Travaux pratiques
JAVA CARD

UE Carte à puce
Master Cryptis

TP n°1 - Porte monnaie électronique de base
TP n°2 - Porte monnaie électronique sécurisé
TP n°3 - Porte monnaie électronique sécurisé sans création de monnaie
TP n°4 - Porte monnaie électronique et fidélité
TP n°5 - Porte monnaie électronique et fidélité sécurisé
TP n°6 – Numéro de séquence unique

Jean-Louis LANET
2/10/2009

TP n°1 - Porte monnaie électronique de base

Objectif : connaissance des outils GemXplore (loader, debugger, script), de la chaîne de compilation et de conversion, écriture d'une applet Java Card, définition de l'AID, utilisation de l'API *javacard.framework.*

Description :
L'application contient un compteur appelé balance qui est incrémenté par un montant avec une commande de crédit et décrémenté par une commande de débit. Il est possible de lire le contenu de la balance. La balance est persistante.

La variable contenant la balance est un 16 bits. La valeur initiale de la balance est zéro. Il n'est pas possible de décrémenter un compteur au dessous de zéro.

Commandes

GET BALANCE
Renvoie la valeur de la balance.

CLA	80h
INS	34h
P1	00h
P2	00h
Le	02h

Réponse :

Data	Valeur de la balance sur 2 octets
SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Le Incorrect, 6D00h, INS incorrect
	9000h commande exécutée normalement

CREDIT
Additionne le crédit à la balance en cours

CLA	80h
INS	36h
P1	00h
P2	00h
Lc	02h
Data	Amount (2 octets)

Réponse :

SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Lc Incorrect, 6D00h, INS incorrect
	9000h commande exécutée normalement

DEBIT
Cette commande décrémente le contenu de la balance par le montant si et seulement si la valeur de la balance est supérieure ou égale au montant.

CLA	80h
INS	38h
P1	00h
P2	00h
Lc	02h
Data	Amount (2 octets)

Réponse :

SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Lc Incorrect, 6D00h, INS incorrect
	6986h, débit impossible
	9000h commande exécutée normalement

Travail demandé :

1. Ecrire le programme Java correspondant à cette spécification. Pour ce faire, créer un projet Java Card (un projet par package). Dans l'étape 2, on choisira comme AID de package **limogestp01**. Les cartes utilisées sont des GemXplore 3G définies dans le wizard par USIMCard R5 dans l'étape 3. Dans ce même wizard, au pas 5 choisir l'option JCardManager comme outil d'exécution et de mise au point de l'application. Une fois ce projet créé, le wizard en lance un second pour la création d'une applet java card. Donner comme AID d'applet **limogestp01purse**. Comme il ne sera pas nécessaire de créer d'instance, l'AID de l'instance sera le même que celui de l'applet. Pas d'option Java Card au pas 4.
2. Implémenter les commandes.
3. Compiler le programme et le convertir. Projet -> exécuter
4. Charger l'application dans la carte. Pour ce faire, on peut soit lancer la suite de commande **Authenticate**, **Install for Load**, **Load**, **Install for Install** ou plus simplement **Quick Load**. Pour vérifier si l'application est bien chargée utiliser la commande **GetStatus**.
5. Tester votre application en envoyant des APDU. Créer un script de test pour valider l'ensemble des fonctions de l'application.

La validation du TP se fera sur l'exécution automatique du script sans erreur pour toutes les fonctionnalités attendues. Comme il n'est pas demandé de créer d'instance d'applet pensez à mettre comme **instance AID l'applet AID**.

TP n°2 - Porte monnaie électronique sécurisé

Objectif : guide de développement de sécurité présenté en cours.

Description :
L'application contient un compteur appelé balance qui est incrémenté par un montant avec une commande de crédit et décrémenté par une commande de débit. Il est possible de lire le contenu de la balance. La balance est persistante. La variable contenant la balance est un 16 bits. La valeur initiale de la balance est zéro. Il n'est pas possible de décrémenter un compteur au dessous de zéro.

La commande de débit est sécurisée par la présentation d'un Pin code afin d'authentifier l'utilisateur avant le retrait. L'authentification est valide pour la session en cours.

Commandes

GET BALANCE
Renvoie la valeur de la balance.

CLA	80h
INS	34h
P1	00h
P2	00h
Le	02h

Réponse :

Data	Valeur de la balance sur 2 octets
SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Le Incorrect, 6D00h, INS incorrect
	9000h commande exécutée normalement

CREDIT
Additionne le crédit à la balance en cours

CLA	80h
INS	36h
P1	00h
P2	00h
Lc	02h
Data	Amount (2 octets)

Réponse :

SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Lc Incorrect, 6D00h, INS incorrect
	9000h commande exécutée normalement

DEBIT
Cette commande décrémente le contenu de la balance par le montant si et seulement si la valeur de la balance est supérieure ou égale au montant. Cette commande ne peut être effectuée qu'après l'authentification de l'utilisateur.

CLA	80h
INS	38h

P1	00h
P2	00h
Lc	02h
Data	Amount (2 octets)

Réponse :

SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Lc Incorrect, 6D00h, INS incorrect
	6986h, débit impossible
	6985h, PIN non vérifié
	9000h commande exécutée normalement

VERIFY PIN

Cette commande autorise l'utilisateur à valider un Pin code avant une demande de débit. Après trois tentatives infructueuses la vérification est bloquée. Le code sera initialisé avec la valeur par défaut 0x99, 0x99, 0xFF et 0xFF. Le Pin reste valide pour la session en cours.

