COMPTE-RENDU
TP CARTE SANS CONTACT

Programmation sous QT d'une interface graphique pour gérer une carte sans contact Mifare Classic

Aloïs MANDRY et Corentin FAUCHART
alois_mandry@etu.u-bourgogne.fr / corentin_fauchart@etu.u-bourgogne.fr
ESIREM Dijon 2021

Table des matières

Table des illustrations 1

Introduction 2

Connexion à la carte 3

Lecture des informations sur la carte 4

Ecriture des informations sur la carte 6

- **Identité** 6
- **Porte-monnaie** 7
- **Reset** 8

Conclusion 9

Table des illustrations

Figure 1 : Information secteur identité 2
Figure 2 : Information secteur porte-monnaie 2
Figure 3 : Code pour la fonction de connexion 3
Figure 4 : Documentation fonction de lecture d'un bloc 4
Figure 5 : Morceau de la fonction de lecture des données 5
Figure 6 : Morceau de la fonction d'écriture de l'identité 6
Figure 7 : Morceau de la fonction d'écriture du porte-monnaie 7
Figure 8 : Morceau de la fonction reset 8
Figure 9: Interface réalisée 9

GitHub : https://github.com/Corentin-Fauchart/TP-ComSansFil/

Introduction

Le but de ce TP est de développer l'interface graphique pour la gestion de la carte sans-contact **MIFARE Classic**. Une carte **MIFARE Classic** contient 16 secteurs de 4 blocs, un bloc contient 16 octets. Nous travaillerons dans ce TP seulement sur **deux secteurs** le 2 et le 3.

La carte sans-contact contient un secteur (2) de donnée de quatre blocs contenant l'identité de la personne à qui appartient la carte, c'est-à-dire son nom et son prénom ainsi que le titre du secteur (Identité).

Sector	Block	Data
2	11	KeyA+AccessBit+KeyB
	10	Nom
	9	Prenom
	8	« Indentite »

L'authentification se fera avec les clés suivantes :

- Key A : A0 A1 A2 A3 A4 A5
- Key B : B0 B1 B2 B3 B4 B5

Figure 1 : Information secteur identité

La carte contient aussi un autre secteur (3) de donnée qui correspond à un porte-monnaie avec un compteur et un backup ainsi que le titre du secteur (Porte-monnaie).

Sector	Block	Data
3	15	KeyA+AccessBit+KeyB
	14	Compteur
	13	Backup Compteur
	12	« Porte Monnaie »

L'authentification se fera avec les clés suivantes :

- Key A : C0 C1 C2 C3 C4 C5
- Key B : D0 D1 D2 D3 D4 D5

Figure 2 : Information secteur porte-monnaie

De plus pour chaque bloc, il lui est attribué un « **sector trailer** » (en jaune sur les images ci-dessus) qui dicte les droits d'écriture et de lecture avec les différentes clés.

Nous devons donc développer une solution graphique pour traiter ces différentes données en lecture et écriture. Nous expliquerons brièvement la solution proposée dans ce rapport en omettant quelques subtilités tels que l'allumage des LEDs disponible sur le **GitHub**.

Connexion à la carte

```
void MainWindow::on_button_Connect_clicked()
{
    uint16_t status = 0;

    MonLecteur.Type = ReaderCDC;
    MonLecteur.device = 0;
    status = OpenCOM(&MonLecteur);
    if(status==0){
        ui->affichageCarte->update();
        ui->button_Connect->setEnabled(false);
        ui->button_Deconnexion->setEnabled(true);
        ui->button_Reset->setEnabled(true);
        ui->button_Lecture1->setEnabled(true);
        status = LEDBuzzer(&MonLecteur, LED_RED_ON);
    }
    qDebug() << "OpenCOM1" << status;

    RF_Power_Control(&MonLecteur, TRUE, 0);
    status = Mf_Classic_LoadKey(&MonLecteur, Auth_KeyA, key_sec2Lec, 2);
    status = Mf_Classic_LoadKey(&MonLecteur, Auth_KeyB, key_sec2Ecr, 2);
    status = Mf_Classic_LoadKey(&MonLecteur, Auth_KeyA, key_sec3Lec, 3);
    status = Mf_Classic_LoadKey(&MonLecteur, Auth_KeyB, key_sec3Ecr, 3);
    status = Version(&MonLecteur);
    ui->affichageCarte->setText(MonLecteur.version);
}
```

