Dan NACACHE dan.nacache@edu.esiee.fr
Florian NEUVILLE florian.neuville@edu.esiee.fr
Jean-Laurent ROUVIÈRE jean-laurent.rouviere@edu.esiee.fr

RAPPORT DE PROJET E4

PTF MIFARE ET

DISTRIBUTEUR DE BOISSONS

SOMMAIRE

1. INTRODUCTION

Dans un monde où nous utilisons de plus en plus de produits sans contact et où la sécurité joue un rôle crucial, certains produits continuent d'être commercialisés et utilisés malgré le fait qu'ils comportent des vulnérabilités critiques : c'est le cas des cartes MIFARE Classic 1K et 4K.

Notre projet a pour but de développer un jeu en Python disponible sur Linux et Windows, permettant de montrer et d'utiliser les principales faiblesses de cette gamme de produits.

Celui-ci simulera les situations du quotidien les plus courantes dans lesquelles ces failles sont susceptibles d'apparaître et donnera à l'utilisateur la possibilité de les exploiter. Pour cela, il sera composé de plusieurs scénarios eux-mêmes divisés en plusieurs niveaux, chacun représentant une vulnérabilité différente.

Nous étudierons d'abord le fonctionnement des cartes MIFARE ainsi que leur vulnérabilité, puis nous établirons une maquette du jeu avant de détailler l'implémentation de celui-ci. Nous aborderons enfin les difficultés que nous avons rencontrées au cours de notre travail.

2. ÉTAT DE L'ART

Afin de pouvoir développer un environnement d'apprentissage, nous devions d'abord prendre connaissance de l'architecture des cartes MIFARE ainsi que de leurs usages courants pour pouvoir étudier en détail puis implémenter leurs principales vulnérabilités. Pour cela, nous avons dû réaliser un état de l'art des cartes d'accès MIFARE ainsi qu'une étude de leurs vulnérabilités. Ce document détaille de manière exhaustive le fonctionnement de ces cartes et peut être directement inclus dans ce rapport : nous résumerons néanmoins ici les informations essentielles à la compréhension de nos travaux.

a. Étude des cartes sans contact MIFARE

MIFARE est une marque affiliée à NXP Semiconductors (Philips) dont les produits se décomposent en plusieurs familles de tags sans contact fonctionnant grâce à la technologie RFID :

- MIFARE Ultralight,
- MIFARE DESFire,
- MIFARE Plus,
- MIFARE Classic.