CLA	00h
INS	20h
P1	00h
P2	00h
Lc	04h
Data	PIN (4 octets)

Réponse :

SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Lc Incorrect, 6D00h, INS incorrect
	6985h, PIN blocked
	6A80h, Incorrect PIN
	9000h commande exécutée normalement

Travail demandé :

1. Ecrire le programme Java correspondant à cette spécification. Pour ce faire, créer un projet Java Card (un projet par package). Dans l'étape 2, on choisira comme AID de package **limogestp02**. Les cartes utilisées sont des GemXplore 3G définies dans le wizard par USIMCard R5 dans l'étape 3. Dans ce même wizard, au pas 5 choisir l'option JCardManager comme outil d'exécution et de mise au point de l'application. Une fois ce projet créé, le wizard en lance un second pour la création d'une applet java card. Donner comme AID d'applet **limogestp02purse**. Comme il ne sera pas nécessaire de créer d'instance, l'AID de l'instance sera le même que celui de l'applet. Pas d'option Java Card au pas 4.
2. Implémenter les commandes on pourra judicieusement utiliser la classe OwnerPin..
3. Compiler le programme et le convertir. Projet -> exécuter
4. Charger l'application dans la carte. Pour ce faire, on peut soit lancer la suite de commande **Authenticate**, **Install for Load**, **Load**, **Install for Install** ou plus simplement **Quick Load**. Pour vérifier si l'application est bien chargée utiliser la commande **GetStatus**.
5. Tester votre application en envoyant des APDU. Créer un script de test pour valider l'ensemble des fonctions de l'application.

La validation du TP se fera sur l'exécution automatique du script sans erreur pour toutes les fonctionnalités attendues. Comme il n'est pas demandé de créer d'instance d'applet pensez à mettre comme instance AID l'applet AID.

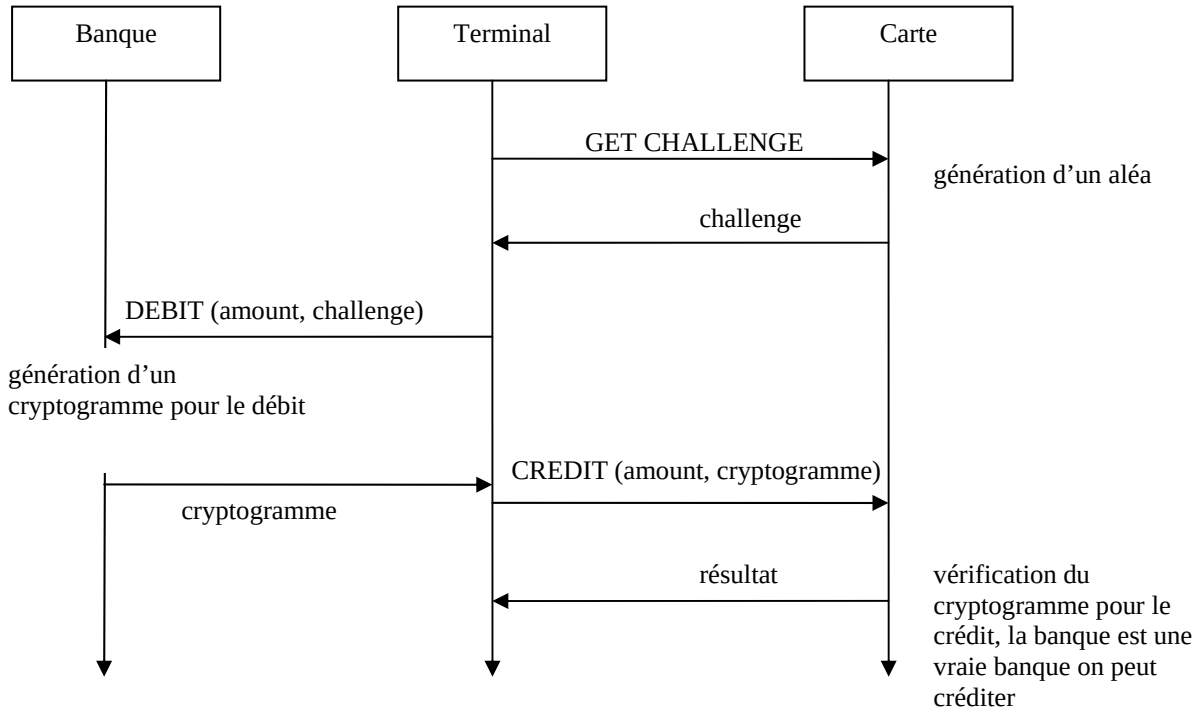
TP n°3 - Porte monnaie électronique sécurisé sans création de monnaie.

Objectif : manipulation des primitives de cryptographie.