Figure 3 : Code pour la fonction de connexion

Pour réaliser la connexion a une carte **MIFARE Classic** quelques prérequis sont nécessaires, tout d'abord nous devons définir le type de lecteur utilisé, dans notre cas un **Coupleur USB RFID** Sans Contact de la marque **Odalid**. Il nous permet d'utiliser une carte ou périphérique NFC avec le standard **ISO14443A**. L'interface utilise un type de USB 2.0 CDC interopérable avec tous les ordinateurs.

Par la suite nous lançons la communication entre l'appareil et l'ordinateur avec la fonction **OpenCOM**, puis activons le champs RF (Radiofréquences) avec la fonction **RF_Power_Control** pour pouvoir communiquer avec la carte. Il nous faut maintenant nous authentifier, deux méthodes sont possibles **Mf_Classic_Authenticate** et **Mf_Classic_LoadKey** nous avons choisi de nous authentifier avec la seconde methode qui permet aussi de charger nos quatre clés d'authentifications dans les différents secteurs pour la lecture et l'écriture (déterminer dans le sector trailer).

Dans notre cas les informations de sector trailer ont été programmées pour que les **clés A** soient utilisées pour l'écriture, et les **clés B** soient utilisées pour la lecture des informations.

Si l'opération est réussie l'interface se met à jour et une **LED rouge** s'allume sur le lecteur **Odalid**.

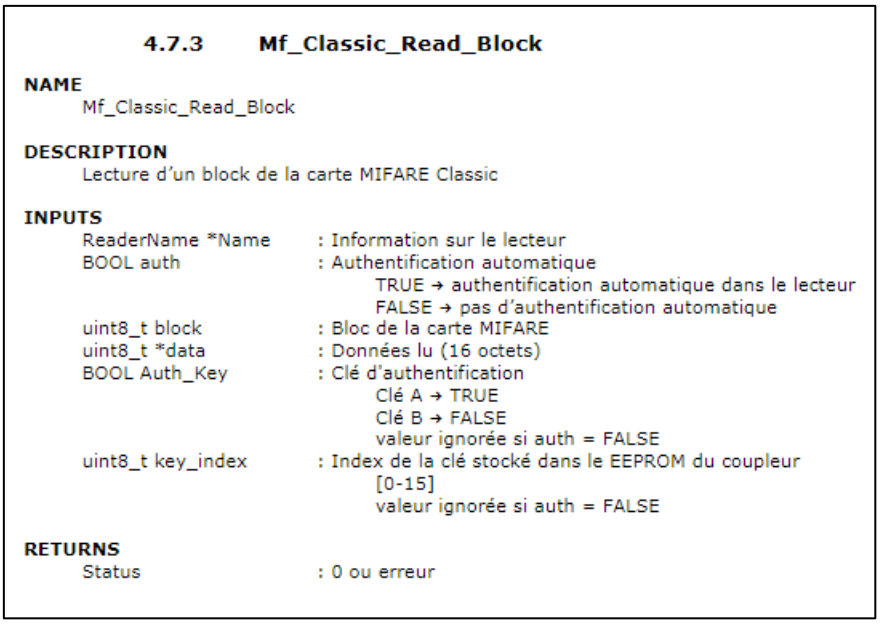
Lecture des informations sur la carte

Il y a plusieurs informations enregistrées sur la carte. Dans le sujet du TP, il nous a été demandé de lire les données stockées dans le secteur 2 et 3 de la carte. Respectivement l'identité du propriétaire et un compteur représentant ici son argent comme expliqué dans l'introduction.

Nous avons recherché dans la documentation les fonctions nous permettant de lire sur la carte. Il est possible de lire soit toutes les données du secteur soit bloc par bloc. Nous avons opté pour cette seconde option qui nous a paru plus adapté à nos besoins.