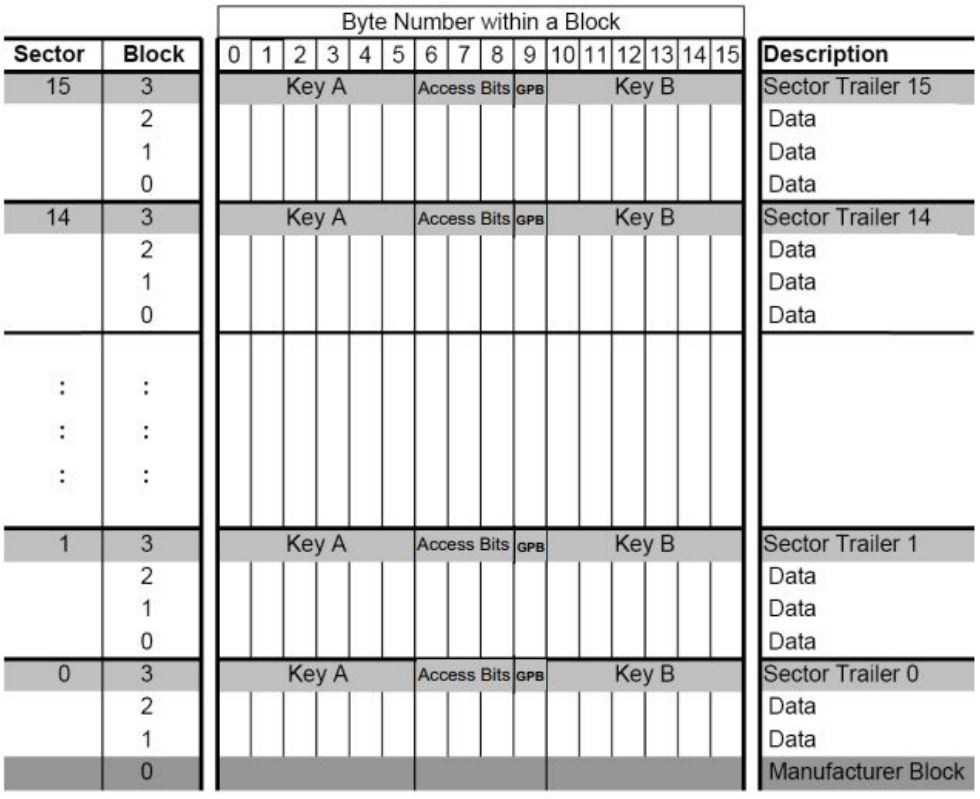
Product features	MIFARE Ultralight®				MIFARE Classic®		MIFARE Plus®							MIFARE® DESFire®						
	Nano	EV1		C	EV1		S		SE	X		EV1		EV1				EV2		
RF Interface	ISO/IEC 14443-2, Type A 13.56 MHz																			
Protocol	ISO/IEC 14443-3						ISO/IEC 14443-3&4							ISO/IEC 14443-4						
UID - unique identifier	7-byte UID				7-byte UID, 4-byte NUID, Random ID									7-byte UID, Random ID						
Communication speed	106 Kbps						106-848 Kbps													
Memory size [Bytes]	40	48	128	144	1K	4K	2K	4K	1K	2K	4K	2K	4K	256	2K	4K	8K	2K	4K	8K
Memory model	Compact, 4-byte pages				Compact, sectors & 16-byte blocks									Flexible file system						
Crypto	-			TDES	Crypto-1		Crypto-1, AES							DES / 2K3DES / 3K3DES / AES						
Key length	-			112-bit	48-bit		48-bit Crypto-1, 128-bit AES							128-bit AES, up to 168-bit DES						
Authentication	-	Password		3-pass mutual																
Communication security	-				Encrypted		Plain, CMACed, encrypted w. CMAC							Plain, CMACed, encrypted w. CMAC						
MIsmartApp	-													-				✓		
Transaction MAC	-						-					✓		-				✓		
Multi key sets	-													-				✓		
Proximity check	-									✓				-				✓		
Virtual card select	-						✓							-				✓		
Originality check features	ECC signature programmable	ECC signature		-	ECC signature	-	AES originality keys					AES originality keys, ECC signature		-				AES originality keys, ECC signature		
CC Certification	-						EAL4+		-	EAL4+		EAL5+		EAL4+				EAL5+		
ISO 7816-4 APDU	-											✓		✓						
NFC compliance	NFC Forum type 2 tag compliant				Not supported by majority of NFC devices		NFC capable in SL3					NFC capabilities in SL1 and SL3		NFC Forum type 4 tag V2.0 compliant						
Target applications	Public transport & event ticketing loyalty programs, limited use tickets				Various applications – recommended to move to higher security ICs		Public transport / campus cards / access management							Smart city platform / advanced mobility multi-applications / micropayment / loyalty programs / access management						
Input capacitance [pF]	17 / 50				17		17		17/70	17		17	70	17 / 70						
Multi applications	-				supported via MAD		supported via MAD							dynamic						

C'est cette dernière catégorie, dont l'algorithme de chiffrement propriétaire (CRYPTO1) a été cassé et dont la gestion des UID dans un cadre de contrôle d'accès peut être facilement contournée, qui est la plus vulnérable.