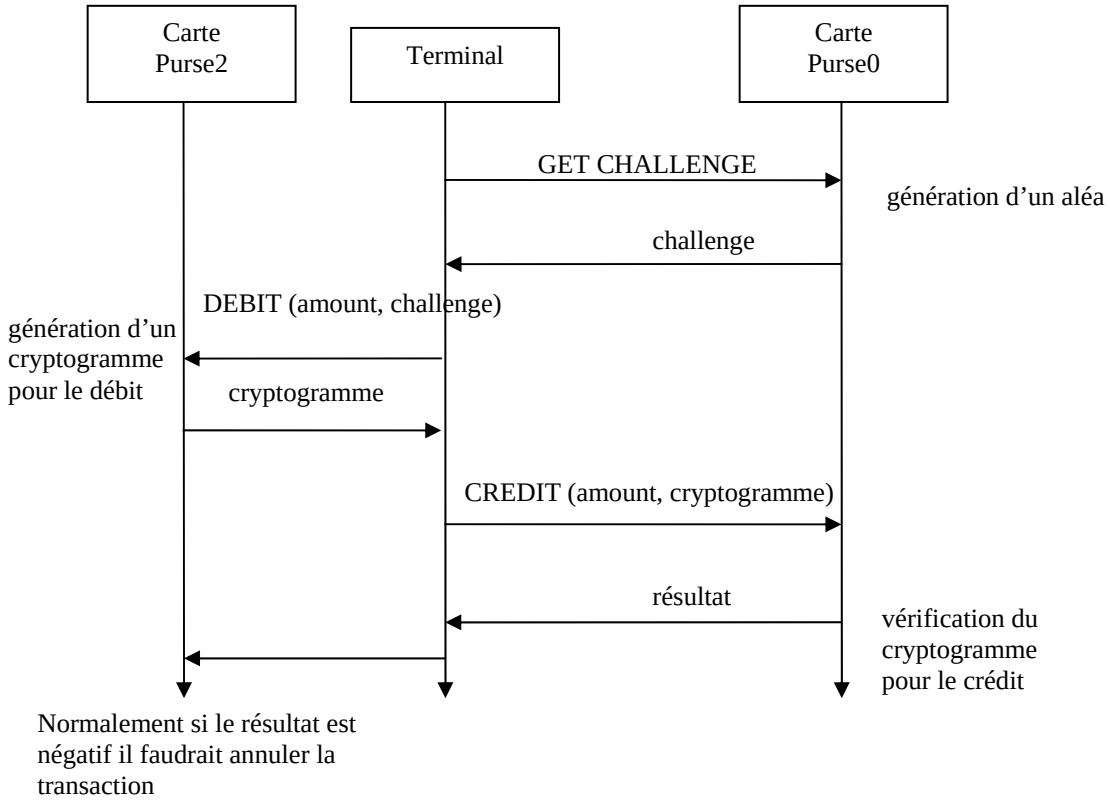
Description :
L'application contient un compteur appelé balance qui est incrémenté par un montant avec une commande de crédit et décrémenté par une commande de débit. Il est possible de lire le contenu de la balance. La balance est persistante. La variable contenant la balance est un 16 bits. La valeur initiale de la balance est 0x4000. Il n'est pas possible de décrémenter un compteur au dessous de zéro.

La commande de débit est sécurisée par la présentation d'un Pin code afin d'authentifier l'utilisateur avant le retrait. L'authentification est valide pour la session en cours.

La commande de crédit est sécurisée par un cryptogramme afin d'éviter la création de monnaie. Ainsi un crédit ne peut se faire que si un débit sur un compte bancaire a été réalisé auparavant. La banque ET la carte partage une clé.



POUR LA SIMPLICITE DU TP, l'architecture proposée ne met en œuvre que la carte (le rôle de la banque est assuré par la carte, dans laquelle deux instances du purse seront utilisées. On transférera l'argent d'un porte monnaie à l'autre produisant un cryptogramme à chaque débit. Le schéma est donc le suivant :



Pour le calcul du cryptogramme, on utilisera un MAC avec triple DES et un padding ISO9797 méthode 1[1] sur les données suivantes :

Amount (2 octets), Challenge (8 octets)

On utilisera un triple DES avec 2 clés de 8 octets soit la clé :
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
Le challenge sera stocké sous forme de transient de type « clear on deselect »

Commandes

GET BALANCE
Renvoie la valeur de la balance.

CLA	80h
INS	34h
P1	00h
P2	00h
Le	02h

Réponse :

Data	Valeur de la balance sur 2 octets
SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Le Incorrect, 6D00h, INS incorrect
	9000h commande exécutée normalement

[1] Confère API *javacard.security* classe signature: ALG_DES_MAC8_ISO9797_M1

CREDIT

Additionne le crédit à la balance en cours. Le cryptogramme est vérifié avant le crédit. Si le cryptogramme ne correspond pas la balance n'est pas créditée.

CLA	80h
INS	36h
P1	00h
P2	00h
Lc	0Ah
Data	Amount (2 octets) Cryptogramme (8 octets)

Réponse :

SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Lc Incorrect, 6D00h, INS incorrect
	6984h, Cryptogramme incorrect
	9000h commande exécutée normalement

DEBIT

Cette commande décrémente le contenu de la balance par le montant si et seulement si la valeur de la balance est supérieure ou égale au montant. Cette commande ne peut être effectuée qu'après l'authentification de l'utilisateur. La commande renvoie le cryptogramme pouvant être utilisé pour créditer du même montant une autre carte (Dans le cadre de ce TP il s'agit de la même carte).