Avant de pouvoir lire sur la carte, il faut tout d'abord effectuer la prise de contact avec la carte pour lui indiquer que nous allons effectuer une action. Pour cela, nous utilisons la fonction **ISO14443_3_A_PollCard** qui effectue la prise de contact selon la norme **ISO14443A** avec un **Request standard**.

Pour lire un bloc de données de la carte, nous utilisons la fonction décrite ci-dessous :



4.7.3 Mf_Classic_Read_Block

NAME
Mf_Classic_Read_Block

DESCRIPTION
Lecture d'un block de la carte MIFARE Classic

INPUTS

ReaderName *Name	: Information sur le lecteur
BOOL auth	: Authentification automatique TRUE → authentification automatique dans le lecteur FALSE → pas d'authentification automatique
uint8_t block	: Bloc de la carte MIFARE
uint8_t *data	: Données lu (16 octets)
BOOL Auth_Key	: Clé d'authentification Clé A → TRUE Clé B → FALSE valeur ignorée si auth = FALSE
uint8_t key_index	: Index de la clé stocké dans le EEPROM du coupleur [0-15] valeur ignorée si auth = FALSE

RETURNS

Status	: 0 ou erreur

Figure 4 : Documentation fonction de lecture d'un bloc

Nous devons préciser à la fonction le lecteur, un booléen (que l'on mettra tout le temps à TRUE pour l'authentification automatique) pour sélectionner notre mode d'authentification, le numéro du bloc que l'on souhaite lire, un tableau qui récupèrera les données lues, un booléen permettant de sélectionner la clé d'authentification utilisée (clé A ou clé B) et enfin un nombre entier permettant d'indiquer à la fonction qu'elle clé elle devra prendre pour effectuer l'opération.

On souhaite donc lire les informations contenues dans le secteur 2 et 3. Plus précisément, l'identité du propriétaire de la carte (nom et prénom) correspondant aux blocs 9 et 10 et le compteur correspondant au bloc 14.

```
status = ISO14443_3_A_PollCard(&MonLecteur, atq, sak, uid, &uid_len);

//identité
status = Mf_Classic_Read_Block(&MonLecteur, TRUE, 9, data, Auth_KeyA, 2);
strcpy(data2, (char*)data);
ui->affichageInfo2->setText(data2);

status = Mf_Classic_Read_Block(&MonLecteur, TRUE, 10, data, Auth_KeyA, 2);
strcpy(data3, (char*)data);
ui->affichageInfo3->setText(data3);

//porte-monnaie
status = Mf_Classic_Read_Value(&MonLecteur, TRUE, 14, &dataMonnaie, Auth_KeyA, 3);


ui->affichageMonnaie->setText(QString::number((int32_t)dataMonnaie));
```

Figure 5 : Morceau de la fonction de lecture des données

C'est ce que l'on fait dans la capture d'écran ci-dessus. Nous faisons la prise de contact avec la carte et ensuite nous effectuons nos opérations. Nous lisons **le bloc 9 et 10** pour le prénom et le nom en spécifiant à la fonction le numéro de secteur ainsi que la clé que l'on souhaite utiliser pour l'opération. Ici nous utilisons la clé A car nous souhaitons lire sur la carte.

Les données sont inscrites dans notre tableau de données que nous lui donnons en paramètre, ici notre **variable « data ».**

Après avoir récupéré ces données, nous les affichons sur notre interface pour pouvoir les visualiser simplement en un clic.

Puis nous effectuons la même opération pour le compteur en n'oubliant pas de changer le numéro de secteur (le compteur se situe sur le secteur 3, et non le 2).

Nous utilisons cette fois-ci la fonction **Mf_Classic_Read_Value** qui prend des paramètres similaires à la fonction vu précédemment, car ce secteur (Porte-monnaie) ne contient pas de données textuelles mais des nombres entiers.

