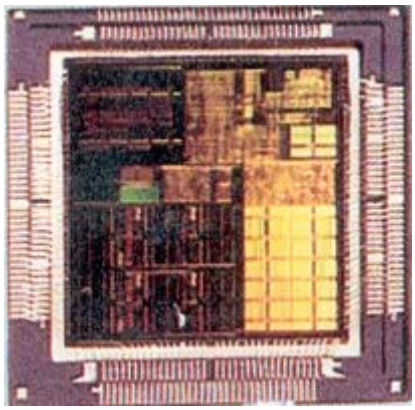
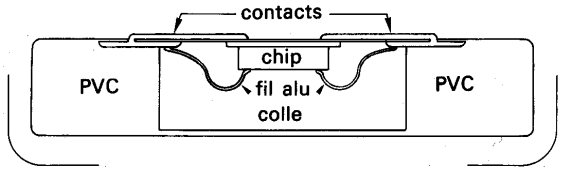
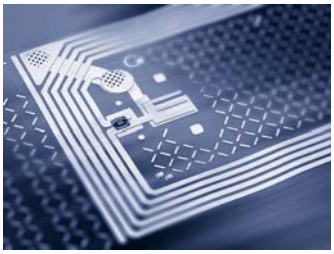
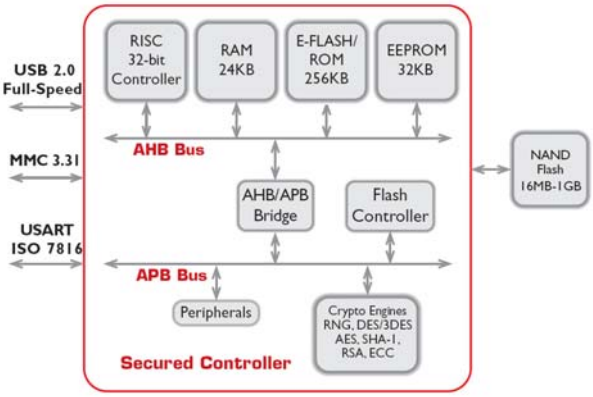


CARTES A PUCE







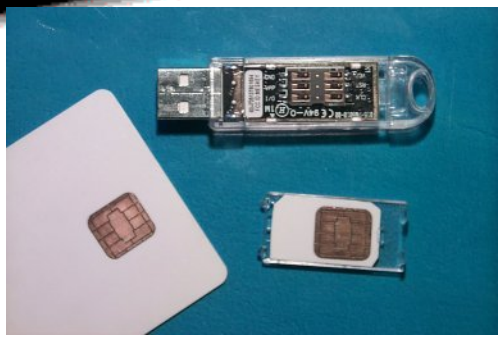


Table des matières

I- Aperçu de la carte à puce.

Historique

C'est en 1950 que la compagnie américaine *Diners'Club* lance la première carte de voyage et de loisirs. Elle se présente sous la forme d'un petit carnet (en carton) qui contient une liste d'adresses (des hôtels restaurants qui acceptent cette carte) et dont la page de garde comporte la signature du titulaire et diverses informations.

L'année 1958 voit la naissance des cartes *American Express* (sur support plastique), émises par les héritiers de la célèbre compagnie de diligences *Wells & Fargo*.

En France le groupe carte Bleue apparaît en 1967 afin d'offrir un moyen de paiement concurrent des cartes américaines. L'objectif est de réduire le nombre de chèques en circulation qui représente déjà 40 % des opérations de paiement et atteindra 90% en 1980.

Les premier distributeurs automatiques de billets (DAB) voient le jour en 1971 et utilisent des cartes bleues munies de pistes magnétiques (une piste magnétiques est constituée de deux zones de lectures d'une capacité de 200 octets).

En 1974 Roland Moreno dépose un premier brevet sur un objet *portable à mémoire*. Il décrit un ensemble (l'ancêtre des cartes à mémoires) constitué d'une mémoire électronique (E^2PROM) collé sur un support (une bague par exemple) et un lecteur réalisant l'alimentation (par couplage électromagnétique) et l'échange de donnés (par liaison optique). Il réalise la démonstration de son système à plusieurs banques. Il fonde la compagnie Innovatron.

Grâce au ministère de l'industrie il est mis en relation avec la compagnie Honeywell Bull qui travaille sur une technologie (TAB Transfert Automatique sur Bande) réalisant le montage de circuits intégrés (puces) sur un ruban de 35 mm, à des fins de test.

En 1977 Michel Ugon (Bull) dépose un premier brevet qui décrit un système à deux puces un microprocesseur et une mémoire programmable. La compagnie BULL CP8 (*Cartes des Années 80*) est crée. La première carte à deux composants est assemblée en 1979. Le Microprocesseur Auto-programmable Monolithique (MAM, en anglais Self Programmable One chip Microprocessor – SPOM)) voit le jour en 1981, c'est en fait le composant qui va équiper toutes les cartes à puces.

Marc Lassus (futur fondateur de Gemplus) supervise la réalisation des premières puces (microprocesseur et mémoire puis du MAM) encartées par CP8 (un nouveau process de fabrication est mis au point pour obtenir des épaisseurs inférieures à un mm).

Le Gie carte à mémoire est crée en 1980 et comprend des industriels (CP8, Schlumberger, Philips), le secrétariat d'Etat des P&T, et plusieurs banques.

En 1982 plusieurs prototypes de publiphones utilisant des cartes à mémoires (les télécartes) sont commandées par la DGT (Délégation Générale des Télécommunications) à plusieurs industriels. Les télécartes vont constituer par la suite un marché très important pour les cartes à puces.

En 1984 la technologie CP8 (MAM) est retenue par les banques françaises, le système d'exploitation B0 va devenir le standard des cartes bancaires française. Le groupement des cartes bancaires (CB) émet une première commande de 12,4 millions de cartes.

Les standards de base des cartes à puces (ISO 7816) sont introduits à partir de 1988.

A partir de 1987, la norme des réseaux mobiles de 2°génération (GSM) introduit la notion de module de sécurité (une carte à puce SIM –Subscriber Identity Module). En raison du succès de la téléphonie mobile, les télécommunications deviennent le premier marché de la carte à puce.

A partir des années 96, l'apparition des cartes java marque l'entrée des systèmes cartes à puce dans le monde des systèmes ouverts. Il devient en effet possible de développer des applications dans un langage largement diffusé.
Depuis 2005, certains composants intègrent des machines virtuelles .NET, ils sont usuellement dénommés *dotnet card.*

Les marchés.

La plus importante entreprise de cartes à puces GEMALTO, détient environ 80% de part de marché et emploie environ 10,000 personnes. En 2006 son chiffre d'affaire était de 1,700 millions d'euros, dont 994 dans le domaine de la téléphonie mobile (carte SIM).

Les tableaux suivants illustrent quelques aspects du marché de la carte à puce.

Cards (Millions of units - Mu)			
Sectors	Memory (MU)	Microprocessor	
			Growth % From 2005
Telecom	440	2150	19%
Financial services / Retail / Loyalty	30	480	20%
Government/ Healthcare	275 (3)	140	55%
Transport	100	30	20%
Pay TV	-	60	-
Corporate Security	10	20	-
Others	10	15	-
Total 2007	865	2895	20 %
Total 2007 Forecast	3760		

Marché de la carte à puce, chiffres 2007, source eurosmart.com

Year	Telecom	Banking	Government healthcare	Total	% Banking	% SIM
1999	200	108		398	27,1	50,3
2000	370	120		551	21,8	67,2
2001	390	140		599	23,4	65,1
2002	430	175		791	22,1	54,4
2003	670	205		979	20,9	68,4
2004	1050	280		1469	19,1	71,5
2005	1220	330		1727	19,1	70,7
2006	2150	480	140	2895	16,6	74,2

Evolution du marché de la carte à puce depuis 1999

En 2002 on estimait que la puissance installée (en MegaDhrystone) du parc informatique était de 4K pour les mainframes, 20K pour les ordinateurs personnels et 34K pour les cartes à puce à microprocesseur.

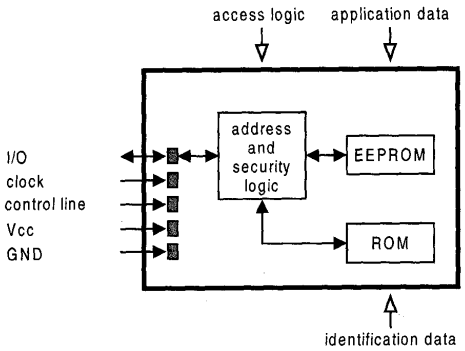
Les principales caractéristiques d'une carte à puce sont les suivantes,

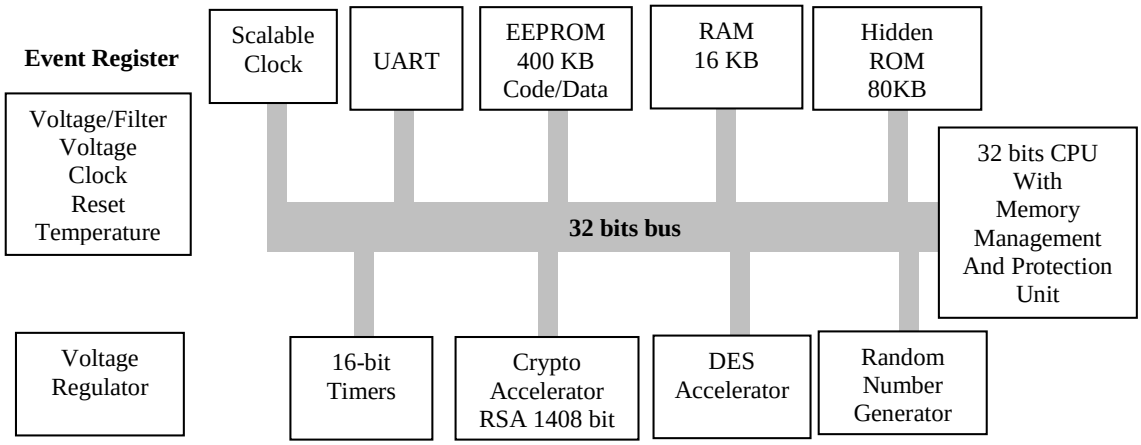
- C'est un objet portable, qui loge des données et des procédures.
- C'est un objet sécurisé
  - Il est difficile de lire les données stockées dans les mémoires de la puce.
  - Le code est exécuté dans un espace de confiance, il n'est pas possible d'obtenir les clés associées à des algorithmes cryptographiques.
- C'est un objet de faible prix (1-5$ pour les SPOM, 0,1-0,5$ pour les cartes magnétiques), mais personnalisable pour des centaines de millions d'exemplaires.
- Une puce ne peut fonctionner seule, elle nécessite un CAD (*Card Acceptance Device*) qui lui délivre de l'énergie, une horloge (base de temps), et un lien de communication. Un CAD usuel est un lecteur de cartes.

La technologie des cartes à puce

Les cartes à mémoire.

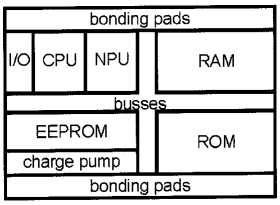
Elles comportent un bloc de sécurité (optionnel) qui contrôle les accès à des mémoires de type ROM ou E^2PROM. Les télécartes de 1ière génération (ou TG1) n'étaient pas sécurisées ; les télécartes de 2ième génération (ou TG2) comportent un bloc de sécurité.



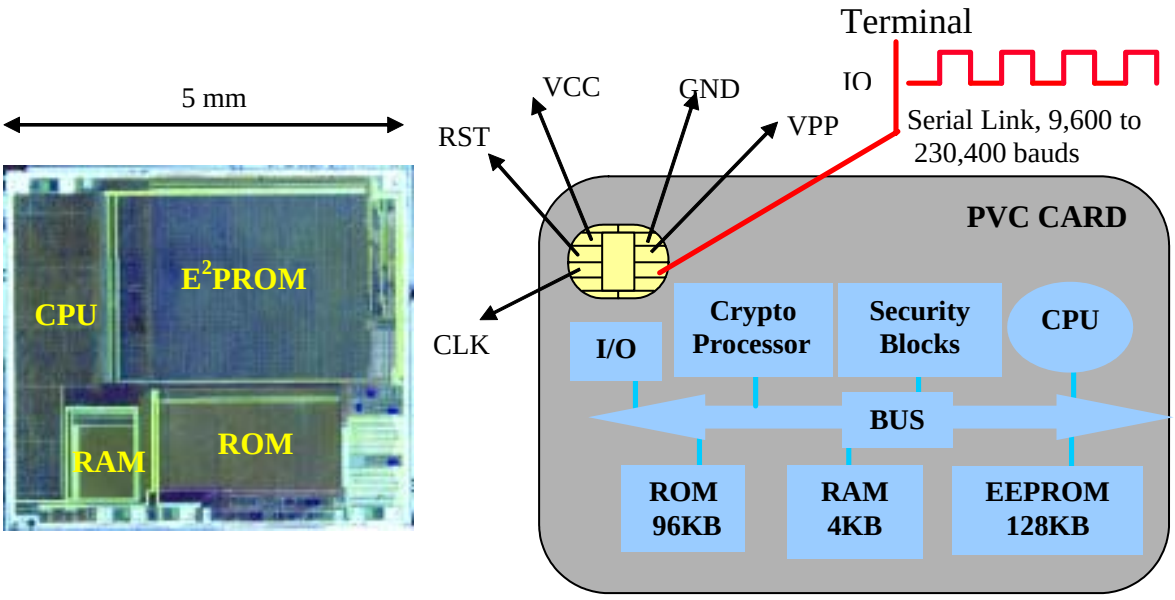


Le microcontrôleur 88CFX4000P

Un microcontrôleur se présente typiquement sous la forme d'un rectangle de silicium dont la surface est inférieure à 25 mm^2. D'une part cette taille est imposée par les contraintes de flexion induite par le support en PVC, et d'autre part cette dimension limitée réalise un compromis entre sécurité physique et complexité du composant.