CLA	80h
INS	38h
P1	00h
P2	00h
Lc	0Ah
Data	Amount (2 octets) Challenge (8 octets)

Réponse :

Data	Cryptogramme de débit (8 octets)
SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Lc Incorrect, 6D00h, INS incorrect
	6986h, débit impossible
	6985h, PIN non vérifié
	9000h commande exécutée normalement

VERIFY PIN

Cette commande autorise l'utilisateur à valider un Pin code avant toute demande de débit. Après trois tentatives infructueuses la vérification est bloquée. Le code sera initialisé avec la valeur par défaut 0x99, 0x99, 0xFF et 0xFF.

CLA	00h
INS	20h
P1	00h

P2	00h
Lc	04h
Data	PIN (4 octets)

Réponse :

SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Lc incorrect, 6D00h, INS incorrect
	6985h, PIN bloqué
	6A80h, PIN incorrect
	9000h commande exécutée normalement

GET CHALLENGE

Cette commande renvoie un aléa de 8 octets utilisé pour calculer le cryptogramme de la commande DEBIT ainsi que pour vérifier le cryptogramme de la commande CREDIT.

CLA	00h
INS	84h
P1	00h
P2	00h
Le	08h

Réponse :

Data	Aléa (8 octets)
SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect, 6D00h, INS incorrect
	6700h, Le incorrect
	9000h commande exécutée normalement

Travail demandé :

1. Ecrire le programme Java correspondant à cette spécification. Pour ce faire, créer un projet Java Card (un projet par package). Dans l'étape 2, on choisira comme AID de package **limogestp03**. Les cartes utilisées sont des GemXplore 3G définies dans le wizard par USIMCard R5 dans l'étape 3. Dans ce même wizard, au pas 5 choisir l'option JCardManager comme outil d'exécution et de mise au point de l'application. Une fois ce projet créé, le wizard en lance un second pour la création d'une applet java card. Donner comme AID d'applet **limogestp03purse**. Comme il ne sera pas nécessaire de créer d'instance, l'AID de l'instance sera le même que celui de l'applet. Pas d'option Java Card au pas 4. **Attention**, il faut créer deux instances du même purse et donc on fera attention aux problèmes d'AID, de mémoire transiente de canaux logiques et de multiselection.
2. Implémenter les commandes.
3. Compiler le programme et le convertir.
4. Charger l'application dans la carte. Pour ce faire, on peut soit lancer la suite de commande **Authenticate**, **Install for Load**, **Load**, **Install for Install** ou plus simplement **Quick Load**. Pour vérifier si l'application est bien chargée utiliser la commande **GetStatus**.
5. Tester votre application en envoyant des APDU. Créer un script de test pour valider l'ensemble des fonctions de l'application. **DANS** un premier temps il peut être judicieux de ne pas créer d'instance et de jouer contre un seul porte monnaie implémentant toutes les commandes…

La validation du TP se fera sur l’exécution automatique du script sans erreur pour toutes les fonctionnalités attendues.

TP n°4 - Porte monnaie électronique et fidélité

Objectif : utilisation du mécanisme de partage en Java Card

Description :
A chaque débit d'un montant supérieur à 100 unités de compte, le porte monnaie appelle une méthode grantPoint d'une application de fidélité non située dans son contexte. Cette méthode incrémente un compteur de point à 1% du montant. Un débit de 100 ajoute un point au compteur.

L'application de fidélité possède une seule commande GetPoint qui lit la valeur du compteur.

Commandes du loyalty

GET POINT
Renvoie la valeur du compteur de point de l'application de fidélité.

CLA	80h
INS	35h
P1	00h
P2	00h
Le	02h

Réponse :

Data	Valeur du compteur sur 2 octets
SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Le Incorrect, 6D00h, INS incorrect
	9000h commande exécutée normalement

Commandes du Purse

GET BALANCE
Renvoie la valeur de la balance.

CLA	80h
INS	34h
P1	00h
P2	00h
Le	02h

Réponse :

Data	Valeur de la balance sur 2 octets
SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Le Incorrect, 6D00h, INS incorrect
	9000h commande exécutée normalement

CREDIT
Additionne le crédit à la balance en cours

CLA	80h
INS	36h
P1	00h
P2	00h
Lc	02h
Data	Amount (2 octets)

Réponse :

SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Lc Incorrect, 6D00h, INS incorrect
	9000h commande exécutée normalement

DEBIT

Cette commande décrémente le contenu de la balance par le montant si et seulement si la valeur de la balance est supérieure ou égale au montant.

CLA	80h
INS	38h
P1	00h
P2	00h
Lc	02h
Data	Amount (2 octets)

Réponse :

SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Lc Incorrect, 6D00h, INS incorrect
	6986h, débit impossible, 6F01 purse absent
	9000h commande exécutée normalement

Travail demandé :

1. Ecrire le programme Java correspondant à cette spécification. Pour ce faire, créer un projet Java Card (un projet par package). Dans l'étape 2, on choisira comme AID de package **limogestp04**. Les cartes utilisées sont des GemXplore 3G définies dans le wizard par USIMCard R5 dans l'étape 3. Dans ce même wizard, au pas 5 choisir l'option JCardManager comme outil d'exécution et de mise au point de l'application. Une fois ce projet créé, le wizard en lance un second pour la création d'une applet java card. Donner comme AID d'applet **limogestp04pur**. Comme il ne sera pas nécessaire de créer d'instance, l'AID de l'instance sera le même que celui de l'applet. Pas d'option Java Card au pas 4. Compiler le programme et le convertir. Pour le loyalty, on créera un autre projet Java Card (le package doit être différent), puis on donnera comme AID **limogestp04a** pour la classe implémentant l'interface. **ATTENTION**, même si cela n'est écrit nulle part dans la spécification de SUN, il est impossible d'avoir dans le même package, l'interface **ET** son implémentation. On créera donc un troisième package **limogestp04b** pour l'interface. On fera attention au chemin à importer pour l'export file. Il est fortement conseillé de développer l'interface du loyalty **puis** son implémentation **puis** le purse.
2. Implémenter les commandes.