Selon les modèles, la structure de la mémoire n'est pas partitionnée de la même façon : dans le cas des cartes MIFARE Classic, elle est divisée en blocs de seize octets regroupés en secteurs, avec un schéma différent selon la version.
Les cartes MIFARE Classic 1K possèdent une mémoire de type EEPROM découpée en 16 secteurs de 4 blocs.
La carte MIFARE Classic 4K possède une mémoire de type EEPROM également, ses 32 premiers secteurs étant découpés en 4 blocs et les 8 secteurs suivants étant eux divisés en 16 blocs.
Le dernier bloc de chaque secteur, appelé "sector trailer", possède une structure identique et contient encodé en hexadécimal :

- octets 0 à 5 : la clé A (48 bits)
- octets 6 à 9 : ACs (=access conditions), gère les conditions d'accès au secteur
- octets 10 à 15 : la clé B (optionnelle)

Ces valeurs peuvent être définies pour chaque secteur, et les conditions d'accès au secteur sont définies selon la valeur des différents bits de chaque octet.
La structure mémoire des cartes MIFARE Classic 1K peut être schématisée de la façon suivante :



Les données peuvent être stockées sous la forme de bloc de données ou de bloc de valeurs, selon la valeur des « Access Bits ». Selon le format utilisé l'accès à la mémoire se fera avec des opérations mémoires, et seules seront disponibles :
- pour un bloc de valeurs : lecture, écriture
- pour un bloc de valeurs : lecture, écriture, incrémentation, décrémentation, restauration (prépare la valeur courante à être écrasée) et transfert (écrit le résultat d'une des opérations précédentes dans la mémoire non volatile).

b. Vulnérabilités

i. Liées aux produits MIFARE

Les vulnérabilités des cartes MIFARE Classic peuvent être regroupées en 4 catégories selon leur origine :

- faiblesse du générateur de nombre pseudo-aléatoire
- faiblesse de l'algorithme de chiffrement CRYPTO1
- faiblesse du protocole de communication
- faiblesse de l'implémentation

Elles peuvent être exploitées de façon concrète à travers l'utilisation de clés par défaut d'une grande partie des cartes commercialisées pour les clés A et B, qui peuvent ainsi être retrouvées par une simple attaque par dictionnaire, et grâce à des outils comme MFOC ou MFCUK. Ces programmes implémentent les attaques nested et dark-side afin de casser les clés de chiffrement utilisées dans les différents secteurs des cartes.

Il existe également une vulnérabilité sur les cartes MIFARE Ultralight rendant possible leur réécriture, ou sur la première génération de cartes DESFire, permettant de casser les clés utilisées en approximativement 7h.

ii. Liées à l'implémentation

En plus de ces failles intrinsèques aux produits, certaines peuvent apparaître avec l'utilisation de ceux-ci. Les données circulent par exemple en clair dans certains lecteurs, et peuvent être interceptées. L'UID n'est plus considérable comme « unique » et ne doit ainsi pas être utilisé comme moyen d'identification, et un solde inscrit sur une carte peut être facilement modifié une fois que les clés de chiffrement ont été cassées.
Enfin, il arrive que les opérations d'incrémentation et de décrémentation ne soient pas contrôlées et puissent être contournées, en enlevant par exemple sa carte au moment du paiement.

3. CONCEPTION

a. Document de conception

Après avoir étudié le cadre de notre projet, nous avons effectué un document de conception à l'aide du Statement of Work fourni par notre tuteur. Le jeu prendra ainsi la forme d'un programme Python fonctionnant avec la librairie PyGame, et utilisera un lecteur RFID MFRC522 relié à une carte Arduino Uno pour lire les cartes MIFARE Classic.

On pourra y parcourir différents tableaux dans lesquels un personnage se trouve face à des situations courantes d'usage de cartes MIFARE Classic. L'utilisateur devra lire sa carte sur le MFRC522, et si les conditions de succès sont réunies il pourra accéder au niveau suivant.
Afin de traiter la majorité des vulnérabilités existantes, nous avons décidé de découper le jeu en 3 niveaux, « Porte », « Hôtel » et « Distributeur », comportant chacun des sous-niveaux.

On aura ainsi :

- **Un écran de configuration** :

C'est le premier écran du jeu, il permet de choisir le système d'exploitation de l'utilisateur et le numéro du port utilisé par le lecteur RFID.

- **Un écran d'introduction** :

Il présente brièvement le jeu à l'utilisateur.

- **Un menu principal** :

Il permet de sélectionner un niveau parmi les 3 possibles, accessibles à l'aide de boutons. L'utilisateur pourra également quitter le jeu s'il le souhaite à l'aide d'un bouton « Quitter ».

