

Série n°1

Certains éléments de cette série sont issus des TP de Samia BOUZEFRANE & Khaled GARRI.

Avant Propos

1. Introduction

Les normes ISO 7816 « Identification cards – Integrated circuit cards with contacts » ont été publiées par l'organisation internationale de normalisation (ISO, International Organisation for Standardisation). C'est le plus important standard définissant les caractéristiques des cartes à puce qui fonctionnent avec un contact électrique. Sachant que 15 normes sont proposées pour les cartes à contact, nous décrivons brièvement ici uniquement les 7 premières normes.

1.1 ISO 7816-1

Cette norme définit les caractéristiques physiques des cartes à puce à contact : la géométrie, la résistance, les contacts, etc.

1.2 ISO 7816-2

Cette norme spécifie le dimensionnement physique (extérieur) des contacts de la puce. Deux des huit contacts réservés à une utilisation future (RFU) sont redéfinis pour l'utilisation USB dans la norme ISO 7816-12.

1.3 ISO 7816-3

Cette norme définit l'interface électrique et les protocoles de transmission :

- les protocoles de transmission (TPDU, Transmission Protocol Data Unit) : T=0 : protocole orienté octet, T1 : protocole orienté paquet, T=14 : réservé pour les protocoles propriétaires.
- la sélection d'un type de protocole.
- la réponse à un reset (ATR, ou Answer To Reset en anglais) qui correspond aux données envoyées par la carte immédiatement après la mise sous tension.
- les signaux électriques, tels que le voltage, la fréquence d'horloge et la vitesse de communication.

1.4 ISO 7816-4

Cette norme vise à assurer l'interopérabilité des échanges. Elle définit les messages APDU (Application Protocol Data Units), par lesquels les cartes à puce communiquent avec le lecteur. Les échanges s'effectuent en mode client-serveur, le terminal ayant toujours l'initiative de communication.

1.5 ISO 7816-5

Cette norme définit le système de numérotation et les procédures d'enregistrement et d'attribution des identifiants des applications (AID, ou Application Identifier). Un unique AID est associé à chaque application et à certains fichiers sur la carte. Ils sont représentés par des tableaux d'octets de taille allant de 5 à 16. Les cinq premiers octets représentent le numéro d'enregistrement du fournisseur d'application (RID, Registered Application Provider Identifier en anglais) qui est

attribué par la Copenhagen Telephone Company Ltd ou l'ISO. Ils sont suivis par l'identifiant optionnel PIX (Proprietary Application Identifier eXtension) d'une longueur allant jusqu'à 11 octets.

L'identifiant RID est le même pour le paquetage et l'applet, mais le PIX doit être différent.

1.6 ISO 7816-6

Cette norme spécifie des éléments de données inter-industrie pour les échanges, tels que le numéro du porteur de carte, sa photo, sa langue, la date d'expiration, etc.

1.7 ISO 7816-7

Cette norme définit les commandes inter-industrie pour langage d'interrogation de carte structurée (SCQL).

2. ATR (Answer To Reset) défini dans l'ISO 7816-3

Dès que la carte est mise sous tension, elle envoie un message de réponse d'initialisation appelé ATR, il peut atteindre une taille maximale de 33 octets. Il indique à l'application cliente les paramètres nécessaires pour établir une communication avec elle. Il fournit un nombre varié de paramètres liés à la carte et au protocole de transmission utilisé :

- Le protocole de transport ;
- Taux de transmission des données ;
- Numéro de série de la puce ...

Le premier octet noté TS = "3F" pour convention indirecte ou "3B" pour convention directe.

3. Echange de commandes avec le lecteur de carte à puce tel que défini dans l'ISO 7816-4

La communication entre l'hôte et la carte est half-duplex. Elle se fait à l'aide de paquets appelés APDU (Application Protocol Data Units) en respectant le protocole de l'ISO 7816-4. Un APDU contient une commande ou une réponse. Le mode Maître/Esclave est utilisé. Ainsi la carte joue un rôle passif et attend une commande APDU à partir de l'hôte. Elle exécute l'instruction spécifiée dans la commande et retourne une réponse APDU.