3. Charger l’application dans la carte. Pour ce faire, on peut soit lancer la suite de commande **Authenticate**, **Install for Load**, **Load**, **Install for Install** ou plus simplement **Quick Load**. Pour vérifier si l’application est bien chargée utiliser la commande **GetStatus**.
4. Tester votre application en envoyant des APDU.
5. Créer un script de test pour valider l’ensemble des fonctions de l’application

La validation du TP se fera sur l’exécution automatique du script sans erreur pour toutes les fonctionnalités attendues. Comme il n’est pas demandé de créer d’instance d’applet pensez à mettre comme instance AID l’applet AID.

TP n°5 - Porte monnaie électronique et fidélité sécurisé

Objectif : Cryptage de donnée, utilisation de l'API cipher

Description :
A chaque débit d'un montant supérieur à 100 unité de compte, le porte monnaie appelle une méthode grandPoint d'une application de fidélité non située dans son contexte. Cette méthode incrémente un compteur de point à 1% du montant. Un débit de 100 ajoute un point au compteur.

L'application de fidélité possède une commande GetPoint qui lit la valeur du compteur et une commande qui renvoie des informations confidentielles Get Info. Cette instruction demande de renvoyer une chaîne de caractère contenant le nom du propriétaire (sur 10 octets) crypté avec une clé. Il faudrait une commande permettant de stocker cette valeur dynamiquement dans la carte on l'implémentera si on a le temps.

On utilisera un triple DES avec 2 clés de 8 octets soit la clé :
20 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F

Les cartes utilisées sont des GemXplore 3G.

Commandes du loyalty

GET INFO
Renvoie le nom du propriétaire du programme de fidélité crypté.

CLA	80h
INS	36h
P1	00h
P2	00h
Le	0Ah

Réponse :

Data	Cryptogramme d'information (10 octets)
SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	9000h commande exécutée normalement

GET POINT
Renvoie la valeur du compteur de fidélité.

CLA	80h
INS	35h
P1	00h
P2	00h
Le	02h

Réponse :

Data	Valeur du compteur sur 2 octets
SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6C02h, Le Incorrect
	9000h commande exécutée normalement

Commandes du Purse

GET BALANCE
Renvoie la valeur de la balance.

CLA	80h
INS	34h
P1	00h
P2	00h
Le	02h

Réponse :

Data	Valeur de la balance sur 2 octets
SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6C02h, Le Incorrect
	9000h commande exécutée normalement

CREDIT
Additionne le crédit à la balance en cours

CLA	80h
INS	36h
P1	00h
P2	00h
Lc	02h
Data	Amount (2 octets)

Réponse :

SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Lc Incorrect
	9000h commande exécutée normalement

DEBIT
Cette commande décrémente le contenu de la balance par le montant si et seulement si la valeur de la balance est supérieure ou égale au montant.

CLA	80h
INS	38h
P1	00h

P2	00h
Lc	02h
Data	Amount (2 octets)

Réponse :

SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	6700h, Lc Incorrect
	6986h, débit impossible
	9000h commande exécutée normalement

Travail demandé :

1. Ecrire le programme Java correspondant à l'applet de fidélité, implémenter la modification de la commande débit du purse.
2. Compiler le programme et le convertir
3. Charger l'application dans la carte en ayant soin de fournir un AID valide.
4. Créer un script de test pour valider l'ensemble des fonctions de l'application.

La validation du TP se fera sur l'exécution automatique du script sans erreur pour toutes les fonctionnalités attendues. Comme il n'est pas demandé de créer d'instance d'applet pensez à mettre comme instance AID l'applet AID.

TP n°6 – Numéro de séquence unique (NSU)

Objectif : Cryptage de données, utilisation de l'API crypto et/ou sécurité, utilisation de plusieurs PIN. Tous ces éléments ont été vus dans le cours et dans les TP précédents.