Si l'opération est réussie l'interface se met à jour et une **LED Jaune** s'allume sur le lecteur **Odalid** pendant une seconde. (Voir GitHub pour précision du code)

Ecriture des informations sur la carte

Identité

```
status = ISO14443_3_A_PollCard(&MonLecteur, atq, sak, uid, &uid_len);
char DataIn[16];
sprintf(DataIn, ui->affichageInfo2->toPlainText().toUtf8().data(), 16);
status = Mf_Classic_Write_Block(&MonLecteur, TRUE, 9, (uint8_t*)DataIn, Auth_KeyB, 2);

sprintf(DataIn, ui->affichageInfo3->toPlainText().toUtf8().data(), 16);
status = Mf_Classic_Write_Block(&MonLecteur, TRUE, 10, (uint8_t*)DataIn, Auth_KeyB, 2);
```

Figure 6 : Morceau de la fonction d'écriture de l'identité

Il est important de savoir lire les informations stockées sur notre carte sans contact, mais pouvoir les modifier l'est tout autant. Pour écrire dans un bloc, nous allons utiliser la fonction **Mf_Classic_Write_Block**. Elle prend en paramètre les mêmes éléments que la fonction utilisée pour la lecture. Il faudra juste penser à utiliser la clé B sinon l'opération ne sera pas effectuée.

Pour la modification de l'identité, on récupère les informations inscrites par l'utilisateur sur notre interface dans un tableau de caractère et nous ajustons les données au format adapté pour les spécifications de la carte.

Si l'opération est réussie l'interface se met à jour et une **LED Verte** s'allume sur le lecteur **Odalid** pendant une seconde. (Voir GitHub pour précision du code).

Porte-monnaie

```
if(choixAction)
{
    status = Mf_Classic_Increment_Value(&MonLecteur,TRUE,14,valOpe,13,Auth_KeyB,3);
    status = Mf_Classic_Restore_Value(&MonLecteur,TRUE,13,14,Auth_KeyB,3);


}else{
    status = Mf_Classic_Decrement_Value(&MonLecteur,TRUE,14,valOpe,13,Auth_KeyB,3);
    status = Mf_Classic_Restore_Value(&MonLecteur,TRUE,13,14,Auth_KeyB,3);

}
```

Figure 7 : Morceau de la fonction d'écriture du porte-monnaie

Le porte-monnaie est légèrement différent du bloc identité car il ne comporte pas de valeur textuelle mais des entiers signer qu'il faut incrémenter et décrémenter suivant l'opération voulu. Pour cela nous utiliser les fonctions **Mf_Classic_Increment_Value** et **Mf_Classic_Decrement_Value** qui prend en paramètre le lecteur, les modes d'authentification comme vu précédemment, puis les informations de bloc et la valeur que nous voulons modifier.

Juste ces fonctions seules ne suffisent pas, notre opération est simplement affectée dans le backup de portemonnaie (bloc 13), il nous faut maintenant placer la valeur du backup dans le bloc 14 c'est-à-dire le compteur du porte-monnaie. Pour cela nous utilisons la fonction **Mf_Classic_Restore_Value** qui prend comme toujours des paramètres similaires avec la clef B car nous somme en écriture.

Si l'opération est réussie l'interface se met à jour et une **LED Verte** s'allume sur le lecteur **Odalid** pendant une seconde. (Voir GitHub pour précision du code)

Reset

```
status = ISO14443_3_A_PollCard(&MonLecteur, atq, sak, uid, &uid_len);
char DataIn[16];
sprintf(DataIn, "Vincent", 16);
status = Mf_Classic_Write_Block(&MonLecteur, TRUE, 9, (uint8_t*)DataIn, Auth_KeyB, 2);

sprintf(DataIn, "Thivent", 16);
status = Mf_Classic_Write_Block(&MonLecteur, TRUE, 10, (uint8_t*)DataIn, Auth_KeyB, 2);

uint32_t NoMoney = 0;

status = Mf_Classic_Write_Value(&MonLecteur,TRUE,13,NoMoney,Auth_KeyB,3);
status = Mf_Classic_Restore_Value(&MonLecteur,TRUE,13,14,Auth_KeyB,3);

status = LEDBuzzer(&MonLecteur, LED_ON);
QThread::sleep(1);
status = LEDBuzzer(&MonLecteur, LED_OFF);
status = LEDBuzzer(&MonLecteur, LED_RED_ON);
```

Figure 8 : Morceau de la fonction reset

Dans la fonction reset, nous souhaitons remettre les valeurs inscrites par défaut sur notre carte dans nos 2 secteurs. Pour cela, nous allons utiliser les mêmes méthodes utilisées pour l'écriture. Les données seront juste prédéfinies de base dans le code. C'est-à-dire, nous plaçons dans le bloc 9 nom : « **Thivent** » et dans le bloc 10 prénom « **Vincent** ».