Les capacités mémoires sont comprises entre 128 et 256 Ko pour la ROM (surface relative 1), 64 et 128 ko pour l'E^{2}PROM (surface relative 4), 4 et 8 Ko pour la RAM (surface relative 16). En raison de ces contraintes technologiques la taille de RAM est modeste; l'E^{2}PROM occupe une portion importante du CHIP. Les écritures en E^{2}PROM sont relativement lentes (de l'ordre de 1 ms par mot mémoire de 32 à 64 octets), et le nombre de ces opérations est limité (de 10^4 à 10^6). L'introduction des mémoires FeRAM devrait amoindrir ces contraintes (10^9 opérations d'écriture, capacités mémoire de l'ordre du Mo, temps d'écriture inférieur à 200ns).



Les classiques processeurs 8 bits ont des puissances de traitements comprises entre 1 et 3 MIPS, ce paramètre est supérieur à 33 MIPS pour les nouvelles architectures à bases de processeurs RISC 32 bits.
En termes de puissance de calcul cryptographique, les systèmes 32 bits réalisent un algorithme DES à un Mbit/s et un calcul RSA 1024 bits en 300mS.
A l'horizon 2004 l'introduction des technologies de type mémoires FLASH devrait conduire à des capacités de l'ordre de 1 Mo. Les puissances de calculs estimées sont de l'ordre de 100 à 200 mips.

Couches de communications ISO 7816.

Requête.	Réponse.
CLA INS P1 P2 Lc [Lc octets]	sw1 sw2
CLA INS P1 P2 Le	[Le octets] sw1 sw2
CLA INS P1 P2 Lc [Lc octets] CLA C0 00 00 Le	61 Le [Le octets] sw1 sw2

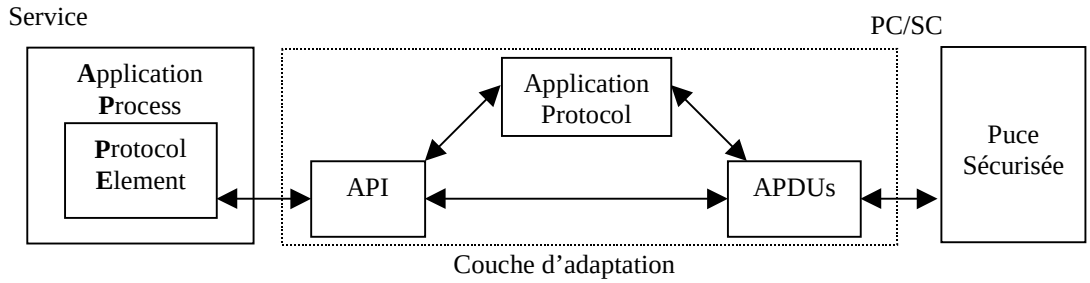
Commandes (APDUs) définis par la norme ISO 7816.

La norme ISO 7816 décrit l'interface de communication entre la puce et son terminal associé. Ce dernier fournit l'alimentation en énergie et une horloge dont la fréquence est typiquement 3.5 Mhz. L'échange de données entre puce et terminal est assuré par une liaison série dont le débit est compris entre 9600 et 230,400 bauds. La norme 7812-12 définir cependant une interfcae USB à 12 Mbit/s. Le terminal produit une requête (APDU) qui comporte conformément au protocole de transport T=0 au moins 5 octets (CLA INS P1 P2 P3) et des octets optionnels (dont la longueur *Lc* est précisée par la valeur de l'octet P3). La carte délivre un message de réponse qui comprend des octets d'information (dont la longueur *Le* est spécifiée par l'octet P3) et un mot de status (sw1 sw2, 9000 notifiant le succès d'une opération) large de deux octets. Lorsque la longueur de la réponse n'est pas connue à priori un mot de status «61 Le» indique la longueur du message de réponse. Une fois ce paramètre connu le terminal obtient l'information au moyen de la commande *GET RESPONSE* (CLA C0 00 00 Le).
Les opérations de lecture et d'écriture, l'invocation des fonctions cryptographiques sont associées à des APDUs spécifiques. L'information stockée dans la puce sécurisée est stockée dans un système de fichiers qui inclut un répertoire racine (MF Master File), des sous répertoires (DF Dedicated File) et des fichiers (EF Elementary File). Chaque composant est identifié par un nombre de **deux octets**; la navigation à travers ce système s'effectue à l'aide d'APDUs particulières (SELECT FILE, READ BINARY, WRITE BINARY). La sécurité est assurée par des protocoles de simple ou mutuelle authentification (transportés par des APDUs), qui en cas de succès autorisent l'accès aux fichiers.
La mise en œuvre d'une carte utilise donc un paradigme d'appel de procédure, transporté par des APDUs (généralement définies pour un système d'exploitation spécifique); l'information embarquée est connue à priori et classée par un système de fichier 7816.

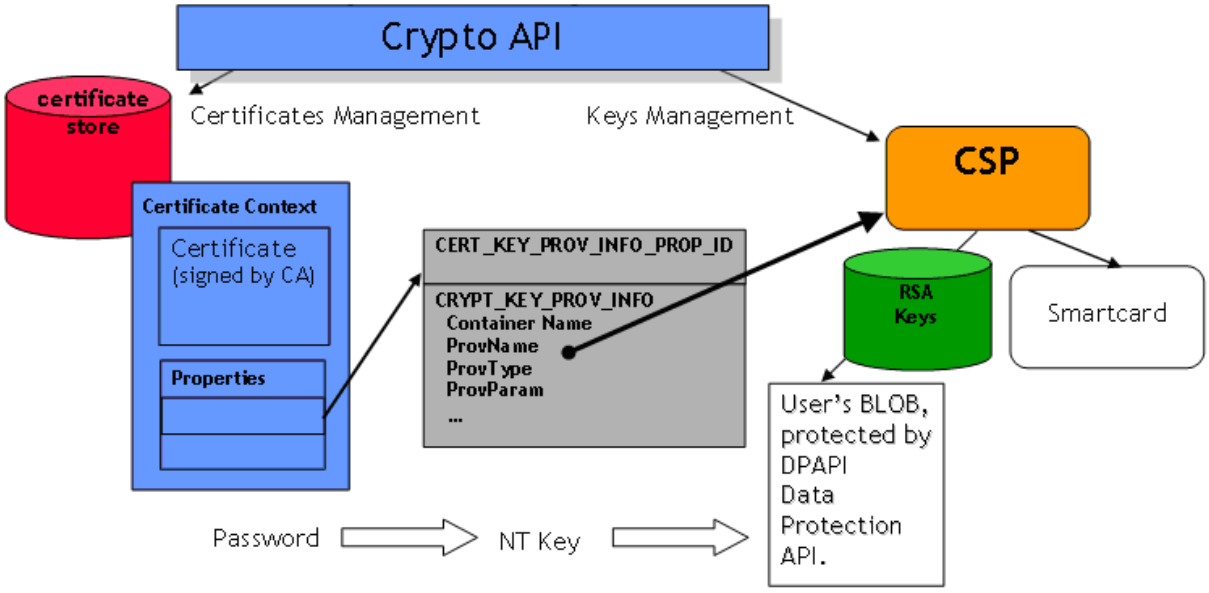
Intégration des cartes à puce aux technologies de l'information.

L'intégration des puces sécurisées aux technologies de l'information implique l'adaptation des logiciels applicatifs de telle sorte qu'ils génèrent les APDUs nécessaires à l'utilisation des ressources embarquées. Schématiquement le middleware classique consiste à définir les éléments protocolaires (PE) requis par un service (*Application Process*) et exécutés dans la puce; chaque élément est associé à une suite d'APDUs (*Application Protocol*) variable selon

le type de carte utilisée. L'application localisée sur le terminal utilise la puce aux moyens d'APIs (Application Programmatic Interface) plus ou moins normalisées (par exemple PC/SC pour les environnements win32), qui offre une interface de niveau APDUs ou plus élevé (PE).



Middleware classique d'une carte à puce.



Crypto Service Provider Microsoft

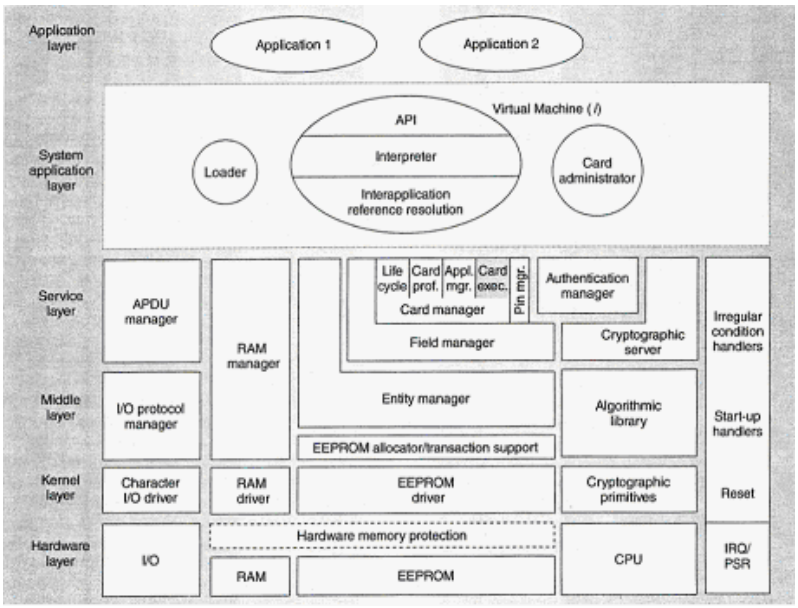
Système d'exploitation d'une carte à puce.

Schématiquement un système d'exploitation d'une carte à puce comporte les éléments suivants,

- Un bloc de gestion des ordres (APDUs) transportés par la liaison série.
- Une bibliothèque de fonction cryptographiques, dont le code est réalisé de telle manière qu'il soit résistant aux attaques logiques connues (Timing attack, DPA, SPA, …).
- Un module de gestion de la RAM
- Un module de gestion de la mémoire non volatile (E^2PROM…), qui stocke les clés des algorithmes cryptographiques et les secrets partagés).
- Un module de gestion de la RAM, qui est une ressource critique en raison de sa faible quantité et de son partage entre procédures et applications.
- Un bloc de gestion d'un système de fichiers localisé dans la mémoire non volatile.
- Un module de gestion des événements indiquant une attaque probable de la puce sécurisée comme par exemple,
  - Une variation anormale de la tension d'alimentation (*glitch*)
  - Une variation anormale de l'horloge externe de la puce sécurisée.
  - Une variation anormale de température.

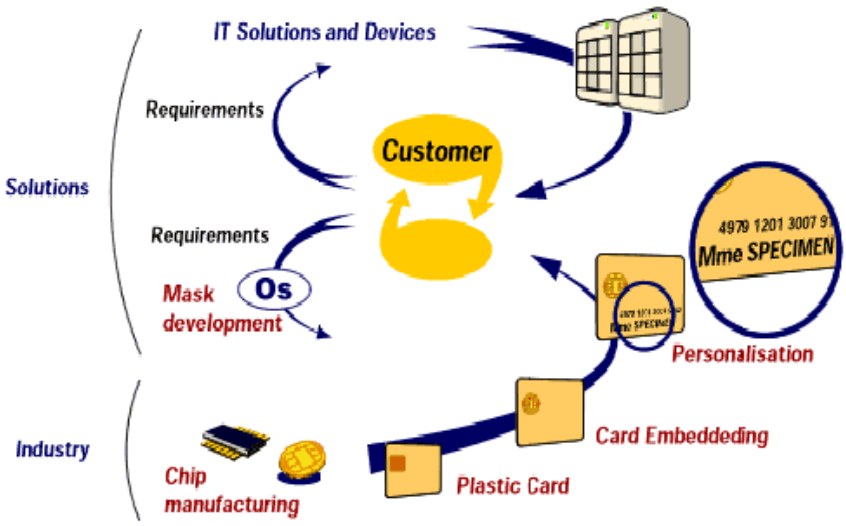
- La détection d'une perte d'intégrité physique du système. La surface d'une puce est généralement recouverte par un treillis métallique qui réalise une sorte de couvercle dont le système teste la présence.