Description : lors de paiement sur Internet l'usager doit fournir son numéro de carte bancaire lequel peut être réutilisé frauduleusement ultérieurement. Plusieurs organismes bancaires proposent des services basés sur des numéros de carte à usage unique. L'usager peut lui-même demander à sa carte de lui fournir un tel numéro, exécute son paiement et la banque vérifie que le numéro fournit appartient bien à l'utilisateur déclaré et que se numéro n'a jamais été utilisé auparavant. Nous vous proposons de développer une telle application sur une carte à puce. Cette applet génère un numéro de transaction unique basé sur le triple DES pour authentification dans un système. Il faut utiliser les commandes ISO 7816-4 à la fois pour les commandes administratives et les échanges opérationnels. Dans la phase administrative (voir les cas d'utilisation, attention le PUT DATA devra être décomposé) il est possible après présentation du PIN code administrateur d'écrire des objets en mémoire. Ces objets seront transmis dans l'APDU sous la forme TLV comme vu dans le cours en utilisant le paramètre P2. Dans la phase opérationnelle, il est possible d'obtenir de la carte après présentation d'un PIN code utilisateur, un numéro de séquence unique. Ne pas implémenter de *secure messaging*. La phase administrative (la personnalisation) est faite à l'aide de la commande PUT DATA et toutes les données doivent être initialisée à l'aide de cette commande. La vérification des PIN administrateur et utilisateur se fait à l'aide de la commande VERIFY. Les données du générateur de numéro seront impérativement initialisées lors de la phase de personnalisation.

Algorithme pour la génération du numéro de séquence unique

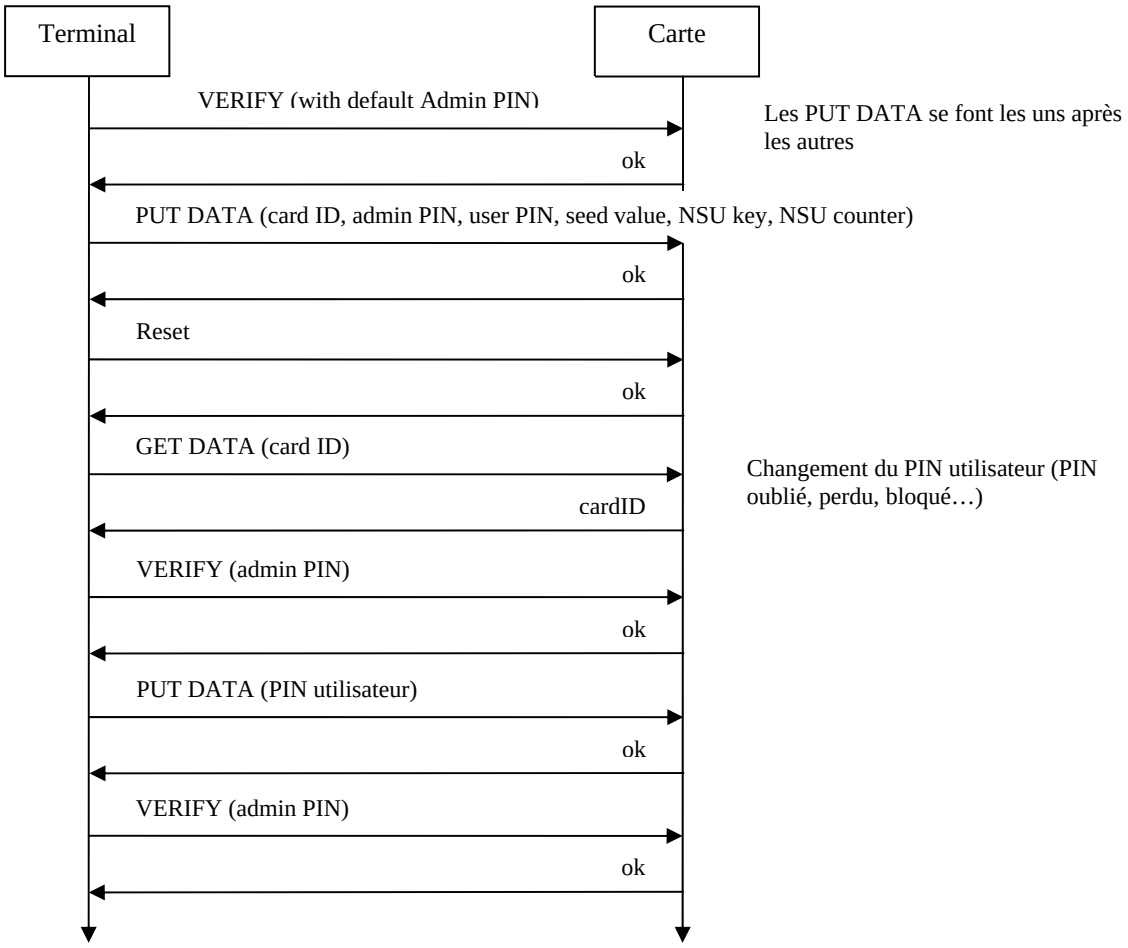
NSUkey: clé privée de la carte de type 3-2 (triple DES à 2 clefs)
C: compteur NSU
S: graine individuelle de la carte
NSUhex := $DES_{(NSUkey)}$ [C || S || padding according to ISO/IEC 9797 Method 1]