Pour le compteur, au lieu d'incrémenter ou de décrémenter la valeur sauvegardée, nous allons directement modifier la valeur pour la remettre à 0. Grâce à la fonction, nous plaçons cette valeur dans le bloc 13 (bloc de backup) puis nous appelons la fonction Restore comme expliquée précédemment.

Toutes ces opérations n'ont pas pu être vérifiées durant les séances de TP, il se peut donc que quelques subtilités notamment le format soit invalide. Mais nous pensons que notre approche est correcte.

Si l'opération est réussie l'interface se met à jour et les 3 **LEDs de couleur** s'allument sur le lecteur **Odalid** pendant une seconde.

Conclusion

Nous avons découvert lors de ses séances de TP les principes pour interagir avec une carte sans fils **MIFARE Classic**. La compréhension des blocs et secteurs était indispensable pour pouvoir traiter les données de la carte correctement mais a été simplifier grâce aux nombreuses fonctions proposées par **Odalid** dans les spécifications techniques ainsi que dans nos séances de TD et CM.

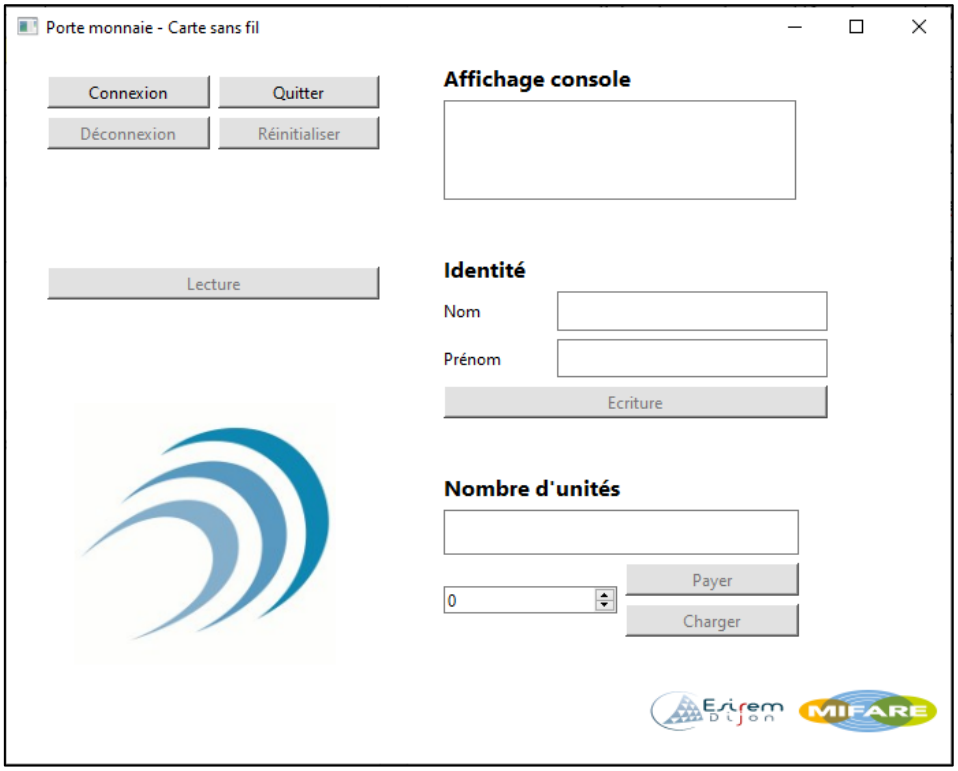


Figure 9: Interface réalisée