- **3 menus de sélection de sous-niveaux** :

Il existe un menu de sélection pour chacun des niveaux. Chacun de ces menus affichera les sous-niveaux disponibles relatifs au niveau sélectionné, qui seront accessibles sous la forme de boutons cliquables également. Il y aura également un bouton « Retour », permettant d'accéder au menu principal depuis chaque écran de sélection de sous-niveaux. Pour le niveau « Porte », il y a 4 sous-niveaux possibles, pour « Hôtel » 3, dont un sous-niveau de transition se déroulant dans les transports en commun et pour « Distributeur » il y a 3 sous-niveaux également.

- **Un écran par sous-niveau** :

Pour chaque niveau, une partie graphique illustrera pour chaque sous-niveau les niveaux « Porte » et « Hôtel ». Les niveaux « Distributeur » utiliseront le même fond correspondant à un distributeur de boissons automatique. Chaque niveau permettra à l'utilisateur d'exploiter une vulnérabilité différente, avec une difficulté graduelle à l'intérieur de chaque niveau. Au début d'un sous-niveau, un écran « Aide » est automatiquement affiché, permettant de donner un contexte à l'utilisateur. Cet écran possède un bouton « Suivant », permettant de le réduire.

Sur chaque sous-niveau, 2 boutons seront accessibles :

- « **Retour** » : Il permet de retourner à l'écran de sélection de sous-niveaux.
- « **Aide** » : Il permet d'afficher à nouveau la fenêtre d'explication du sous-niveau à l'écran, qui pourra être fermée à l'aide du bouton « Suivant » qu'elle contient.

Le contexte, le but ainsi que la vulnérabilité exploitée dans chaque niveau ont été détaillés dans un document laissé en annexe dans le fichier *Document de conception.pdf*.

b. Maquette

Afin de réaliser une esquisse du jeu, nous avons conçu une maquette sur Adobe XD. Celle-ci nous a permis de nous accorder sur l'emplacement des boutons et de leur comportement ainsi que de celui du distributeur, qui représente la partie la plus complexe de notre travail.

Durant l'avancement du projet, nous avons réalisé nous-mêmes les graphismes du jeu sur Adobe Illustrator et Photoshop, et remplacé les illustrations génériques de la maquette par les nôtres.

4. DÉVELOPPEMENT

Le code source du projet est divisé en 2 parties : le programme exécuté par l'utilisateur est développé en Python, tandis que le lecteur MFRC522 est relié à une carte Arduino possédant son propre programme.