|| *dénote la concaténation*

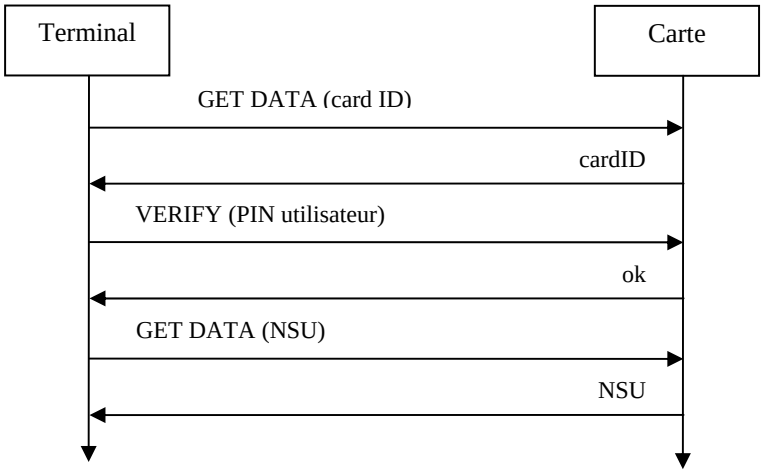
Utilisation de la carte dans un système : durant la phase administrative (chez le banquier ou le fournisseur de carte) toutes les données sont écrites dans la carte ainsi que dans l'applicatif possédant le terminal (par exemple le service bancaire de NSU). En mode opérationnel, l'utilisateur s'identifie auprès de la carte et le terminal récupère à l'aide de la commande GET DATA le numéro de séquence unique pour un paiement sur Internet. La banque reçoit le CardID et peut lui associer, la clé, la graine ainsi que l'état du compteur de NSU. Lorsque la banque reçoit le NSU elle peut ainsi authentifier le NSU sans que l'un des éléments transite en clair sur le réseau. CardID, NSU sont les seuls éléments récupérables en mode utilisateur pour une transaction et l'utilisateur peut vérifier le nombre restant de NSU à utiliser à l'aide du GET DATA. Un PIN utilisateur est nécessaire pour se prémunir de la perte de la carte et de son utilisation frauduleuse, le PIN protège la génération d'un NSU et la valeur du compteur. A chaque génération de NSU le compteur est incrémenté. Attention ce dernier ne peut dépasser strictement une valeur maximale de 1000 unités. La partie applicative n'est pas à implémenter (sauf si vous avez du temps…).

Use Cases

Phase administrative



Phase opérationnelle



A l'issue de cette phase l'utilisateur effectue son paiement en fournissant le n° de carte bancaire : CardID (4 octets), et le NSU (4 octets). Si l'utilisateur souhaite il peut savoir combien il lui reste de NSU en demandant un GET DATA (Nsu counter) et il reçoit le nombre restant de NSU.

6. Commandes du NSU

VERIFY

La longueur du PIN administrateur sera de 4 octets alors que le PIN utilisateur sera de deux octets. Le PIN administrateur sera initialisé à la valeur 12 (soit 0x0C), le PIN utilisateur à 0. Le nombre d'essai sur le PIN administrateur est limité à 2, l'utilisateur à 3.

CLA	00h
INS	20h
P1	00h
P2	01 = user PIN, 02 = admin PIN
Lc	longueur du PIN
data	PIN

Réponse

SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	Lc Incorrect, SW_WRONG_LENGTH
	63Cxh, PIN erroné, avec x le nombre de tentatives restantes.
	Pin Bloqué, SW_CONDITION_NOT_SATIFIED
	9000h commande exécutée normalement

PUT DATA

On utilisera les tags suivants (sous forme de short) pour identifier les objets qui pourront être utilisés dans la commande et initialisés durant la phase d'administration :

cardID 0x50, quatre octets,
userPIN 0x51,
adminPIN 0x52,
seed 0x54, huit octets
nsuKEY 0x55, seize octets
nsuCOUNTER 0x56, deux octets

CLA	00h
INS	DAh
P1	00h
P2	TAG
Lc	Taille des données
data	Donnée avec longueur Lc

Réponse

SW	6E 00h, CLA incorrect
	6A86h, P1
	Lc Incorrect, SW_WRONG_LENGTH
	SW_SECURITY_STATUS_NOT SATISFIED, Pin admin non vérifié
	9000h commande exécutée normalement
	Tag inconnu 0x6A88 équivalent à P2 incorrect

GET DATA

CLA	00h
INS	CAh
P1	00h
P2	TAG
Le	Taille attendu des données

Réponse

Data	Valeur des données
SW	6E 00h, CLA incorrect
	6A86h, P1 ou P2 incorrect
	SW_SECURITY_STATUS_NOT SATISFIED, Pin user non vérifié, Init incomplète
	Le Incorrect, SW_WRONG_LENGTH
	SW_DATA_INVALID, le compteur de NSU est à max
	9000h commande exécutée normalement
	Tag inconnu 0x6A88

Tag de la partie opérationnelle :

cardID	0x50, quatre octets,
userPIN	0x51,
nsuCOUNTER	0x56, deux octets
nsu	0x57, quatre octets

Travail demandé :

1. Ecrire le programme Java correspondant à la spécification du NSU en utilisant l'algorithme décrit, le package AID sera limtp06 et l'applet AID limtp06nsuapp.
2. Compiler le programme et le convertir
3. Charger l'application dans la carte en ayant soin de fournir un AID valide.
4. Créer un script de test (ceci vous simplifiera la vie) pour valider l'ensemble des fonctions de l'application. Ce script testera au moins les scénarios suivants :
    a. tester les cas d'utilisation décrits précédemment,
    b. écrire tous les objets et s'assurer de leur écriture correcte
    c. vérifier que le compteur NSU compte correctement
    d. tester le PIN administrateur correct et incorrect en s'assurant du comportement du compteur de ratification,
    e. s'assurer qu'il n'est possible d'exécuter un PUT DATA que si l'administrateur a présenté son PIN code,
    f. vérifier que la présentation du PIN utilisateur ne confère pas de droit,
    g. garantir que la génération de NSU est bloquée si le compteur de ratification a atteint sa valeur max.