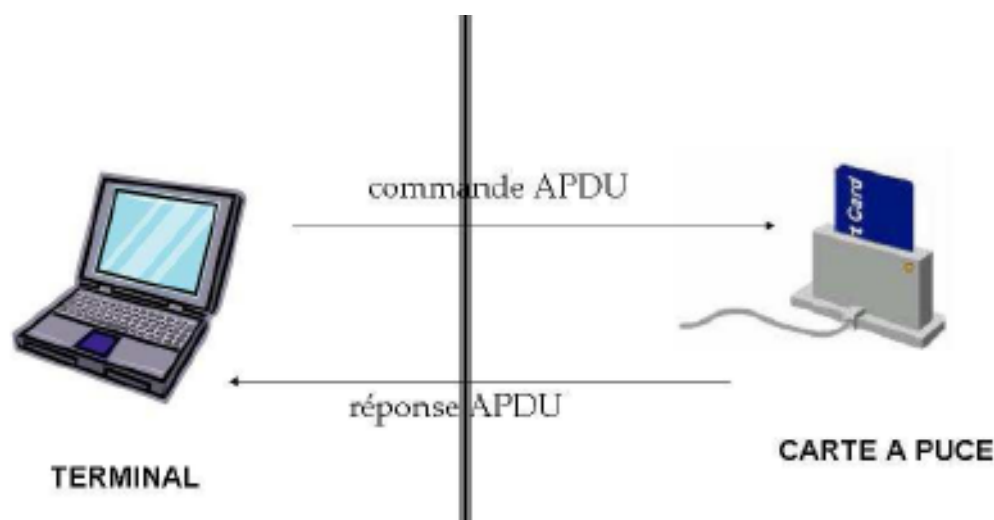


Figure 1 : Le modèle de communication de la carte à puce

3.1. Format des commandes APDU

Commande APDU						
Entête obligatoire				Corps optionnel		
CLA	INS	P1	P2	Lc	Data field	Le
➤ CLA (1 octet): Classe d'instructions : indique la structure et le format pour une catégorie de commandes et de réponses APDU. ➤ INS (1 octet): code d'instruction: spécifie l'instruction de la commande. ➤ P1 (1 octet) et P2 (1 octet): paramètres de l'instruction. ➤ Lc (1 octet): nombre d'octets présents dans le champ données de la commande. ➤ Data field (octets dont le nombre est égal à la valeur de Lc): une séquence d'octets dans le champ données de la commande.						

3.2. Format des réponses APDU

Réponse APDU		
Corps optionnel		Partie obligatoire
Data field		SW1 SW2
➤ Data field (longueur variable): une séquence d'octets reçus dans le champ données de la réponse. ➤ SW1 (1 octet) et SW2 (1 octet): Status words (Mots d'état) état de traitement par la carte.		

4. Exemples de cartes

Le tableau suivant donne des exemples de commandes APDU utilisées dans le monde de la carte.

Champ de la commande	Valeurs
CLA	BC = cartes de crédit françaises, cartes vitales françaises, A0 = cartes SIM (téléphonie)
INS	20 = présentation du PIN, 40 = validation (ratification du code PIN) B0 = Lecture B2 = Lecture de record D0 = Ecriture DC = Ecriture de record A4 = Sélection du répertoire (directory) C0 = Demander une réponse (get an answer)
P1, P2	paramètres contenant des adresses à lire
LEN	longueur prévue pour la réponse ou bien longueur de l'argument de l'instruction.
ARG	contient LEN octets (octets à écrire, PIN à vérifier, etc.)

La réponse APDU sert à accuser réception la commande APDU envoyée par le terminal. Ainsi, la carte répond en envoyant le code instruction INS, suivi de données de longueur LEN en terminant par SW1 et SW2 (0x90 0x00 lorsque la commande s'est déroulée avec succès). En cas d'échec, seuls les champs SW1 et SW2 seront envoyés au terminal avec les codes d'erreur suivants :

0x6E 0x00	CLA error
0x6D 0x00	INS error
0x6B 0x00	P1, P2 error
0x67 0x00	LEN error
0x98 0x04	Bad PIN
0x98 0x08	Unauthorized Access
0x98 0x40	Card blocked

...

Exercice 1 (Exploration d'une carte bancaire EMV)

Au début, sachez bien qu'il n'y a aucun risque d'interroger votre carte bancaire (envoyer des commandes APDUs), car la plupart du temps, il s'agit juste de lecture mémoire, sauf bien entendu, pour quelques instructions telles que la ratification du code PIN (Personal Identification Number) qui s'il est saisi de façon erronée trois fois de suite risque de bloquer la carte.