Le système d'exploitation est contenu dans la ROM dont le contenu n'est pas chiffré. La connaissance son code, bien que difficile ne doit pas rendre possible des attaques autorisant la lecture de la mémoire non volatile.



Un exemple de système d'exploitation

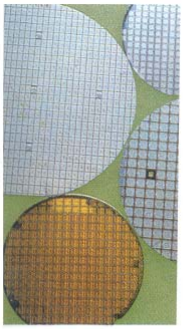
Cycle de vie d'une carte à puce.

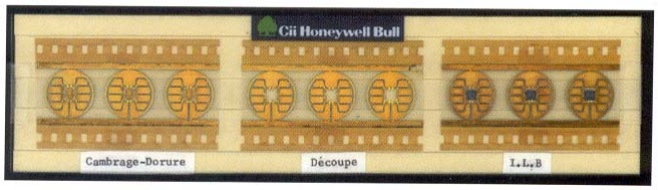


Cycle de vie d'une carte à puce

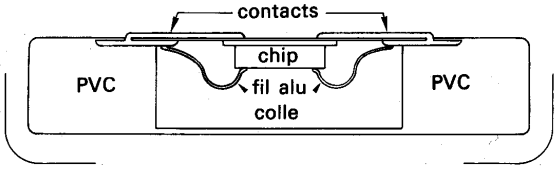
Un système d'exploitation est réalisé par une entreprise spécialisée. Ce logiciel étant ajustée pour un composant électronique particulier, il est appelé ***masque***.

Le masque est stocké dans la ROM du composant lors du processus de **fabrication**. Au terme de cette phase le fondeur de silicium écrit dans la puce une clé dite clé de fabrication et écrit dans la mémoire de cette dernière des informations telles que numéro de série du produit, date de fabrication etc.





Le wafer (plaque de silicium circulaire qui comporte un ensemble de puces) est alors envoyé à l'**encarteur** qui réalise sa découpe, colle les puces sur des micromodules et en réalise le micro câblage. L'ensemble est alors protégé par une substance isolante. Il est ensuite collé sur un support en plastique (PVC) dans lequel on a préalablement usiné une cavité (le *bouton*).



L'encarteur, qui connaît les clés de fabrication, inscrit de nouvelles informations dans la puce et active un verrou de fabrication qui annule la clé de fabrication. Une nouvelle clé est inscrite dans la puce permettant de contrôler les opérations ultérieures. Cette opération est encore dénommée **personnalisation**.
Les cartes sont par la suite transférées vers **l'émetteur** de la carte qui peut inscrire de nouvelles informations.
La vie de vie d'une carte consiste à poser à verrou d'invalidation (IV) qui rend non fonctionnel le système d'exploitation.

Systèmes fermés et systèmes ouverts.

On peut distinguer deux types de systèmes d'exploitation de cartes à puces

- Les systèmes fermés, généralement mono application, dédiés à un usage unique par exemple cartes bancaires (masque CP8 M4 B0'), les cartes santé (VITALE), les cartes pour la téléphonie mobile (modules SIM).
- Les systèmes ouverts, tels que les javacard par exemple, qui ne sont pas destinés à une application particulière, et pour lesquels il est possible de ***charger*** des logiciels (applets) après la réalisation du masque et l'encartage.

Quelques exemples de systèmes fermés.

La carte bancaire BO'

Les cartes bancaires sont dérivées du masque CP8 M4, conçu date du milieu des années 80. La mémoire E^2PROM est associée à des adresses comprises entre 0200h et 09F3 (les adresses référencent des ½ octets) soit environ 2 kilo octets. Elle se divise en sept zones,

- La zone secrète qui stocke les clés émetteurs primaires & secondaires, un jeu de clés secrètes, les codes PIN (Personal Identity Number) du porteur.
- La zone d'accès qui mémorise le nombre de présentations erronées de clés.
- La zone confidentielle, dont le contenu est défini en phase de personnalisation.
- La zone de transaction, qui mémorise les opérations les plus récentes.
- La zone de lecture, qui loge des données en accès libre.

- La zone de fabrication, qui réalise la description de la carte et comporte des informations sur sa fabrication.
- La zone des verrous, qui mémorise l'état de la carte (en fabrication, en service, annulée).

Les clés associées aux opérations de lecture ou d'écriture sont déterminées par le système d'exploitation.

Les cartes TB.

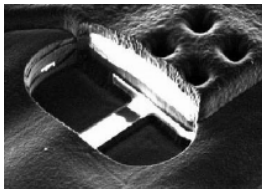
Cette carte est une carte dite d'usage général (*general purpose*) commercialisée dans le courant des années 90 par la société CP8. Elle intègrait des algorithmes cryptographiques DES, RSA ainsi que des mécanismes d'authentification par PIN code et blocage après trois présentations infructueuses. Son principe de fonctionnement est basé sur le contrôle des accès de fichiers élémentaires (lecture / écriture) à l'aide de mécanismes d'opération de présentation de clés. Il existe deux types d'opérations d'authentification,

- Authentification par PIN code, avec blocage du répertoire (après trois échecs).
- Authentification par clé cryptographique. Un premier ordre demande au système d'exploitation de produire un nombre aléatoire. Un deuxième ordre présente au système la valeur chiffrée de la valeur précédemment fournie.
- Chaque répertoire dédié (DF) comporte trois types de fichiers
  - Des fichiers secrets, qui abritent les clés.
  - Des fichiers de contrôle d'accès, qui mémorisent le nombre d'échecs des opérations d'authentification.
  - Des fichiers dont les accès sont plus ou moins conditionnés à la présentation de clés.
- Il existe divers types de clés, associés à plusieurs algorithmes cryptographiques.
  - Clé de fabrication.
  - Clé de personnalisation.
  - Clé d'émetteur.
  - PIN codes
  - Clé d'authentification.
- Un répertoire est un bloc mémoire de taille fixe qui possède un en tête des fichiers et des sous répertoires.
  - Les opérations de création de fichiers et de sous répertoires peuvent impliquer la présentation de clés.
  - Les clés d'authentification sont définies lors de la création des fichiers ou sous répertoires.

Quelques attaques contre les cartes à puces.

Attaques matérielles (intrusives).

Pose de microsondes à la surface du circuit. L'attaquant désire obtenir les secrets de la mémoire non volatile, par exemple en l'isolant du reste de la puce sécurisé et en produisant les signaux électriques nécessaires à sa lecture.



Réactivation du mode test, via un plot de connexion, dans le but de lire la mémoire. En phase de fabrication une procédure de test, réalisé par le système d'exploitation permet de vérifier le bon fonctionnement du système et d'éliminer les composants défectueux. Un fusible désactive ce mode. L'attaquant essaye de rétablir cette connexion.

Reverse engineering, reconstruction du *layout* de la puce, visualisation du code ROM.

Injection de fautes; grâce à des interactions physiques (injection de lumière, etc.) on perturbe le fonctionnement normal du microcontrôleur, afin de produire des erreurs de calculs permettant de déduire la clé d'un algorithme cryptographique.

Attaques logiques (non intrusives).

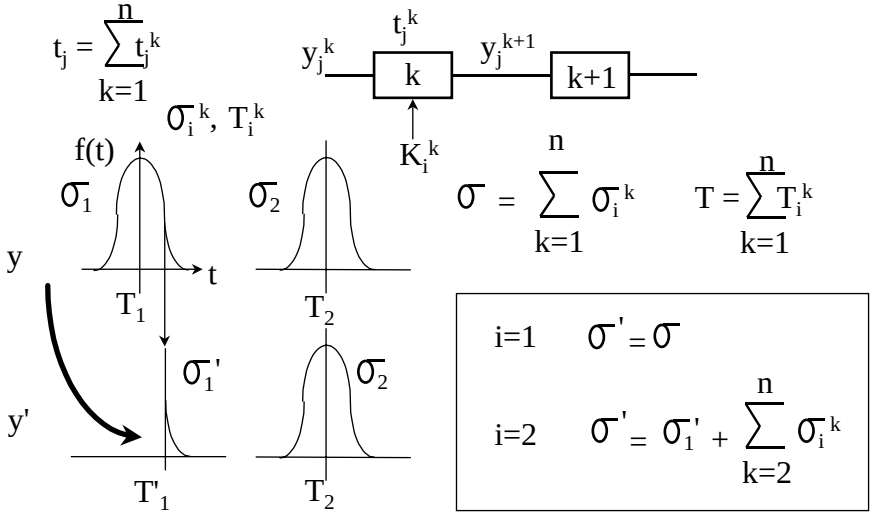
Attaques temporelles (moyenne, écart type). Certaines réalisations logicielles d'algorithmes peuvent présenter des temps de calculs différents en fonction des valeurs calculées et de la clé utilisée.

Attaques par corrélation statistique, telles que *Simple Power Attack* (SPA) ou *Differential Power Analysis* (DPA). Un processeur réalise un algorithme à l'aide d'une suite d'instructions nécessairement différentes en fonction de la clé.

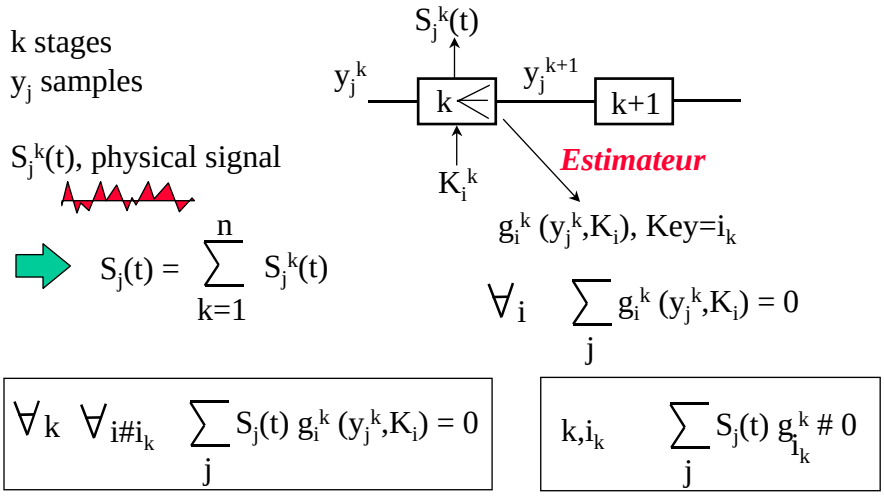
Ainsi un algorithme utilisant une clé parmi $n=2^p$ possible, utilise p instructions différentes pour une clé particulière. Il produit donc des signaux électriques (par exemple puissance consommée) ou radioélectriques qui sont corrélés à la clé opérationnelle.

Défauts de conception.

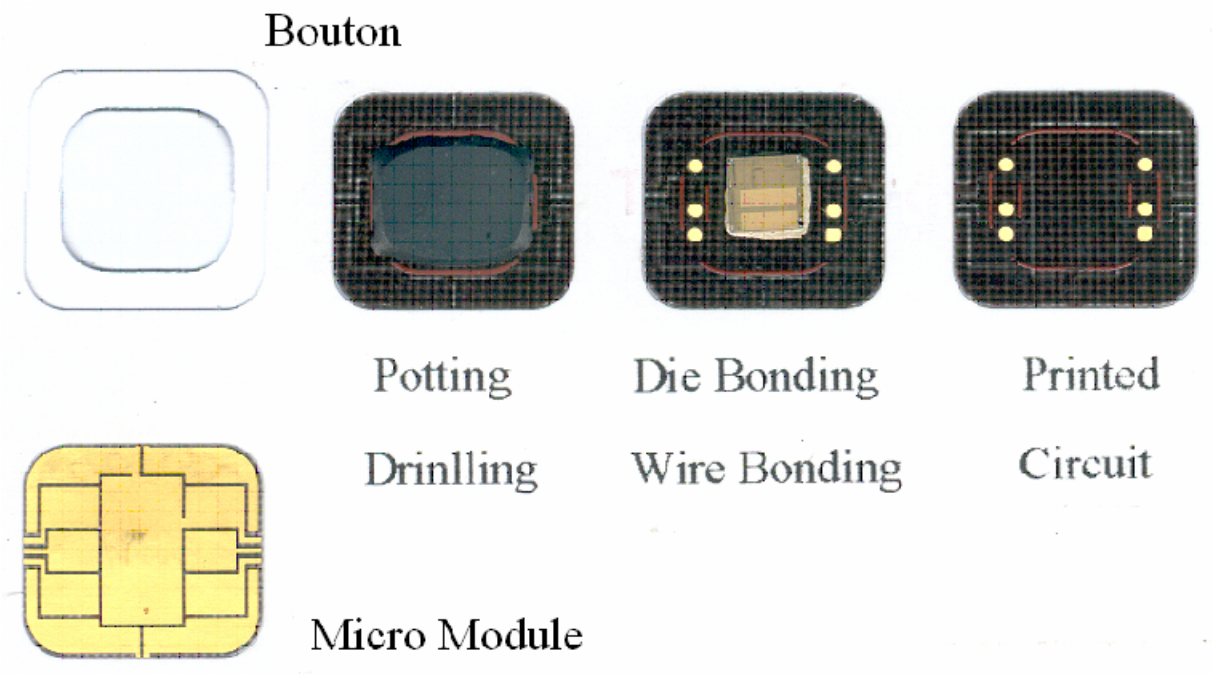
Coupure d'alimentation intempestive, parasite d'horloge, remise à zéro abusive, attaque par éclairement. L'attaquant cherche à créer un défaut dans le déroulement du programme. Il espère par exemple exécuter le code permettant de lire le contenu de la mémoire non volatile E^2PROM.