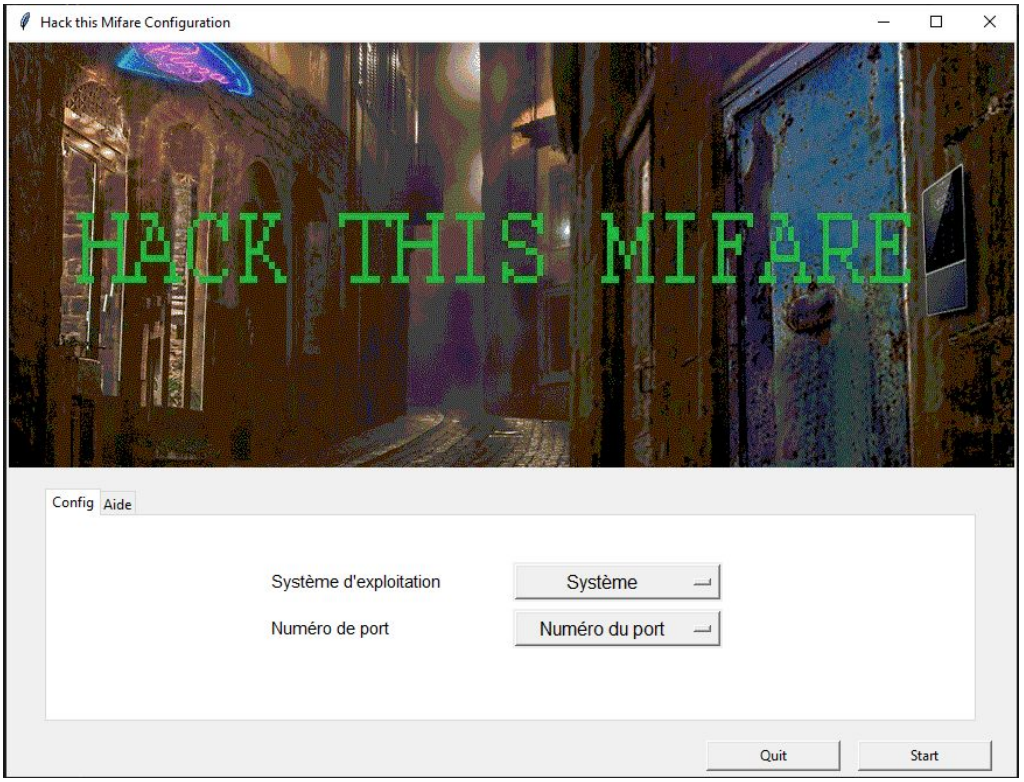
a. Python

i. Librairies

Les principales librairies utilisées sont Pygame, Tkinter, Pyserial et Thread.

Pygame est la librairie la plus utilisée et sert de base au jeu. Elle permet notamment de gérer les différents boutons de celui-ci.

Tkinter est une bibliothèque graphique permettant de créer des interfaces graphiques. Elle est utilisée pour afficher la fenêtre des paramètres du jeu.



PySerial est la librairie qui permet d'accéder au port série. Ici, elle sert à établir la liaison série entre le programme et la carte Arduino par l'intermédiaire du port COM sur lequel la carte est reliée.

Enfin, **Thread** est la librairie qui permet d'utiliser différents threads afin que le jeu puisse s'exécuter tout en écoutant les données reçues sur le port série.

ii. Classes

Pour notre programme, nous avons défini 9 classes que nous avons réunies dans un fichier *classe.py* et que nous importons dans le main.

Les classes sont réunies en **3 groupes** :

- le premier est celui des classes qui définissent tous les objets utilisés pour le jeu en lui-même comme les terrains, les portes, les boutons, les portiques et les panneaux de niveaux.

- le deuxième regroupe les classes dédiées au distributeur. Son fonctionnement étant plus complexe qu'une simple porte porte à ouvrir, nous avons eu besoin de définir une classe *Distributeur*, une classe *Item* décrivant les objets qu'il contient et une classe *Solde* qui permet de gérer le solde.

- le dernier, enfin, est composé d'une seule classe *Recevoir*, qui gère la liaison série en utilisant un thread.

iii. Fonctions

Notre programme possède des **fonctions principales** comme par exemple :

- *bouclePrincipale()* permet au jeu de s'exécuter : elle se compose de plusieurs boucles while correspondant chacune à un niveau du jeu, et d'une partie définition décrivant chaque objet composant les niveaux à l'aide des classes et des images présentes dans les fichiers du jeu.

- *selecteurNiveau()* permet de faire la liaison entre les différents niveaux tout en utilisant des objets graphiques. Elle est composée d'une partie où chaque objet est défini avant d'être utilisé dans une seconde partie, qui est une boucle while composée de plusieurs conditions permettant de choisir le niveau à exécuter.

- *intro()* permet de créer la fenêtre d'introduction du jeu, qui présente et explique son but ainsi que son fonctionnement.

- Enfin, les fonctions *start()*, *fctDeroulant1()* et *fctDeroulant2()* permettent de lancer le jeu et de créer les menus déroulants de la fenêtre des paramètres du jeu gérée avec Tkinter.

b. Arduino

i. Librairie MFRC522

Il existe une librairie Arduino développée spécifiquement pour le lecteur MFRC522 ainsi que les autres lecteurs RFID basés sur des modules RC522. Elle est disponible sur le github[1] de son auteur, miguelbalboa, même s'il n'en assure plus le maintien.

