
  Serial.begin(9600);

  // Init SPI bus
  SPI.begin();

  // Init MFRC522
  rfid.PCD_Init();

  // Init LEDs
  pinMode(AccesFlag_PIN, OUTPUT);
  pinMode(Gate_PIN, OUTPUT);

  digitalWrite(AccesFlag_PIN, LOW);
  digitalWrite(Gate_PIN, LOW);
}

void loop() {
  // Initialiser la boucle si aucun badge n'est présent
  if (!rfid.PICC_IsNewCardPresent())
    return;

  // Vérifier la présence d'un nouveau badge
  if (!rfid.PICC_ReadCardSerial())
```

```
  return;

// Afffichage
Serial.println(F("Un badge est détecté"));

// Enregistrer l'ID du badge (4 octets)
for (byte i = 0; i < 4; i++) {
  nuidPICC[i] = rfid.uid.uidByte[i];
}

// Vérification du code
CodeVerif = GetAccesState(Code_Acces, nuidPICC);
if (CodeVerif != 1) {
  Count_acces += 1;
  if (Count_acces == Max_Acces) {
    // Dépassement des tentatives (clignotement infinie)
    while (1) {
      digitalWrite(AccesFlag_PIN, HIGH);
      delay(200);
      digitalWrite(AccesFlag_PIN, LOW);
      delay(200);
      // Affichage
      Serial.println("Alarme!");
    }
  } else {
    // Affichage
    Serial.println("Code érroné");

    // Un seul clignotement: Code erroné
    digitalWrite(AccesFlag_PIN, HIGH);
    delay(1000);
    digitalWrite(AccesFlag_PIN, LOW);
  }
} else {
  // Affichage
  Serial.println("Ouverture de la porte");

  // Ouverture de la porte & Initialisation
  digitalWrite(Gate_PIN, HIGH);
  delay(3000);
  digitalWrite(Gate_PIN, LOW);
  Count_acces = 0;
}

// Affichage de l'identifiant
Serial.println(" L'UID du tag est:");
```

```
  for (byte i = 0; i < 4; i++) {
    Serial.print(nuidPICC[i], HEX);
    Serial.print(" ");
  }
  Serial.println();

  // Re-Init RFID
  rfid.PICC_HaltA();       // Halt PICC
  rfid.PCD_StopCrypto1();  // Stop encryption on PCD
}

byte GetAccesState(byte *CodeAcces, byte *NewCode) {
  byte StateAcces = 0;
  if ((CodeAcces[0] == NewCode[0]) && (CodeAcces[1] == NewCode[1]) &&
    (CodeAcces[2] == NewCode[2]) && (CodeAcces[3] == NewCode[3]))
    return StateAcces = 1;
  else
    return StateAcces = 0;
}
```