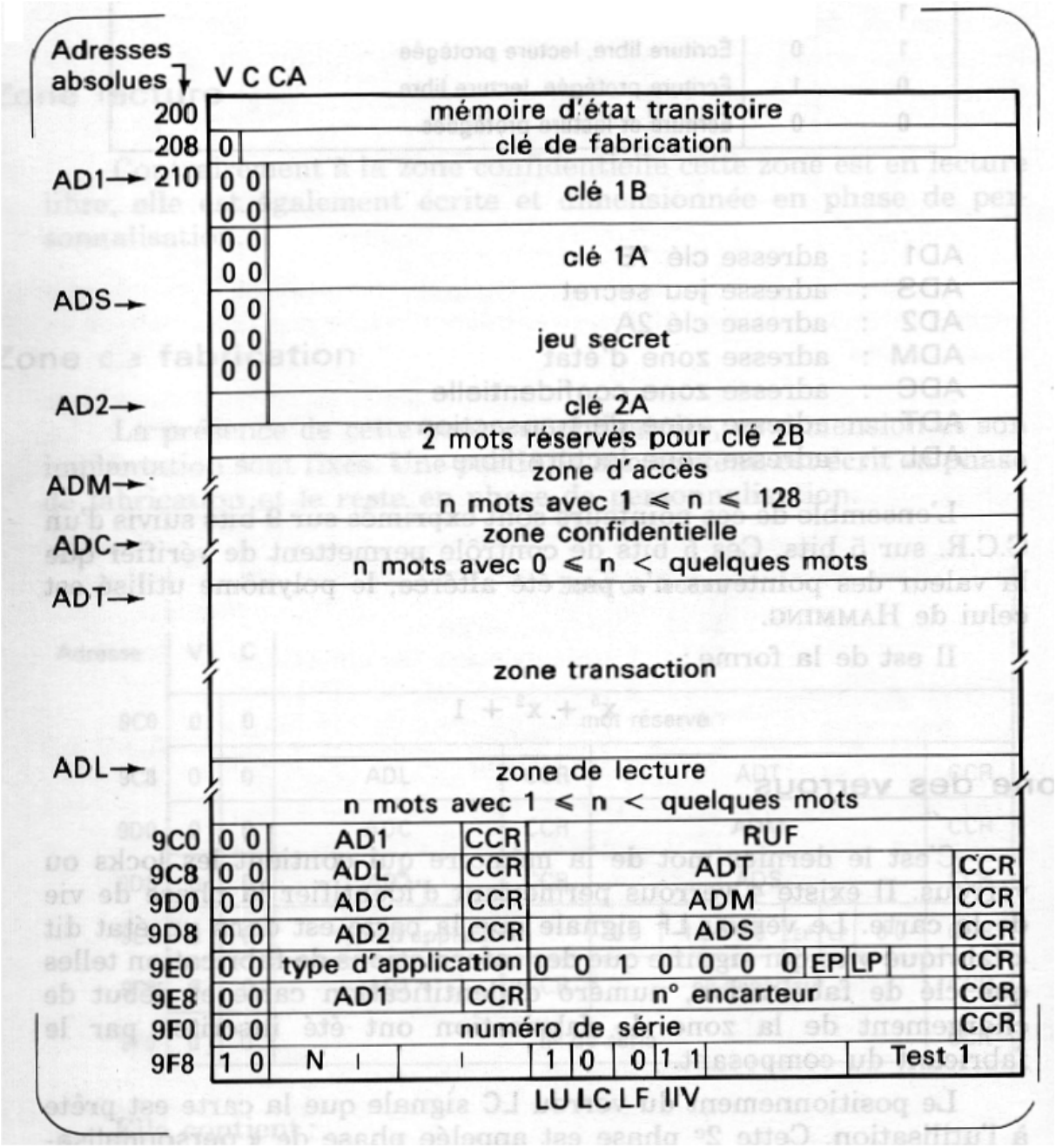
Attaque par écart type.



Attaque par corrélation.



Etapes de fabrication d'une carte à puce



Organisation de la mémoire d'une carte bancaire B0'

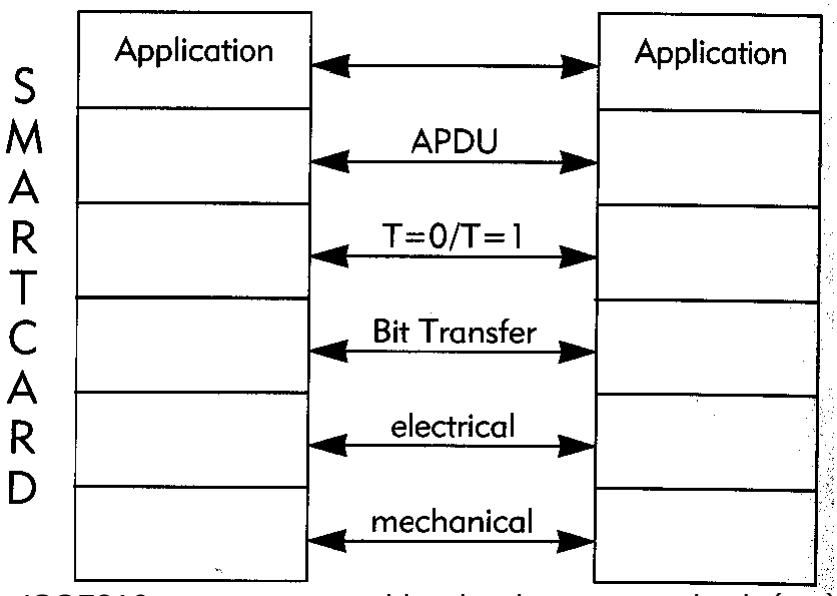
II Normes

La technologie à base de carte à puce est normalisée par plusieurs comités, mais également par divers forums industriels.

Name / Acronym	Content and scope	Headed by	Focus on
Base Standards			
ISO/IEC 7816	Base standard for smart cards	ISO/IEC	Contact cards
ISO/IEC 14443 and 15693	Base standard for contactless cards	ISO/IEC	Cards
Domain – independent standards			
OpenCard Framework	Java framework and API's for terminal – resident smart card applications	OpenCard Consortium	Terminals
PC/SC	API's for smart card applications running in Windows environment	Microsoft	Terminals
Javacard	Definitions for smart cards capable of executing Java programs	Sun	Cards
Smart Card for Windows	Specifications and development environment for seemless integration of smart cards into Windows	Microsoft	Cards
PKCS #15	Specification for storage of of cryptographic tokens	RSA	Cards
MULTOS	Operating System for multi-application cards	MAOSCO	Cards
Finance			
EMV '96	Specifications for multi-application smart cards and payment terminals	Europay Mastercard, Visa	Cards and Terminals
Visa OP	Visa Open Platform Specifications	Visa	Cards and Terminals
CEPS	Electronic Purse Specifications	Europay	Cards and Terminals
EN 1546	Electronic Purse Specifications	CEN	Cards
Telecommunication			
GSM 11.11 and 11.14	Mobile subscriber identity module (SIM) and application toolkit	ETSI	Cards
EN 726	Telecommunication cards	CEN	Cards and Terminals

Un aperçu rapide des standards

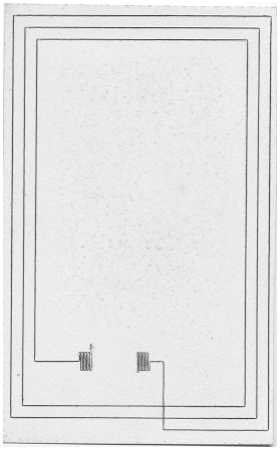
ISO/IEC 7816.



La norme ISO7816 est un ensemble de documents destinés à assurer l'inter opérabilité entre cartes à puce et lecteurs. Elle comporte les parties suivantes:

- 1 Caractéristiques physiques.
- 2 Dimensions et positions des contacts.
- 3 Signaux électriques et protocoles de transmission.
- 4 Commandes intersectorielles pour les échanges.
- 5 Système de numérotation et procédure d'enregistrement pour les identificateurs d'applications.
- 6 Eléments de données interindustrielles.
- 7 Commandes intersectorielles pour langage d'interrogation de carte structurée (SCQL).
- 8 Commandes intersectorielles de sécurité
- 9 Commandes intersectorielles additionnelles et attributs de sécurité.
- 10 Réponse à la RAZ des cartes synchrones.
- 11 Architecture de sécurité.

ISO/IEC 14443 (http://wg8.de)



Cette série de standards décrit des cartes à puce dites sans contacts qui sont alimentées et qui communiquent grâce à des couplages électromagnétiques. Schématiquement il existe deux types principaux de dispositif, les types A d'origine Philire (cartes mifare) et les type B. Les intensités du champ magnétiques sont comprises entre 1,5 et 7,5 A/m, la distance de fonctionnement entre lecteur et carte varie de 0 à 10 cm, la fréquence utilisée est de 13,56 Mhz. Le débit binaire des données est de 106 kbps (13.56 Mhz/128), la fréquence porteuse des données (sub-carrier) est de 847 Khz (13.56Mhz/16).

- 1 Physical characteristics
- 2 Radio Frequency power and signal interface.
- 3 Initialization and anti-collision.
- 4 Transmission protocol.

ISO/IEC 15693 – Vicinity cards (VICCs)

Ce standard définit les tags ou encore étiquettes électroniques (RFIDs à 13.56 Mhz). Les distances de fonctionnement sont de l'ordre du mètre.

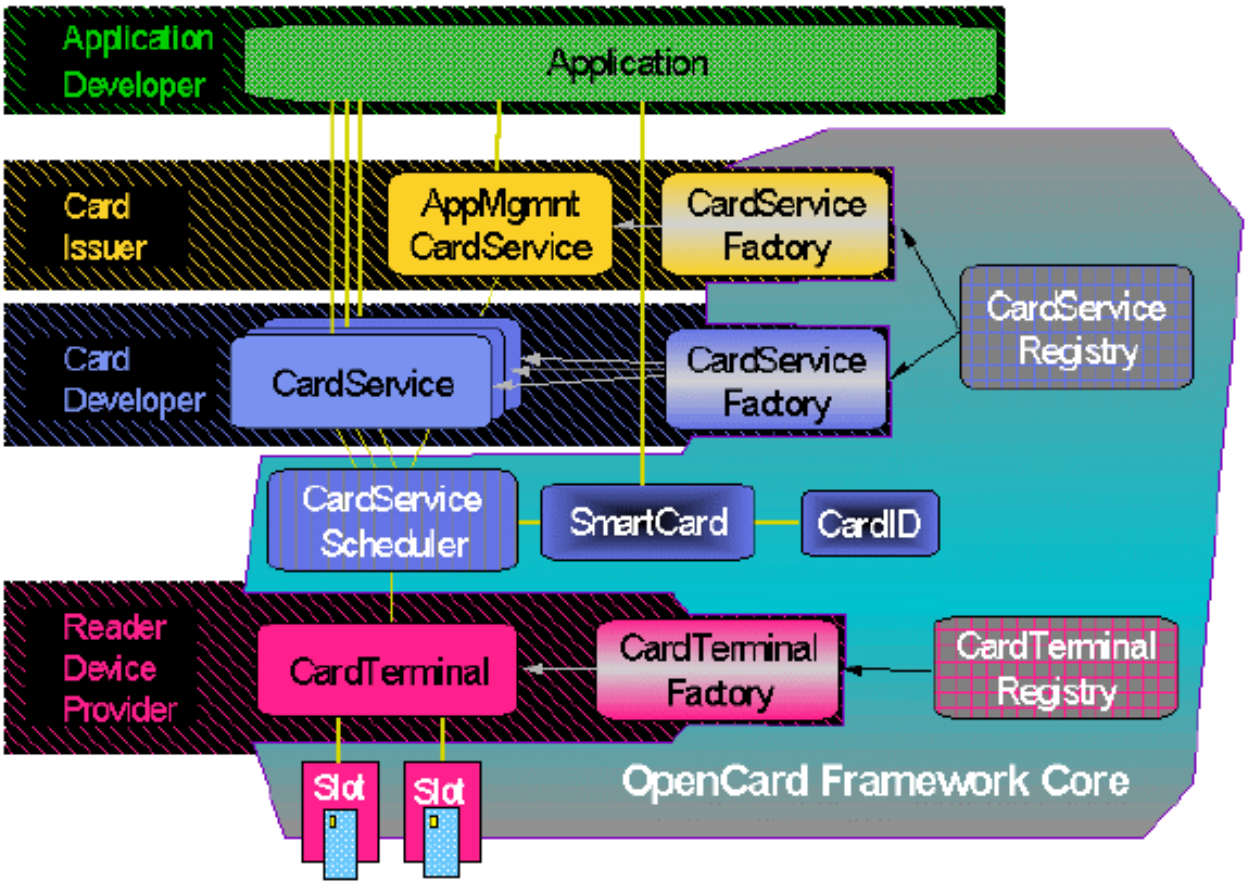
- 1 Physical characteristics
- 2 Air Interface and initialization
- 3 Protocols
- 4 Registration of applications issuers.