La validation du TP se fera sur l'exécution automatique du script sans erreur pour toutes les fonctionnalités attendues. Comme il n'est pas demandé de créer d'instance d'applet pensez à mettre comme instance AID l'applet AID.

Les règles du jeu :

- Vous pouvez développer sur le simulateur et charger quand vous le souhaitez seulement dans la carte. Attention je juge sur la base de la carte.
- Vous pouvez accéder librement à Internet,
- Tous les documents sont autorisés, rappelez-vous que le meilleur outil reste votre raisonnement qui sera beaucoup plus efficace que Google…
- Vous pouvez échanger par chat avec vos collègues,
- MAIS vous ne vous déplacez pas, ni échangez oralement,
- Le tp sera jugé non seulement sur l'implémentation correcte des fonctionnalités dans une carte, mais aussi sur la qualité du code source (documentation). Pensez à bien noter dans le header de votre fichier le numéro de votre kit ainsi que votre nom.

Vous livrez la carte dans l'état d'installation sans commande administrative exécutée.

Bon courage… et *carpe diem*.

Pensez au header avec votre nom, le n°du kit et tous les commentaires pouvant m'aider lors de la correction. (État du proto, ceci est fait et testé, pas implémenté, etc.).

TP n°7 – Attaques logiques

Objectif : manipulation du CAP file, utilisation du mécanisme de transaction.

Description : ce travail consiste dans un premier temps à écrire une applet qui renvoie au terminal l'adresse physique (ou logique). Pour ce faire, on propose de renvoyer au terminal dans un premier temps l'adresse de l'instance de l'applet. Ecrire donc une applet ayant une méthode d'instance permettant d'avoir dans la pile d'exécution Java cette adresse, puis éditer le fichier CAP à l'aide d'un éditeur hexadécimal (par exemple : http://www.hhdsoftware.com/Family/hex-editor.html), retrouver le composant méthode, la méthode à manipuler et faites les remplacements nécessaires. Renvoyer l'adresse sous forme de short.

Pour vous aider à retrouver les informations dans l'éditeur hexa décimal pensez à l'utilitaire de gemplus CapFileUtility qui donne une vision logique à un CAP file.

La seconde partie consiste à utiliser la faiblesse du mécanisme de transaction et de trans-typer illégalement un objet en tableau et à en lire le contenu. Pour ce faire utiliser l'attaque de Poll *et alii*, et essayer d'afficher sur le terminal le contenu d'un objet. Par exemple afficher l'AID et essayer le modifier.

Rappel :
L'idée consiste à faire une confusion de type entre un tableau et un objet. Attention certaines cartes contiennent des contre mesures (cf. documentation Sun Java Card 3.0). Par exemple la carte que vous utilisez interdit la création du short dans la transaction mais autorise un tableau de transient donc remplacer la ligne 5 par `arrayS = JCSystem.makeTransientShortArray ((short)5, JCSystem.CLEAR_ON_RESET);`

```
1)short[] localArray = null;
2)short[] arrayS;
3)Fake a;
4)JCSystem.beginTransaction();
5)arrayS = new short[5];
6)localArray = arrayS;
7)JCSystem.abortTransaction();
8)a = new Fake();
```

En 1, 2 et 3 on réserve de la mémoire sur la pile java (variable locale) pointant vers différentes structures d'objet. En 4 on débute le mécanisme de transaction, on assigne en 5 une zone mémoire dans le tas, en 6 la variable locale `localArray` pointe sur cette structure. En 7 on annule la transaction et donc on élimine l'objet `arrayS` mais celui ci reste en mémoire. Lorsqu'on créer en 8 un nouvel objet celui pointe sur la dernière zone mémoire libre c'est à dire celle libérée par `arrayS`. Comme en 6 on a maintenu un pointeur `localArray` vers l'objet tableau alors il est possible d'accéder à l'objet `Fake` comme un tableau.

La classe Fake sera définie comme:

```
package mypackage2;

public class Fake {
public short len = (short)0x7FFF;
}
```

Cette classe ne comporte qu'un seul champs d'instance len qui sera confondu lors de l'attaque par le champ length du tableau.

Commandes

GET_ADDRESS
Renvoie l'adresse de l'instance de l'applet.

CLA	80h
INS	50h
P1	00h
P2	00h
Le	02h

Réponse

SW	9000h commande exécutée normalement

READ_DATA
On passe en paramètre la taille des données à lire après la confusion de type entre un tableau et un objet. Répéter l'appel à cette méthode jusqu'à obtenir l'AID de l'instance voulue. Après l'avoir modifié, vérifier en rappelant cette méthode que vous avez réussi à modifier l'AID.

CLA	80h
INS	52h
P1	00h
P2	00h
Le	Taille des données à lire

Réponse données suivies de

SW	9000h commande exécutée normalement

WRITE_DATA
Ecrire au bon emplacement mémoire le nouvel AID...

CLA	80h
INS	52h
P1	00h
P2	00h
Lc	Taille des données à écrire
Data	AID + ???

Réponse données suivies de

SW	9000h commande exécutée normalement