Dans ce TP, nous explorons l'application bancaire EMV (et non plus, feu, l'application bancaire française B0'). Les spécifications sont publiques et téléchargeable ici : <http://www.emvco.com/specifications.aspx?id=223>

Il faut télécharger le Book 3 !

But de l'exercice :

- Lecture des informations de la carte (prénom, nom, numéro de la carte, etc.)

Sous Windows, nous utiliserons TestResMan pour dialoguer avec la carte.

- 1) Trouvez une facturette et chercher l'AID dessus !
- 2) Sélectionnez l'application EMV.
- 3) Récupérez les données avec la commande GET_RESPONSE.

En effet, la prochaine étape après la sélection consistera à envoyer la commande “GET PROCESSING OPTIONS” (GPO) pour récupérer l'*Application Interchange Profile* (AIP) et l'*Application File Locator* (AFL). Or, pour envoyer la commande GPO, vous aurez besoin de la *Processing Data Objects List* (PDOL) pour le champ de données de la commande GPO. La PDOL est une partie de la réponse à la sélection de l'application comme vous pouvez le voir dans la spécification.

Tag	Value		Presence
'6F'	FCI Template		M
	'84'	DF Name	M
	'A5'	FCI Proprietary Template	M
	'50'	Application Label	O
	'87'	Application Priority Indicator	O
	'9F38'	PDOL	O
	'5F2D'	Language Preference	O
	'9F11'	Issuer Code Table Index	O
	'9F12'	Application Preferred Name	O
	'BF0C'	FCI Issuer Discretionary Data	O
	'9F4D'	Log Entry	O
	'XXXX'	1 or more additional proprietary data elements from an application provider, issuer, or IC card supplier, or EMV-defined tags that are specifically allocated to 'BF0C'	O

Le « tag » de la PDOL est “9F38” et la PDOL est optionnelle. Peut-être que dans la réponse de votre carte, il n'y aura pas de PDOL. Ne vous inquiétez pas, cela est courant.

- 4) Envoyez la commande GPO et récupérez les données avec la commande GET_RESPONSE.

Déterminez en fonction de ce que vous trouverez dans les spécifications l'AFL le ou les SFI (Short File Identifier) et les records associés dans lesquels vous pouvez lire des données.

- 5) Une fois les SFI et records déterminés, lisez les données avec la commande READ RECORD et essayez de trouver votre numéro de carte bancaire, votre nom et prénom et la date d'expiration.

Vous trouverez des manipulations similaires à celles de ce TP sur ce blog :
<http://blog.saush.com/2006/09/08/getting-information-from-an-emv-chip-card/>

Exercice 2 (Exploration d'une carte SIM)

Pour l'exploration de la carte nous nous baserons sur les spécifications « 3GPP TS 11.11 » version 8.13.0 (2005-06). Elles vous sont données en PJ.

Il est préférable de désactiver le code PIN de la carte GSM avec votre téléphone portable.

La conversion du PIN s'effectue selon le mode suivant :

```
PIN :   1     2     3     4
CHV :  31    32    33    34  FF FF FF FF
```

La séquence de commandes APDU sera la suivante :

```
SELECT MF
AO A4 00 00 02 3F 00
SELECT DF Telecom (7F10)
AO A4 00 00 02 7F 10
SELECT EF ADN (6F3A) (Abbreviated Dialing Number)
AO A4 00 00 02 6F 3A ----> 9F0F
GET RESPONSE
AO C0 00 00 0F ----> Record size (juste avant le 9000)
VERIFY CHV
AO 20 00 01 08 + PIN converti ----> 9000/9808
LECTURE
AO B2 (N°) 04 (Record Size)
UPDATE RECORD
AO DC (N°Rec) 04 (Record Size)
```

1. Utilisez votre téléphone portable pour retenir quelques entrées de votre répertoire téléphonique. Vérifiez que les noms/numéros que vous avez choisi sont bien enregistrés sur votre carte SIM et non sur votre téléphone (Votre téléphone fait le croisement de toutes les entrées dont il dispose : celle de sa mémoire interne et celle de la puce de la carte).
2. Désactivez sur votre téléphone la vérification du PIN (Menu Sécurité). A tout hasard, coder votre PIN selon la méthode présentée plus haut.
3. Sélectionner le Fichier Elémentaire contenant votre répertoire téléphonique.
4. Lisez la première entrée de ce répertoire et décomposez-la : vous devez trouver le nom codé en ASCII ainsi que le SWAP de votre numéro de téléphone (= inversé par octet):

```
pour le numéro de téléphone 06 01 02 03 04
taille  TON/NPI  06  01  02  03  04
      06      81      60  10  20  30  40
```

```
ou encore pour le numéro de téléphone 950
taille  TON/NPI  95  0
      03      81      59  F0  FF
```

5. rajoutez ou modifiez une entrée de votre répertoire téléphonique.