JavaCard.

Les spécifications de la JavaCard sont la propriété de la compagnie SUN, et sont élaborées par le JavaCard forum. La version actuelle est JC 2.2.

OpenCard.

Le consortium OpenCard a défini un ensemble d'API java et une architecture côté terminal, nommé Open Card Framework (OCF). L'interface carte terminal est réalisée par des objets java et non par l'interaction avec des protocoles de communication.

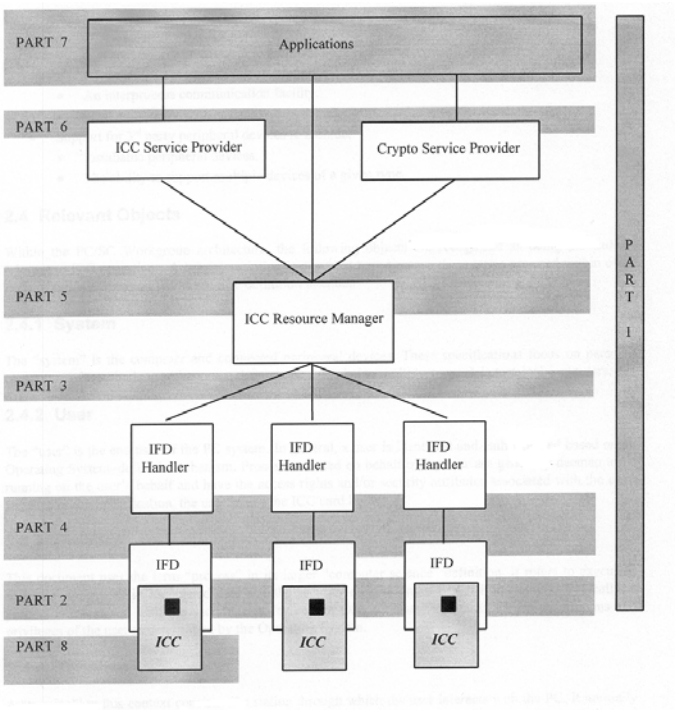


PC/SC – Interoperability Specifications for Integrated Circuit Cards (ICCs) and Personal Computer Systems.

PC/SC permet d'intégrer des lecteurs de carte à puces aux PCs (Interface Device – IFD). La version actuelle est 1.0. Le concept de cette pile consiste à offrir une interface de type API (niveau 6) aux applications qui utilisent les ressources des cartes aux moyens de DLLs.

- 1 Introduction and Overview
- 2 Interface Requirements for compatible IC Cards and Readers.

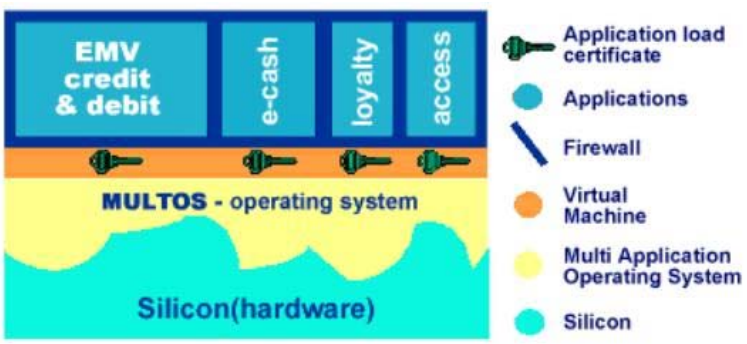
- 3 Requirement for PC-Connected Interface Devices
- 4 IFD Design considerations and Reference Design Information
- 5 ICC Resource Manager Definition. C'est la définition des API cartes.
- 6 ICC Service Provider Interface Definition.
- 7 Application Domain and Developer Design Considerations.
- 8 Recommendations for ICC Security and Privacy Devices.



Microsoft Smartcard for Windows (Windows SC)

En 1999 Microsoft a introduit la spécification d'une carte associée à un kit de développement. Cette carte réalise l'émulation d'un système de fichier (FAT) muni d'un *ACL (Access Control List*). Un ou plusieurs utilisateur utilisent les ressources de la carte après une phase d'authentification. Le développement d'application utilise le langage visual basic dont les *code byte* sont chargés et exécutés par le système d'exploitation de la carte. Il est possible d'utiliser cette dernière à travers des APDUs classique ou des APIs spécifiques.

Multos.



Multos est un système d'exploitation, défini par un consortium industriel (1998) destiné aux cartes multiplicatives. Cette initiative permet de manière analogue à Java Card ou Windows SC de charger des applications (écrites en C) après l'émission d'une carte.

PKCS #15

Ce standard définit le format des fichiers et des répertoires abritant des certificats et des clés cryptographiques.

EMV'96, EMV'2000 et les suivants.

Ce standard publié en 1996 par Europay, Mastercard et Visa, est relatif au domaine bancaire. Il décrit les terminaux, les cartes et leur interaction.
EMV'96 Specification for Payment Systems comporte trois parties,

- 1 caractéristiques mécaniques, électriques, interface logique et protocoles de transport des cartes (ICC).
- 2 Jeu de commandes APDUs
- 3 Mécanismes de sélection des applications.
- 4 Sécurité, méthodes d'authentification et de signature.

Global Platform

Cet ensemble de spécifications est d'origine VISA, il est destiné à la gestion des applications et concerne les cartes et terminaux. Les APIs associées sont basées sur la technologie Java.
Côté carte, trois types de composants sont définis.

- Card Manager. Ce module contrôle la carte et son contenu.
- Gloabl Platform APIs. GP
- Provider Security Domains. Gestion de la sécurité d'une carte multi application.

Les porte monnaie électroniques - PME

Cartes PME.

Il existe trois implémentations cartes de PME, Proton (CEPS+Java), Mondex (Multos), VisaCash (compatible EMV).

CEN Purse EN 1546 (Centre Européen pour la Normalisation)

Description générale des transactions de type PME.

CEPS – Common Electronic Purse Specifcation.

Ce standard est définit par un groupe d'institutions financières européennes et la compagnie VISA. Il est basé sur la spécification EMV, et se décline en trois parties (les spécifications sont disponibles à www.europay.com)

- Business requirements
- Functional requirements
- Technical specifications

Autres spécifications PME

- German Zentraler KreditausschuB (ZKA)
- Carte bancaires (CB).
- Moneo

Standards pour les télécommunications.

GSM 11.11

Cette norme éditée par l'ETSI définit les cartes SIM (Subscriber Identity Module) utilisée par les téléphones mobiles de 2° génération.
Pour l'essentiel un module SIM comporte un algorithme d'authentification et sa clé secrète, un IMSI (identifiant d'un abonné) et deux cartes d'adresses (numéros de services et numéros personnels).

GSM 11.14

Cette norme est une extension de la spécification GSM 11.11. Une carte SIM Tool Kit peut contrôler un téléphone mobile c'est à dire accéder a son clavier et à son écran. Elle peut également envoyer et recevoir des messages SMS.

GSM 3.48

Mécanismes de sécurité dédiés à STK

GSM 3.19

API java pour les cartes SIM.

CEN 726 – Requirements for IC cards and terminals for telecommunicaions use.

Ce standard définit les cartes et terminaux utilisés dans les Publiphones, il est également connu sous l'appellation E9. Il comporte 7 sous ensembles

- 1 System Overview
- 2 Security Framework
- 3 Application independent card requirements
- 4 Application independent card terminal
- 5 Payment Methods
- 6 Telecommunication features
- 7 Security module.

Standards ICAO

L'ICAO ou *International Civil Aviation Organization* publie des standards relatifs aux controles d'identités, plus particulièrement appliqués aux passports électroniques.

III Points clés de la norme 7816.

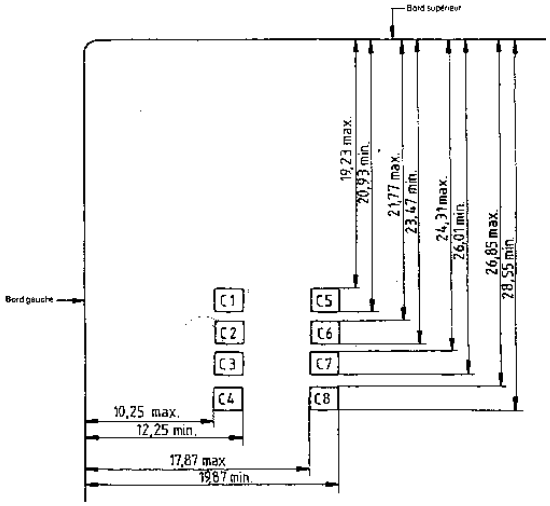
Dimension 7816-1

Le format dit ID1 s'applique à cartes comportant des pistes magnétiques et des caractères en relief.

- Largeur 85,60 mm
- Hauteur 53,98 mm
- Epaisseur 0,76 mm
- Epaisseur maximale des contacts 0.1 mm (0.05 mm pour les cartes EMV).

Contacts 7816-2

La norme définit 8 contacts numérotés de 1 à 8, les niveaux électriques sont compatibles TTL.



- C1, alimentation Vcc, 3v (Classe B) ou 5 V (Classe A)
- C2, RST remise à zéro de la carte Actif à l'état bas.
- C3, CLK l'horloge, niveau bas 0v, niveau haut 0/Vcc.
- C5, GND la masse
- C6, VPP tension de programmation des cartes à EPROM, obsolète
- C7, I/O patte d'entée sortie, 0/Vcc
- C8, non utilisé.

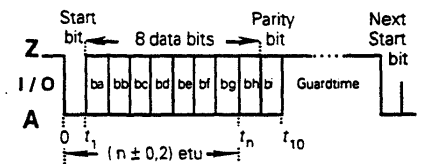
Signaux et protocoles de transport.

L'état haut (*mark*) est référencé par la lettre Z, l'état bas est référencé par la lettre A (*space*).

Horloge

La durée nominale d'un bit, désignée par le terme etu (elementary time unit), est associée par défaut à un débit de 9622 bauds, soit pour une fréquence de référence de f=3,579,545 Hz (valeur par défaut des lecteurs de cartes),
1 etu = F/D périodes d'horloge = 372/1 = F/D
F=372, D=1
L'horloge fournie par le lecteur est comprise entre 1 et 5 Mhz, les paramètres F et D sont négociables.

Format des caractères.



Un caractère comprend 1 bit start, 8 bits de données, un bit de parité (nombre de 1 pair parmi 8+1 bits). Le nombre de stop bit est égal à 2+N (N= ***Guardtime*** = 0 lors de la mise en tension).

ATR – Answer To Reset – Réponse au signal RAZ

La réponse au signal RAZ doit intervenir entre 400 et 40,000 cycles horloges. Elle comporte une série d'octets dont les deux premiers (TS, T0) sont obligatoires.

TS – Transmission. 3B pour une logique directe (positive), 3F pour une logique inverse (négative).

T0 – indique la présence d'octets d'interfaces (**TA1, TA2, TA3, TA4**) optionnels, et d'au plus 15 octets historiques (**T1, T2, …, T15**). Ces octets contiennent généralement un identifiant de la carte.

- B8=1 présence de TA1
- B7=1 présence de TB1
- B6=1 présence de TC1
- B5=1 présence de TD1
- B4 B3 B2 B0, nombre d'octets historiques (0…15)

TA1 – Valeur des paramètres d'ajustement d'etu F et D

TB1 – Paramètres de programmation de l'EPROM (obsolète). Une valeur 25 indique la non utilisation de Vpp (Vpp =Vcc)

TC1 – Fixe le nombre de bits stops excédentaires (N). La valeur par défaut est 00. FF fixe la valeur de N à 0 pour le protocole T=0 (2 stop bits) et –1 (1 stop bit) pour le protocol T=1.