Une fois téléchargée, plusieurs exemples fournis avec celle-ci permettaient d'effectuer un dump mémoire, d'écrire, d'incrémenter, décrémenter et de lire une carte MIFARE. En nous basant sur les connaissances des protocoles de communication utilisés sur les cartes Classic, acquises lors de la réalisation de l'état de l'art, nous avons pu adapter et écrire plusieurs programmes correspondants chacun à un des niveaux que nous souhaitions.

ii. Communication avec la carte MIFARE

Afin de pouvoir accéder à un secteur d'une carte, plusieurs étapes vont être nécessaires.

Tout d'abord, on doit vérifier qu'une carte est sélectionnée avec *mfrc522.PICC_ReadCardSerial().*

On doit ensuite s'authentifier à un secteur avec mfrc522.PCD_Authenticate() : pour cela, nous devons indiquer la clé que l'on utilise, l'UID de la carte à laquelle nous essayons d'accéder ainsi que le *trailer block* du secteur, ou "sector trailer", contenant les clés et les conditions d'accès. Généralement, il est seulement nécessaire de s'authentifier avec la clé A, la clé B étant nulle par défaut. Cependant, lorsqu'elle ne l'est pas, il faut également s'authentifier avec cette même fonction, mais cette fois-ci avec la clé B.

Une fois ces opérations effectuées, l'accès à la mémoire devient possible grâce aux fonctions mfrc522.MIFARE_READ(), mfrc522.MIFARE_DECREMENT(), etc.
Néanmoins, la lecture de l'UID ne nécessite pas d'authentification au préalable et est lisible par tous : elle est réalisée en lisant le contenu de *mfrc522.uid.uidByte*.

iii. Structure du programme

Le premier problème majeur rencontré était de pouvoir gérer plusieurs niveaux, effectuant des actions différentes sur la carte, avec le même programme téléversé sur l'Arduino UNO. En effet, une fois le lecteur relié à l'Arduino UNO et celle-ci connectée au PC exécutant le jeu, l'utilisateur doit pouvoir évoluer dans les différents niveaux sans changer sa configuration ni changer le programme de la carte UNO entre chacun d'entre eux.

[1] https://github.com/miguelbalboa/rfid

Nous avons donc choisi de réaliser un “sélecteur de niveau” : pour chaque niveau du jeu, une fois dans la boucle du programme Python correspondante, un caractère différent est envoyé sur le port série ('a' pour le niveau 1 de la porte, 'b' pour le niveau 2 ...). Ainsi, après avoir initialisé les variables utilisées, on reste en attente de la lecture d’un caractère et, selon sa valeur, on effectue les instructions relatives au niveau souhaité. Afin de faire fonctionner indépendamment les niveaux, et en supposant qu’on utilise une seule carte MIFARE en lecture, les niveaux accèdent tous à des secteurs différents de la carte, et n’utilisent que le premier bloc de chaque secteur.

Le niveau 1 de la porte va lire et retourner au programme Python sur le port série l’UID uniquement, le niveau 2 ne retournera que le contenu du bloc 4 (secteur 1), etc.

Les niveaux “Porte” et “Hôtel” n’effectuent que des opérations de lecture sur la carte, mais les niveaux “Distributeur” nécessitent également de décrémenter et transférer des valeurs. Afin d’implémenter les vulnérabilités voulues, ces niveaux se comporteront de la façon suivante :

- **niveau 1 :** lecture simple du solde. Si le solde est inférieur au prix de la boisson choisie, aucune action n’est effectuée et l’utilisateur est informé que son solde est insuffisant. Sinon, le solde sur la carte est décrémenté du prix de la boisson, et un message est renvoyé indiquant la réussite de la transaction.

- **niveau 2 :** même chose que précédemment, mais le message de réussite de la transaction apparaît environ 300 ms avant l’opération de décrémentation (dépend des paramètres fixés). Cela permettra à l’utilisateur de retirer sa carte juste avant la décrémentation de son solde, afin qu’il ne soit jamais débité.

- **niveau 3 :** même chose que pour le premier niveau, mais au lieu de décrémenter directement le solde de la carte, un calcul est effectué en local sur l’Arduino UNO de la nouvelle valeur du solde. Il est ensuite écrit à l’emplacement désiré.

c. Documentation

Nous avons documenté le code source du projet avec Doxygen, un système de documentation pour plusieurs langages, dont C++ et Python. Il génère ainsi une documentation logicielle à partir de notre code source, en tenant compte de la syntaxe du langage utilisé ainsi que de nos commentaires.