Vous pouvez également consulter le fichier écrit en Perl :

<http://ludovic.rousseau.free.fr/software/SIM-2.0.tar.gz>

Pour aller plus loin (chez vous) ! Vous pouvez installer CardPeek :

<https://code.google.com/p/cardpeek/>

Exercice 3 (Programmation d'une applet faisant un echo de l'entrée)

Dans le cadre de cet exercice nous utiliserons des cartes JCOP et le kit de développement fourni avec. Il s'agit d'un plugins pour Eclipse. Le kit est déjà installé sur votre poste.

Télécharger [6] et installer les spécifications sur votre machine dans votre compte utilisateur.

Nous allons programmer une application faisant un echo de l'APDU reçu. On ajoutera la contrainte que si les deux premiers octets de la commande APDU sont égaux à 0x80 alors la carte renverra seulement 0x6B 0x00 (pour cela on utilisera la classe `ISOException`).

Le but de l'exercice est de comprendre les mécanismes de réception de commande APDU et d'envoi de réponse APDU et de Status Word (SW).

Voici le code minimale d'une applet en Java Card.

```
package applets.NullApp;

import javacard.framework.*;

public class NullApp extends Applet
{
    protected NullApp()
    {
        // Insérer ici les initiatisations
    }

    public static void install(byte[] bArray, short bOffset, byte bLength) throws ISOException
    {
        new NullApp().register();
    }

    public void process(APDU apdu) throws ISOException
    {
        // Insérer ici le code métier
    }
}
```

On compilera puis convertira l'applet d'écho obtenue pour ensuite la charger sur la carte. Son installation se fait en deux phases qui sont le chargement et l'instanciation (appel à la méthode statique `install`).

[6] <http://java.sun.com/products/javacard/>

Exercice 4 (Programmation d'une applet porte-monnaie électronique)

Les spécifications de cette applet sont les suivantes :

- Elle comportera une balance (le solde) exprimé sous forme d'un `short`.
- Elle comportera un PIN qui devra être vérifié pour les opérations de crédit et de débit. On utilisera pour ce PIN la classe `OwnerPin` qui fait tout le travail de gestion du PIN pour nous et de manière sécurisée. Nous le fixerons à 8 octets la longueur et à 3 le nombre maximal d'essai consécutifs faux.
- Elle acceptera 4 commandes `GET_BALANCE`, `CREDIT`, `DEBIT` et `VERIFY` :
 - `GET_BALANCE` renverra le solde ;

GET BALANCE APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0x80	0x50	0x0	0x0	N/A	N/A	2
• The data field contains the balance amount						
Response APDU						
Optional data		Status word		Meaning of status word		
Balance value		0x9000		Successful processing		

- `VERIFY` permettra de proposer un PIN à vérifier ;

VERIFY APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0x80	0x20	0x0	0x0	Length of the PIN data	PIN data	N/A
• CLA byte denotes the structure of the command • INS byte (0x20) indicates a verify instruction • P1 and P2 are not used, and are both set to 0 • The data field contains the PIN value						
Optional data		Status word		Meaning of status word		
N/A		0x9000		Successful processing		
		0x6300		Verification failed		

- CREDIT permettra de créditer la balance et ne fonctionnera que si le PIN a été vérifié avec succès ;

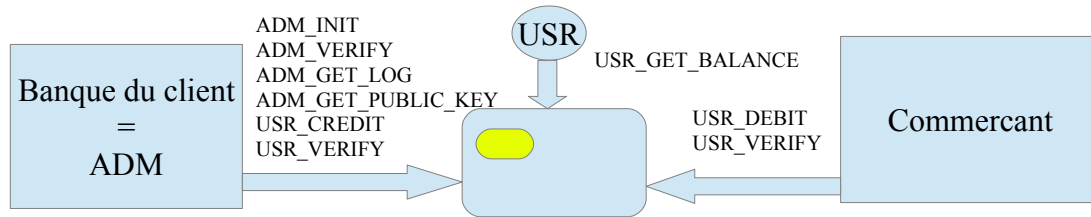
CREDIT APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0x80	0x30	0x0	0x0	1	Credit amount	N/A
The data field contains the credit amount						
Response APDU						
Optional data		Status word		Meaning of status word		
N/A		0x9000		Successful processing		
		0x6301		PIN verification required		
		0x6A83		Invalid credit amount		
		0x6A84		Exceed the maximum amount		

- DEBIT permettra de débiter la balance et ne fonctionnera que si le PIN a été vérifié avec succès.