TD1 – Indique le type de protocole de transport mis en œuvre

- B3 B3 B2 B1, numéro du protocole i= 0…15
- B8 B7 B6 B5 indiquent respectivement la présence d'octets TAi+1, TBi+1, TCi+1, TDi+1 fournissant des informations complémentaires sur le protocole i.
  - TA2, indique la possibilité de négocier les paramètres de transfert (PTS); B8=1 spécifie que l'option est indisponible. L'absence de TA2 (B8=0) indique la disponibilité du protocole PTS. Le numéro du protocole de négociation est renseigné par les bits B4 B3 B2 B1; B5=1 notifie l'usage de paramètres implicites, B5 et B6 sont codés à zéro.
  - TAi, (i>2) indique l'utilisation de CRC (b8=1) ou de somme de contrôle LRC (B8=0).
  - TAi+1 (i>2) et pour un numéro de protocole égal à 15 (i=15), indique le type de tension supporté et la procédure d'arrêt d'horloge supporté.
  - TB2, code la valeur de Vpp, en dixième de volts.
  - TC2, valeur d'un paramètre WI (0…255) permettant de calculer le temps d'attente maximum (WT) d'une réponse carte par le lecteur.
    WT = WI . 960 . F/f secondes, soit 1 s pour f=3,58 Mz, F=372, WI=10 (0A).
  - TBi (i>2), fournit la valeur (0,…,15) des paramètres CWI et BWI utilisés dans le protocole T=0 pour calculer, le délai maximum entre deux caractères d'un même bloc

    $$CWT = (2^{CWI} + 11)etu.$$

    Le délai maximum de réponse de la carte

    $$BWT = (2^{BWI} \; 960 \; 372 / f) + 11 \quad etu$$

    La valeur par défaut est 4D, soit CWT=1 s et BWT=1,6s pour f=3,58 Mhz, F=372, D=1.

TCK , ou exclusif de l'ensemble des octets de l'ATR. Sa présence est indiqué dans un quartet de l'octet TDi.

Quelques exemples d'ATR

Carte bancaire B0' Visa 3F 65 25 08 36 04 6C 90 00
Carte Sesame Vitale 3F 65 25 00 2C 09 69 90 00

Carte SIM Universal	3B 3F 94 00 80 69 AF 03 07 06 68 00 60 0A 0E 83 3E 9F 16
Carte JCOP 2.1 (T=1)	3B E6 00 FF 81 31 FE 45 4A 43 4F 50 32 31 07

Le protocole PTS.

Ce protocole permet de négocier la vitesse de transfert entre carte et lecteur. L'absence de TA2 signifie que la carte supporte un protocole PTS avec F=372 et D=1. La présence de TA2 notifie un mode de fonctionnement particulier et propriétaire.

Le protocole PTS consiste en l'échange d'au plus deux séries de 5 octets entre lecteur et carte. Le lecteur émet l'octet PTSS (valeur fixe FF), PTS0 (équivalent à TDi) PTS1 (équivalent à TA1) PTS2 (valeur réservée 00) PTS3 (valeur réservée 00) et PCK (octet de contrôle, résultat du ou exclusif des 4 précédents octets).

La carte répète ce bloc en cas d'acceptation. La présence des octets PTS1, PTS2 et PTS3 peut être indiquée par les bits B5 (1=Présent), B6 (1=Présent) et B7 (1=Présent) de l'octet PTS0.

En cas d'erreur la carte reste muette.

Les protocoles de transport (7816-3)

T=0.

Les octets sont transmis selon un protocole série (1 start, 8 bit, 1 bit parité, 2+N bits stop). La détection d'une erreur de parité est signalé par le récepteur (carte ou lecteur) en appliquant un zéro logique sur la ligne de transmission pendant 1 ou 2 etu.
Le nombre de répétitions n'est pas limité par la norme.
La paramètre WT définit a la fois le délai entre la fin d'une commande lecteur et la réponse carte, et le temps d'attente maximum entre deux caractères.

T=1.

C'est un protocole orienté bloc. Il est rarement utilisé en mode contact, la norme ISO 14443-4 (dite *contactless*) a défini un protocole de transport très proche du T=1. Un bloc de données comporte un prologue de trois octets (NAD, PCB, LEN), un champ information (INF) de 0 à 254 octets, et un épilogue (LRC de 1 octet ou CRC de 2 octets).

- Le NAD contient pour l'essentiel une adresse source (3 bits) et une adresse de destination (3 bits).
- PCB comporte trois types d'information
  - I(#bloc,more), les blocs sont numérotés en modulo 2, le bit more indique que le bloc n'est pas le dernier d'une liste chaînée.
  - R(#bloc,erreur), les blocs sont numérotés en modulo 2. #bloc indique le numéro du prochain bloc attendu si erreur est égal à 0.
  - S, ce champ indique la présence de diverses commandes (RESYNC, IFS, ABORT, WTX).
- LEN, longueur du champ information
- INF, tout ou partie d'une APDU
- LRC/CRC, somme de contrôle ou CRC.

CWT indique le délai maximum entre caractères
BWT indique la latence maximale entre deux blocs consécutif.

Le système de fichier ISO7816-4

L'information est organisée sous forme d'un système de fichiers hiérarchique qui comprend un répertoire maître (MF) des sous répertoires (DF) et des fichiers EF. Chacun de ses composants est identifié par un nombre de deux octets.
Il existe quatre types de fichiers

- Transparent, une zone de mémoire que l'on peut accèder librement en lecture et/ou écriture
- A enregistrement de taille fixe (*Linear Fixed*). Le fichier est un ensemble de blocs de taille fixes. Les blocs sont lus et/ou écrit de manière atomique.
- A enregistrement de taille variable (Linear Variable). Le fichier est un ensemble de blocs de taille variable.
- Cyclique. (*Cyclic Fixed*). Le fichier se présente sous la forme d'un liste circulaire de blocs de taille fixe.

Les enregistrements sont référencés par un nombre compris entre 1 et 254 (FE).

L'accès à un fichier est réalisé à l'aide d'une série de commandes de sélection de répertoire et enfin du fichier.

La réponse à une commande SELECT FILE est optionnellement une liste d'**objets** (File Control Information **FCI**) exprimés en ASN.1, c'est à dire sous une forme type longueur valeur (TLV).

Les mécanismes de sécurité associés à l'accès des fichiers sont les suivants

- Authentification par mot de passe (commande VERIFY)
- Authentification par un mécanisme de challenge (commandes GET CHALLENGE et EXTERNAL AUTHENTICATE)
- Authentification des données par un mécanisme de signature.
- Chiffrement des données.

Les commandes APDU ISO7816-4

Les commandes APDUs (Application Protocol Data Unit) contiennent soit une message de requête, soit un message de réponse à une précédente requête.

Il existe quatre type d'APDUs

- Cas 1, pas de données dans la requête, pas de données dans la réponse.
- Cas 2, pas de données dans la requête, mais des données sont présentes dans la réponse (ordre sortant).
- Cas 3, la requête comporte des données pas la réponse (ordre entrant).
- Cas 4, la requête et la réponse comportent des données (ordre entrant sortant).

Une requête APDU comprend deux parties

- d'un en tête (header) de 4 octets CLA INS P1 P2
- d'un corps de commande optionnel (body), de longueur variable

Un APDU est représenté symboliquement sous la forme,

CLA INS P1 P2 [Lc 1...3 octets] [Data] [Le 0…3 octets]

Lc désigne la longueur des données (Data) sortantes.
Le champ Le (optionnel) indique la longueur maximale de la réponse.

La représentation des quatre types d'APDUs est en conséquence la suivante

- Cas1 CLA INS P1 P2
- Cas2 CLA INS P1 P2 Le
- Cas 3 CLA INS P1 P2 Lc Data
- Cas 4 CLA INS P1 P2 Lc Data Le

On remarquera que cette représentation est ambiguë pour les cas 3 et 4.

La réponse comprend deux partie une liste d'octets optionnels (body) et un mot de status (trailer) comportant deux octets SW1 SW2

[liste d'octets optionnels] SW1 SW2

La status 90 00 indique une exécution correcte de la commande. En règle générale SW1 indique l'état de la carte, SW2 fournit des informations additives.

- 61, pas de problèmes.
- 62 ou 63, avertissement.
- 64, 65 erreur d'exécution.
- 67 commande incorrecte.

Le transport des messages APDU (TPDU)

Nous avons vu précédemment qu'il existe plusieurs protocoles de transports. Il est nécessaire de définir des règles pour par exemple réaliser la segmentation des APDUs en blocs compatible avec la taille maximale autorisés par ces protocoles. Nous ne détaillerons ici que les règles définies pour le protocol T=0.

L'en tête d'un TPDU (T=0) comporte toujours cinq octets (CLA INS P1 P2 P3) au lieu de quatre dans le cas des APDU.

Cas1. TPDU = CLA INS P1 P2 P3=00

Cas2, ordre sortant.

- Message court (short), CLA INS P1 P2 P3=Le, ordre sortant de 1 à 256 (Le=0) octets. Le status SW1=**6C** SW2=**La** peut indiquer la valeur maximale *La* supportée par la carte.
- Message long (Extended), un nombre adapté (Le/256) de requête courtes CLA INS P1 P2 P3=0 est générée. Dans la dernière requête P3 est égal au reste modulo 256 de Le. Une réponse sans erreur se termine par le status 9000 (dernier segment de la réponse) ou 61 xx (longueur P3 de la prochaine requête).

Cas3, ordre entrant.

- Message Court (short) CLA INS P1 P3 P3=Lc [Lc octets]. Ordre entrant de 1 à 255 octets. Un status 9000 indique l'exécution correcte de l'opération.
- Message long (Extended).

Une série de commandes ENVELLOPE, CLA ***C2*** P1 P2 P3 [P3 bytes 1,..,255] est utilisée pour réaliser la segmentation de l'APDU à transmettre. Le status de la réponse est 9000 en cas d'exécution correcte.

Cas4, ordre sortant/entrant

- Message court (short)

CLA INS P1 P2 P3=Lc.[Lc bytes].

Un status 61 Le indique la taille de la réponse.

La commande sortante GET_RESPONSE CLA ***C0*** P1 P2 P3=Le permet de lire le message de réponse dont la taille est Le

- Message long (Extended).

On utilise dans ce cas les commandes ENVELLOPE et GET_RESPONSE.

Format des APDUs.

CLA.

Ce paramètre est définit pour des types de cartes particuliers ou par des fabricants particuliers. 00 est la valeur ISO, A0 pour les cartes SIM, BC a été utilisé par Bull CP8, FF est réservé pour le protocole PTS.

Le quartet de poids fort est défini par le standard ISO 7816 pour les valeurs 0,8,9 et A. Dans ce cas le quartet de poids faible comporte deux bits qui indiquent l'usage du **Secure Messaging** (SM - signature et chiffrement des APDUs), et les deux derniers bits désignent un numéro de **canal logique** (compris entre 0 et 3).
L'utilisation du service SM n'est pas détaillée dans ce cours.
La notion de canal logique consiste à utiliser simultanément plusieurs application logées dans la carte.

INS – Commandes de bases

1. READ_BINARY.

CLA B0 P1 P2 Le.

Réalise la lecture de Le octets à partir de offset dans un fichier transparent.

- Si le bit de poids fort de P1 est égal à 1, EF est désigné par les 5 bits de poids faible, P2 représente l'offset.
- Sinon l'offset est égal à (256*P1) +P2

2. WRITE_BINARY.

CLA D0 P1 P2 Lc [Lc octets]
Réalise l'ecriture de Le octets à partir de offset dans un fichier transparent.

- Si le bit de poids fort de P1 est égal à 1, EF est désigné les 5 bits de poids faible, P2 représente l'offset.
- Sinon l'offset est égal à (256*P1) +P2

3. UPDATE_BINARY.

CLA D6 P1 P2 Lc [Lc octets].

Réalise l'écriture de Le octets à partir de offset dans un fichier transparent.

- Si le bit de poids fort de P1 est égal à 1, EF est désigné les 5 bits de poids faible, P2 représente l'offset.
- Sinon l'offset est égal à (256*P1) +P2

4. ERASE_BINARY.

CLA 0E P1 P2 [Lc=2 ou non présent] [2 octets ou rien]

Efface le fichier à partir de l'adresse jusqu'à la fin du fichier.

- Si Lc=2, l'offset est indiquée par les deux octets de données ;
- Sinon si le bit de poids fort de P1 est égal à 1, EF est désigné les 5 bits de poids faible, P2 représente l'offset.
- Sinon l'offset est égal à (256*P1) +P2

5. READ_RECORD
CLA B2 P1 P2 Le
Lit un enregistrement dans un fichier

- P1, numéro d'enregistrement ou premier enregistrement à lire égal à 00 indique l'enregistrement courant
- P2= 04 lecture de l'enregistrement P1=05 lecture des enregistrements à partir de P1 jusqu'à la fin du fichier.