De plus un manuel d'utilisation a été écrit pour accompagner le joueur dans l'installation et dans l'utilisation du jeu afin qu'il puisse l'exécuter de manière optimale. Ce manuel est disponible en annexe, avec l’état de l’art ainsi que le document de conception.

5. DIFFICULTÉS RENCONTRÉES

Durant le développement nous avons rencontré certains problèmes qui ont ralenti notre progression.

a. Retard et dysfonctionnement du matériel

Afin de pouvoir réaliser le jeu, nous avions besoin de cartes Arduino UNO ainsi que de lecteurs de cartes RFID/NFC. Ce matériel nous a été envoyé par notre tuteur de projet une fois le cadre de notre étude fixé. Malheureusement avec l'augmentation des délais postaux due à la crise sanitaire, nous avons reçu nos cartes Arduino assez tard dans le développement du projet. Nous avons malgré cela pu commencer à réaliser le programme en Python en travaillant avec des ports série virtuels, mais sans pouvoir tester l'intégration avec Arduino.

En plus de ce délai, une des cartes reçues était dysfonctionnelle et ne permettait de lire que l'UID, une erreur dans le protocole de communication étant retournée à chaque fois dans les autres cas, ce qui a également retardé le développement de l'Arduino.

b. Compatibilité du lecteur RFID

Afin de pouvoir casser les clés de chiffrement A et B des cartes MIFARE Classic, il faut utiliser des outils implémentant les attaques *nested* et *darkside* : MFOC et MFCUK.
MFOC permet de recouvrer toutes les clés de chiffrement d'une carte en n'en possédant qu'une seule, tandis que MFCUK permet de recouvrer une clé de chiffrement sans aucune condition.

Ces outils, indispensables pour pouvoir trouver les clés de chiffrement non utilisées sur certaines cartes afin de pouvoir réécrire celles-ci, ne peuvent-être utilisés avec le lecteur MFRC522. En effet, ils s'appuient sur la bibliothèque libnfc, et nécessitent donc un lecteur NFC, tandis que le lecteur à notre disposition était un lecteur RFID.
Il existe un projet d'adaptation de la bibliothèque libnfc aux lecteurs MFRC522, mais qui à ce jour n'est pas fonctionnelle, et dont le développement semble avoir été abandonné.

Nous n'avons donc pas pu accéder nous-mêmes aux cartes chiffrées avec des clés non triviales. Une alternative possible serait d'utiliser un module NFC PN532 à la place du MFRC522.

6. CONCLUSION

L'objectif principal de ce projet était de développer un jeu ludique sur PyGame ayant pour but de présenter et d'exploiter les vulnérabilités des systèmes à carte sans-contact MIFARE. Pour cela, nous avons d'abord dû étudier le fonctionnement de ces circuits intégrés, puis avons réalisé un document de conception afin de définir les différents scénarios de la vie courante à implémenter. Le choix des niveaux s'est fait en respectant le cahier des charges imposé par notre tuteur ainsi qu'en analysant les utilisations les plus fréquentes de ces produits.

Suite aux délais de livraison importants et à la situation exceptionnelle rencontrée, nous avons commencé le développement de l'application Python en émulant virtuellement une liaison série grâce à un émulateur afin de ne pas prendre de retard. Une fois les cartes reçues, nous avons pu commencer la programmation de la carte Arduino, ainsi que le contrôle des échanges de données entre le programme Python et la carte Arduino.

Ce projet nous a permis d'approfondir nos connaissances des produits de la gamme MIFARE, qui restent extrêmement populaires et toujours utilisés à travers le monde. Nous avons pu étudier à travers le fonctionnement des cartes MIFARE Classic un des principaux risques de la sécurité par l'obscurité, et le danger qu'un chiffrement propriétaire vulnérable peut créer.

Il nous a également permis d'améliorer nos compétences en développement, de manipuler des données échangées à travers un port série, et a été plus généralement une occasion de coordonner nos efforts malgré la complexité que le travail à distance pouvait laisser présager.