DEBIT APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0x80	0x40	0x0	0x0	1	Debit amount	N/A
The data field contains the debit amount						
Response APDU						
Optional data		Status word		Meaning of status word		
N/A		0x9000		Successful processing		
		0x6301		PIN verification required		
		0x6A83		Invalid debit amount		
		0x6A85		Negative balance		

Il suffit donc de programmer cette application simple. Vous trouverez dans la classe `Util` des méthodes intéressantes pour la manipulation de `short` et de tableau de `bytes`. L'interface `ISO7816` dans le paquetage `javacard.framework` contient déjà beaucoup des constantes dont vous aurez besoin pour retourner les erreurs.

Exercice 5 (Programmation d'une applet porte-monnaie électronique) : le projet



Les spécifications de cette applet sont les suivantes :

- L'applet comportera une balance (le solde) comptabilisant les euros et les centimes d'euros, qui ne pourra jamais être négative et qui sera initialisé à 100,00 euros. Le solde maximum sera de 65535,99 euros.
- Les opérations de débit et de crédit seront signées par la carte pour des raisons de traçabilité pour le commerçant (on signera le solde courant avec le montant à débiter/créditer et le montant après opération). On utilisera le RSA d'un hash SHA bourré suivant le standard PKCS#11 pour les signatures.
- Les opérations de débit et crédit seront enregistrées sur la carte dans un fichier circulaire de logs permettant de conserver les 10 dernières transactions pour que la banque puisse faire des vérifications en cas de contestation (on stockera donc le solde courant, le montant à débiter/créditer, le montant après opération et la signature calculée).
- On pensera bien évidemment à s'assurer de l'atomicité de l'opération de débit et de crédit pour éviter les attaques d'arrachage.
- L'applet comportera un PIN utilisateur qui devra être vérifié pour les opérations de crédit et de débit. On utilisera pour ce PIN la classe `OwnerPin` qui fait tout le travail de gestion du PIN pour nous et de manière sécurisée. Nous le fixerons à 8 octets la longueur et à 3 le nombre maximal d'essais consécutifs faux.
- L'applet acceptera 4 commandes de niveau utilisateur `USR_GET_BALANCE`, `USR_CREDIT`, `USR_DEBIT` et `USR_VERIFY` et 4 commandes de niveau administrateur `ADM_INIT`, `ADM_GET_PUBLIC_KEY`, `ADM_GET_LOG` et `ADM_VERIFY`.
- Les fonctions `ADM_INIT`, `ADM_GET_PUBLIC_KEY` et `ADM_GET_LOG` ne seront disponibles qu'après vérification d'un PIN administrateur (de taille 8 octets et disposant de 1 seul essai) dont la valeur sera initialisée lors de l'installation.
- La paire de clés nécessaires à la signature sera générée sur la carte par la fonction `ADM_INIT` et tant que cette fonction n'aura pas été appelée aucune commande utilisateur ne sera accessible. On choisira une paire de clés RSA sur 512 bits avec le format CRT pour la clé privée (la carte utilisée ne sait pas en générée dans un autre format – cf. la spécification de la carte).
- La fonction `ADM_INIT` permettra d'initialiser le PIN utilisateur (et éventuellement de le débloquent) et de fixer le solde à une valeur si besoin.
- La fonction `ADM_GET_PUBLIC_KEY` permettra de récupérer la clé publique RSA.
- La fonction `ADM_GET_LOG` permettra de récupérer une entrée dans le fichier de logs.
- Pour résumer elle acceptera 8 commandes `ADM_INIT`, `ADM_GET_PUBLIC_KEY`, `ADM_GET_LOG`, `ADM_VERIFY`, `USR_GET_BALANCE`, `USR_CREDIT`, `USR_DEBIT` et `USR_VERIFY` (voir les spécifications ci-après).