6. WRITE_RECORD
CLA D2 P1 P2 Lc [lc octets]
Ecriture d'un enregistrement.

- P1 numéro d'enregistrement
- P2=04 enregistrement P1

7. APPEND_RECORD
CLA E2 P1 P2 Lc [Lc octets]

Pour P1 = P2 =0, cette commande ajoute un nouvel enregistrement à la fin d'un fichier à structure linéaire ou réalise l'écriture du premier enregistrement d'un fichier cyclique.

8. UPDATE_RECORD
CLA DC P1 P2 Lc [Lc octets]
Cette commande réalise la mise à jour d'un enregistrement.

Lorsque P2=04 P1 indique la valeur de l'enregistrement.

9. GET_DATA.
CLA CA P1 P2 Le.
Cette commande permet d'obtenir un objet identifié par son tag (P1 P2) dans le contexte courant (par exemple le DF sélectionné ou relatif à une implémentation particulière).

10. PUT_DATA
CLA DA P1 P2 Lc [Lc octets]

Cette commande insère un objet dans le contexte courant (DF sélectionné ou relatif à une implémentation particulière).

P1 P2 désigne le type (tag) de l'objet.

11. SELECT_FILE
CLA A4 P1 P2 Lc [Lc octets] [Le ou omis]

Sélectionne un répertoire ou un fichier.

P1 = 00 (sélection par identifiant MF DF EF)
P1= 01 (sélection DF)
P1=02 (sélection EF)
P1= 03 (Sélection du père du DF courant)

P2 =00 première occurrence.

12. VERIFY
CLA 20 P1 P2 Lc(ou omis) [Lc octets]

Cette commande réalise la vérification d'un mot de passe.
Le nombre d'essai peut être limité.

En général P1=P2=0.

13. INTERNAL_AUTHENTICATE
CLA 88 P1 P2 Lc [Lc octets] Le

Cette commande réalise un calcul d'authentification relativement à une clé interne en transférant un nombre aléatoire (challenge) délivré par le lecteur.

P1 représente la référence d'un algorithme.
P2 est égal à zéro par défaut.

Le challenge est contenu dans les Lc octets sortants.

14. EXTERNAL_AUTHENTICATE
CLA 88 P1 P2 Lc[Lc octets] Le

Cette commande met à jour l'état d'une carte en fonction du résultat d'un calcul réalisé par le lecteur à partir d'un nombre aléatoire délivré par la carte (CHALLENGE).

P1, référence d'un algorithme
P2 est égal à zéro par défaut.

15. GET_CHALLENGE
CLA 84 P1 P2 Le

Cette commande produit un nombre aléatoire de *Le* octets

16. GET_RESPONSE
CLA C0 P1 P2 Le

Cette commande est utilisé pour lire de l'information depuis la carte lorsque le protocole de transport ne le permet pas directement (par exemple T=0).

17. ENVELOPE
CLA C2 P1 P2 Lc [Lc octets] Le ou omis

Cette commande est utilisée par les ordres entrant/sortant lorsque le protocole de transport ne les supportent pas (par exemple T=0).

Identification des Applications – ISO7816-5

Un identifiant d'application (AID Application Identifier) est une nombre qui comporte 16 octets. Les 5 premier octets (RID Registered application provider IDentifier) identifient le fournisseur d'une application. Les 11 octets restant représentent un identifiant d'application.

Une application embarquée peur être activée par la commande SELECT_FILE (00A4 0400 10 [AID]) avec comme paramètre le champ AID.

En syntaxe ASN.1 AID est associé au tag 4F. Ce tag peut être inséré dans les octets historiques de l'ATR.

L'ATR est stocké dans le fichier EF_ATR en /3F00/2F01, il est sélectionné par la commande
00A4 0200 02 2F01

Un fichier EF_DIR (/3F00/2F00) contient une liste d'objet (TLV) fournissant des informations sur la carte, il est sélectionné par la commande.
00A4 0200 02 2F02

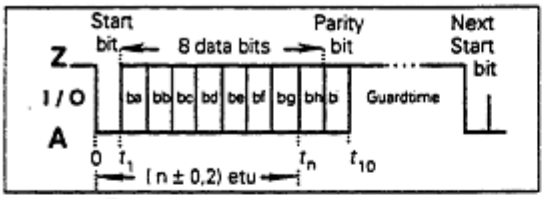
Eléments de données intersectorielles – ISO7816-6

Les éléments de données (DE Data Element) sont insérés dans des containers (DO Data Object) décrit en syntaxe ASN.1. Un DO est un arbre de DE.

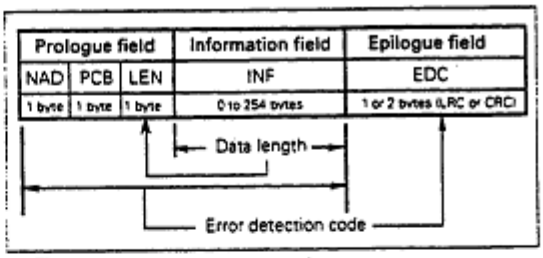
DO: TL TLV TLV ….

Un DO comporte la structure suivante,
Type, l'étiquette de l'objet (TAG) un ou deux octets
B8B6 class
B6 type primitif/construit
B5…B1 numéro d'étiquette de 0 à 30, si cette valeur est égale à 31 le deuxième octet représente le numéro d'étiquette compris entre 31 et 127 (B8=0)

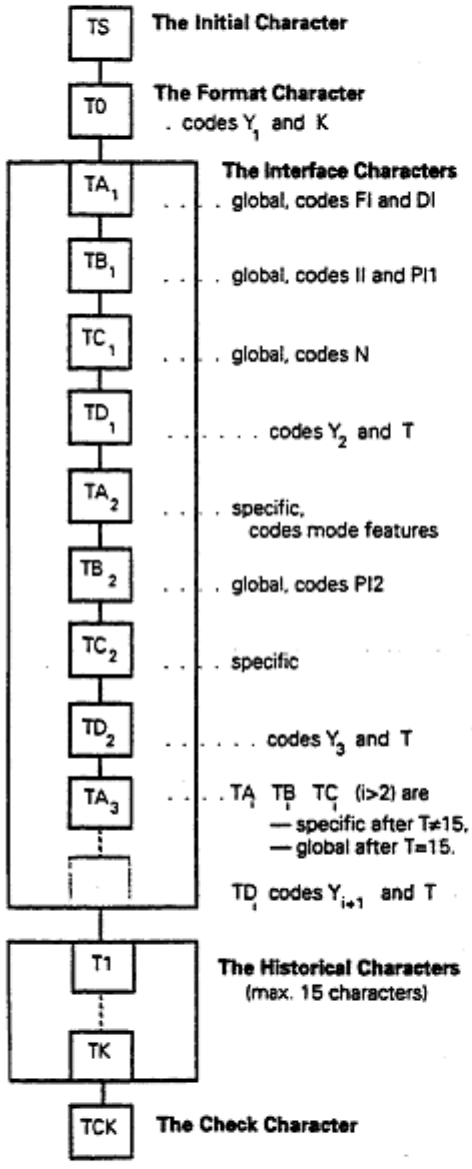
Longueur (1 à 3 octets)
Si la longueur est comprise entre 0 et 127 elle est représentée par un octet (sa valeur).
Si la longueur est comprise en 0 et 255 octets, elle est représentée par 2 octets, 81 valeur.
Si la longueur est comprise entre 0 et 65,535 elle est représentée par trois octets 82 valeur_msb, valeur_lsb.
Valeur.
Une suite de longueur octets.

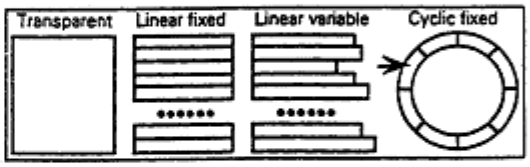


Character frame

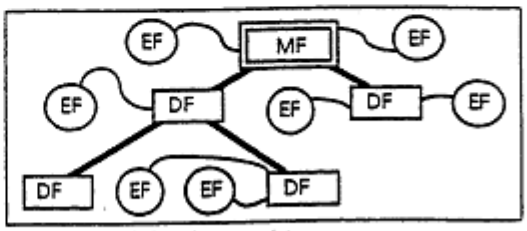


Block structure

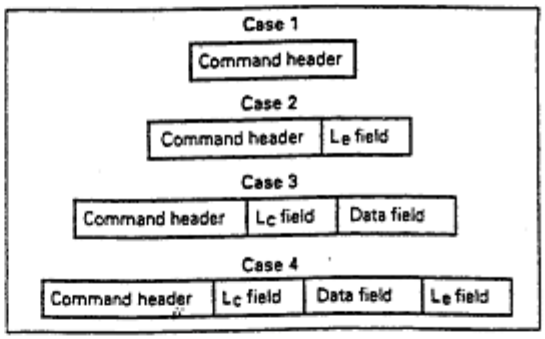




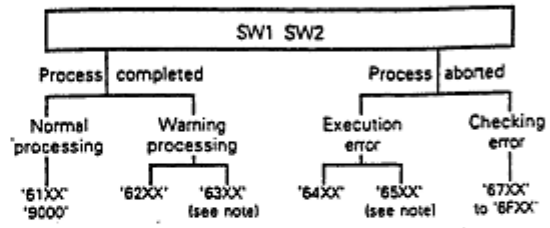
EF structures



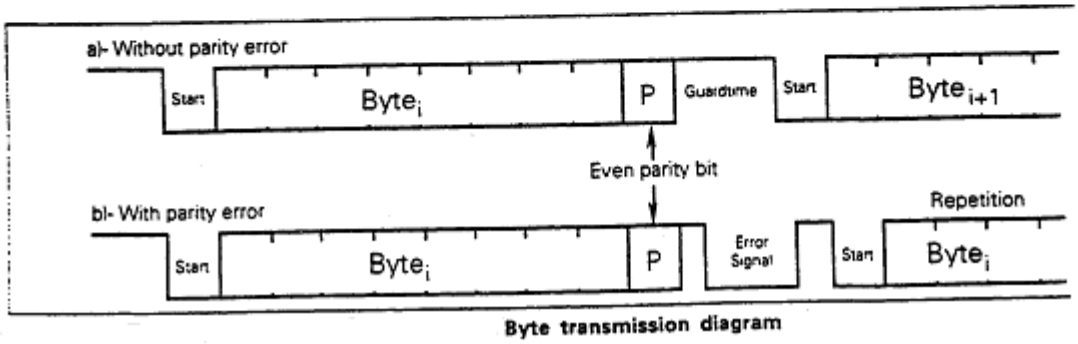
Logical file organization (example)



The 4 structures of command APDUs



Structural scheme of status bytes



Byte transmission diagram

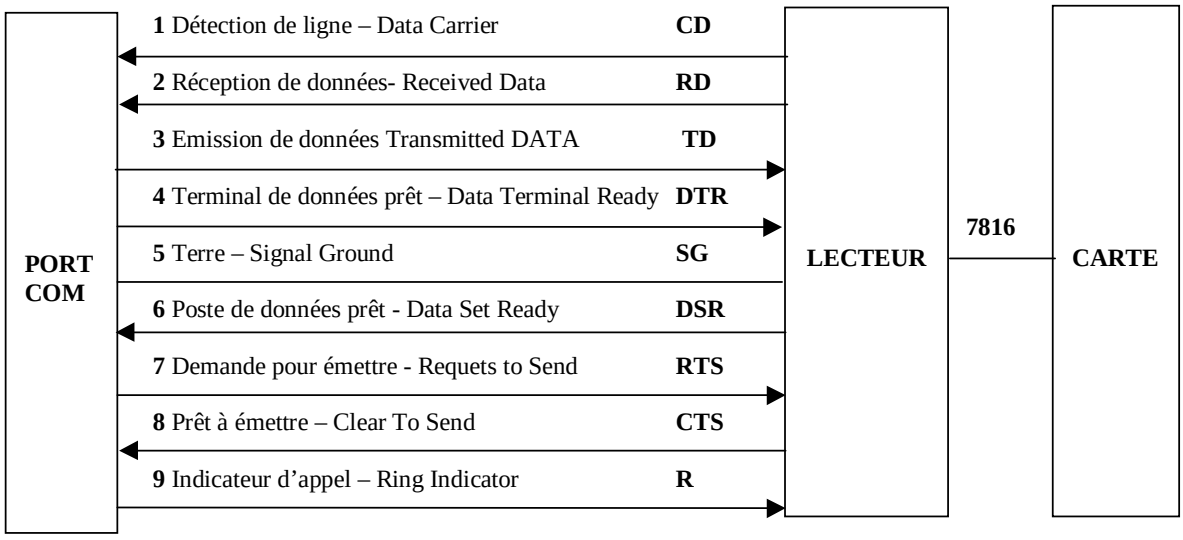
V Les lecteurs de cartes à puce.