Conseils :

- Choisissez les structures de données dont vous avez besoin et regardez les APIs
- Écrivez ensuite le code de dispatch des commandes.

- Remplissez les différentes méthodes par le code métier.

Remarques :

- On essayera d'économiser le nombre d'octets employés (comme sur une application réelle) sauf si bien sûr cela peut avoir des considérations sécuritaires négatives.
- Utilisez l'interface ISO7816 dans le paquetage `javacard.framework` qui contient beaucoup des constantes dont vous aurez besoin pour retourner les erreurs.

Vous n'avez pas à me rendre de rapport. Commentez un minimum votre code quand cela est nécessaire (inutile de paraphraser les opérations).

- `USR_GET_BALANCE` renverra le solde

USR_GET_BALANCE APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0xB0	0x50	0x0	0x0	N/A	N/A	Balance length
Response APDU						
Optional data		Status word		Meaning of status word		
Balance amount		0x9000		Successful processing		

- `USR_VERIFY` permettra de proposer un PIN utilisateur à vérifier

USR_VERIFY APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0xB0	0x20	0x0	0x0	Length of the PIN data	PIN data	N/A
• CLA byte denotes the structure of the command • INS byte (0x20) indicates a verify instruction • P1 and P2 are not used, and are both set to 0 • The data field contains the PIN value						
Optional data		Status word		Meaning of status word		
N/A		0x9000		Successful processing		
N/A		0x63Cy		Verification failed. y is the number of tries remaining		
N/A		0x6300		PIN is blocked		

- USR_CREDIT permettra de créditer la balance et ne fonctionnera que si le PIN a été vérifié

USR_CREDIT APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0xB0	0x30	0x0	0x0	yy	Credit amount	0x14
The data field contains the credit amount						
Response APDU						
Optional data		Status word		Meaning of status word		
signature		0x9000		Successful processing		
		0x6301		PIN verification required		
		0x6A83		Invalid credit amount		
		0x6A84		Exceed the maximum amount		

- USR_DEBIT permettra de débiter la balance et ne fonctionnera que si le PIN a été vérifié

USR_DEBIT APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0xB0	0x40	0x0	0x0	yy	Debit amount	0x14
The data field contains the debit amount						
Response APDU						
Optional data		Status word		Meaning of status word		
signature		0x9000		Successful processing		
		0x6301		PIN verification required		
		0x6A83		Invalid debit amount		
		0x6A85		Negative balance		

- ADM_INIT permettra d'activer l'applet en générant la paire de clés lors de son premier appel. Elle servira aussi à initialiser le PIN utilisateur (et éventuellement à le débloquer) et à fixer le solde à une valeur si besoin.

ADM_INIT APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0xB0	0x90	0x01	0x0	yy (à voir selon votre format)	balance amount	N/A
		0x02		Length of the PIN data	PIN data	
		0x03		N/A	N/A	
• P1 = 0x03 unblock the USR_PIN						
Response APDU						
Optional data		Status word		Meaning of status word		
		0x9000		Successful processing		
		0x6301		PIN verification required		
		0x6A84		Exceed the maximum amount		

- ADM_GET_PUBLIC_KEY permettra de récupérer la clé publique RSA

USR_GET_PUBLIC_KEY APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0xB0	0x80	0x0	0x0	N/A	N/A	0x40
Response APDU						
Optional data		Status word		Meaning of status word		
public key		0x9000		Successful processing		
		0x6301		PIN verification required		

- ADM_GET_LOG permettra de récupérer une entrée dans le fichier de logs.

ADM_GET_LOG APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0xB0	0x70	Entry index	0x0	N/A	N/A	Log entry length
Response APDU						
Optional data		Status word		Meaning of status word		
Log entry length		0x9000		Successful processing		
		0x6301		PIN verification required		

- ADM_VERIFY permettra de proposer un PIN administrateur à vérifier

ADM_VERIFY APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0xB0	0x60	0x0	0x0	Length of the PIN data	PIN data	N/A
• CLA byte denotes the structure of the command • INS byte (0x20) indicates a verify instruction • P1 and P2 are not used, and are both set to 0 • The data field contains the PIN value						
Optional data		Status word		Meaning of status word		
N/A		0x9000		Successful processing		
N/A		0x6300		Verification failed. PIN is blocked		