Certain terminaux supportent de manière native les protocoles ISO7816, comme par exemple les téléphones mobiles. Cependant dans la plupart des cas, il est nécessaire d'utiliser un lecteur (ou encore *Card Acceptance Device* CAD qui réalise la conversion entre un protocole d'échange de données (RS232-C, PS2, USB) et le standard ISO7816.

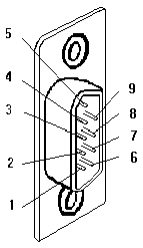
Nous présenterons par la suite le cas classique de lecteurs assurant une interface entre des liens RS232-C et ISO7816.

La norme RS232-C

Cette liaison série est disponible sur les ordinateurs personnels (port COM) et les systèmes UNIX. Une prise DB9 comporte 9 broches, ce type de connexion est utilisée pour relier un modem à un ordinateur personnel.



La fonction des lignes de communication est définie par le standard V24, dans lequel l'ordinateur des dénommé ETTD ,(Equipement Terminal de Traitement de Données dans la norme V24) et le modem ETCD (Equipement Terminal de Circuit de Données).



- Les lignes RD et TD sont dédiées à la réception et à l'émission de données.
- Le signal DTR indique que le l'ordinateur fonctionne normalement. La broche DSR du lecteur précise la présence d'un périphérique.
- Le signal RTS est activé par l'ordinateur pour signifier son intention de transmettre des données. Cette requête (ETTD) est acquittée par le signal CTS de l'ETCD.
- Les broches CD et D indiquent respectivement la détection de porteuse et la présence de sonnerie.

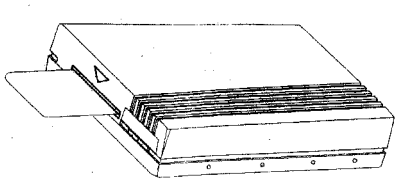
Un lecteur de cartes assure le transport des APDUs d'une part via la liaison série RS323 et d'autre part au moyen de la ligne d'entrée sortie et des protocoles définis par la norme ISO7816. Cependant le protocole mise en œuvre entre l'ordinateur et le lecteur ne répond à aucun standard et donc est différent selon les fabricants de lecteurs; nous le désignerons par la suite par le terme de CADP (*Card Acceptance Device Protocol*) Un lecteur est donc associé à un ***pilote*** (un logiciel exécuté sur le côté terminal) permettant son utilisation.

Le déploiement d'un dispositif électronique intermédiaire entre carte et terminal, implique également la disponibilité de commandes qui en assure la gestion. On distingue donc deux type de messages issus du terminal

- des messages destinés au lecteur
- et d'autres répétés par le lecteur vers la carte.

Schématiquement on peut distinguer deux types de lecteurs,

- Les lecteurs ***transparents*** qui émettent et reçoivent des octets vers/depuis la carte à puce. Ils utilisent par exemples les signaux RTS et DTR pour différencier les messages à destination du lecteur ou de la carte.
- Les lecteurs qui mettent en œuvre un protocole de type ***CADP***, dans lequel les messages échangés entre ordinateur et lecteur sont transportés par des blocs.



Le protocole ***TLP 224***, définit par l'entreprise Bull CP8 au début des années 90, est considéré comme un protocole CADP classique. Bien non normalisé ce protocole est disponible sur de nombreux lecteurs, et est utilisé par le simulateur de Java Card téléchargeable gratuitement sur le site de la compagnie SUN.

Le protocole TLP224

Les échanges entre le terminal et le terminal sont organisés en bloc. Un bloc comporte les éléments suivants

- Un **en-tête** de deux octets.
  - 1° octet
    - 60 pour une commande.
    - 60 pour la réponse positive (ACK) à une commande.
    - E0 pour la réponse négative (NACK) à une commande.
  - 2° octet
    - Longueur des données à transmettre (0…255) en excluant les champs LRC et ETX.
- Des **informations** associées au bloc. Un octet est représenté par deux caractères ASCII (0123456789ABCDEF) qui expriment sa notation en base 16. Par exemple 240 (en hexadécimal E0) est codé par deux caractères ASCII 'E' et '0'. Le nombre d'octets encapsulés dans un bloc est donc le double de la longueur de l'information à transporter.
- Un octet de contrôle **LRC** (*Longitudinal Redundancy Check*). C'est le résultat d'une suite d'opérations de *ou exclusif* réalisées sur les octets d'en tête et de données.
- L'octet **ETX** (03), qui marque la fin du bloc.

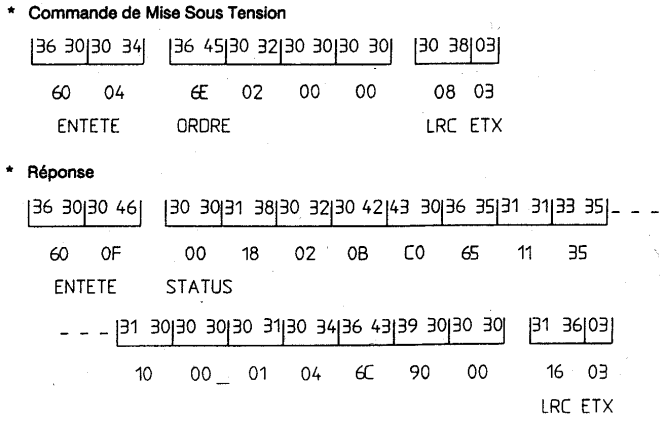
Commandes & Réponses

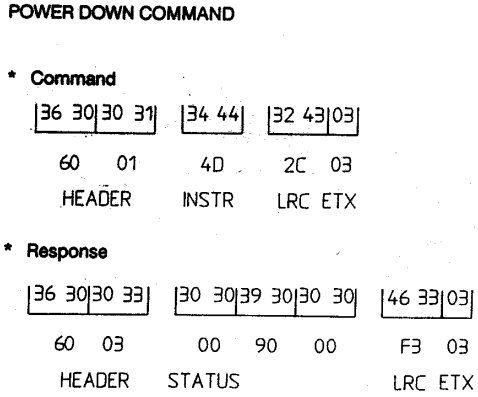
Il existe deux types de blocs différents les commandes et les réponses. Le premier octet d'information d'un commande précise l'ordre associé à cette dernière. Une réponse comporte au moins un octet dans le champ information, nommé status, qui indique le succès d'une opération (0=OK). Il existe quatre principaux type d'ordres

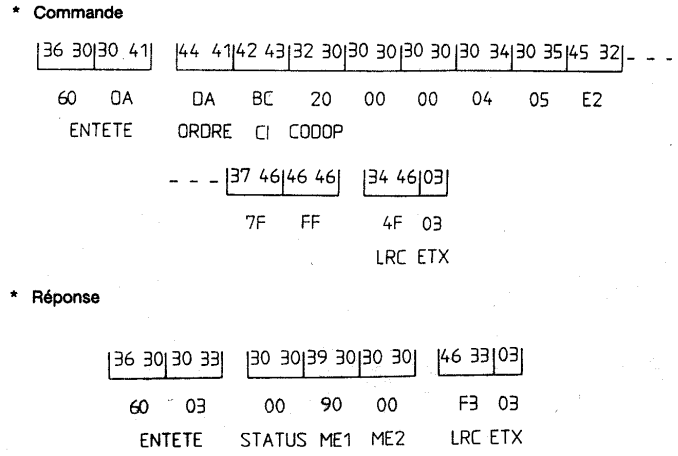
- **6E** mise sous tension de la carte
  Commande, 6E P1 00 00, P1 temps d'attente en secondes.
  Réponse
  STATUS TYPE_DE_LECTEUR TYPE_DE_CARTE ATR_LENGTH ATR_CARTE

- **4D** mise hors tension de la carte.
  Commande, 4D
  Réponse STATUS
- DA ordre entrant transparent (écriture).
  Commande DA CLA INS P1 P2 [P3 octets]
  Réponse STATUS SW1 SW2
- DB ordre sortant transparent (lecture).
  Commande DB CLA INS P1 P2 P3
  Réponse, STATUS [P3 octets] SW1 SW2

Exemples







VI Utilisation des API PC/SC pour win32.

Le standard PC/SC

File Name	Version	Manufacturer	Description	Path
scardsvr.exe	5.0.1708.1.	Microsoft Corporation	Smart Card Resource Management Server	C:\WINDOWS\SYSTEM
scarddlg.dll	5.0.1708.1.	Microsoft Corporation	SCardDlg - Smart Card Common Dialog	C:\WINDOWS\SYSTEM
smclib.sys	5.0.1922.1.	Microsoft Corporation	Smard Card Driver Library	C:\WINDOWS\System32\drivers
smclib.vxd	4.0.0.951.	Microsoft Corporation	Smart Card Driver Library	C:\WINDOWS\SYSTEM
scntvssp.dll	5.0.1708.1.	Microsoft Corporation	SCNtvSSP - Smart Card Java Native Extension Module	C:\WINDOWS\SYSTEM
scarddat.dll	5.0.1708.1.	Microsoft Corporation	SCardDat - Smart Card SSP Module	C:\WINDOWS\SYSTEM
scardmgr.dll	5.0.1708.1.	Microsoft Corporation	SCardMgr - Smart Card SSP Module	C:\WINDOWS\SYSTEM
scardsrv.dll	5.0.1708.1.	Microsoft Corporation	SCardSrv - Smart Card SSP Module	C:\WINDOWS\SYSTEM
winscard.dll	5.0.1708.1.	Microsoft Corporation	Microsoft Smart Card API	C:\WINDOWS\SYSTEM

L'ensemble des composants PC/SC pour un système windows 98.

PC/SC est un ensemble de composants win32 réalisant l'interface à des lecteurs de cartes à puces.
La spécification comporte huit parties, dont le volet 5 (*ICC Resource Manager Definition*) décrit les APIs réalisant le dialogue avec les lecteurs.

- 1 Introduction et aperçu général.
- 2 Interface lecteur cartes, un bref rappel du standard ISO 7816.
- 3 Définition des caractéristiques requises pour les pilotes de lecteurs de cartes (IFD handler), en particulier de paramètres fonctionnels accessibles par l'entité ICC, et identifiés par Tags.
- 4 Guide de référence pour la réalisation de lecteurs fonctionnant avec le port PS2, le port série ou le bus USB.
- 5 Module de gestion des ressources cartes à puces. C'est la clé de voûte de l'architecture PC/SC. Les APIs offertes à ce niveau sont réparties en cinq groupes, RESOURCE MANAGER, RESOURCEDB, RESOURCEQUERY, SCARDTRACK SCARDCOMM.

- 6 Définition des fournisseurs de services cartes à puce (Service Provider) deux types d'interface sont décrites, l'accès aux fichiers des cartes (ICCSP) et la fourniture de ressources cryptographiques.
- 7 Un ensemble de recommandations pour la mise en œuvre des services.
- 8 Décrit une structure des fichiers et un jeu d'APDUs 'au niveau des carte) pour assurer une certaine inter-opérabilité..

Quelques API essentielles

Obtenir la liste des lecteurs.

La fonction

```
SCardListReadersA((IN SCARDCONTEXT)NULL,
                  (IN LPCTSTR)NULL,
                  Reader_String,
                  &dwReaders);
```

permet d'obtenir la liste (Reader_String dans l'exemple ci dessus) des lecteurs disponible sur l'ordinateur personnel. Elle fournit une liste de chaîne séparées par le caractère null (\0). La fin de leste est marquée par un double caractère null 0 (\0\0).

Ouvrir une session PC/SC.

In faut ouvrir une session avant d'engager toute opération avec un lecteur. Cette opération est réalisée par la procédure,

```
Stat = ScardEstablishContext(dwScope,pvReserved1,pvReserved2,&hContext)
```
qui retourne une valeur nulle en cas de succès et une référence de session hContext.

Fermer une session PC/SC

La fonction

```
stat = SCardReleaseContext(hContext);
```
termine une session PC/SC préalablement ouverte.

Ouvrir une session carte.

A l'intérieur d'une session PC/SC (identifiée par sa référence hContext) on ouvre une session (dont la référence est hCard) avec une carte insérée dans un lecteur particulier.

```
Stat=SCardConnectA(hContext,Reader_List[0],
      (DWORD)(SCARD_SHARE_SHARED|SCARD_PROTOCOL_OPTIMAL),
      (DWORD)SCARD_PROTOCOL_T0,
      &hCard,
      &dwActiveProtocol);
```
La méthode *ScardConnect()* réalise cette opération et retourne la valeur zéro en cas de succès. Le lecteur est identifié par une chaîne de caractères (Reader_List[0] dans l'exemple ci dessus).

Fermeture d'une session carte

```
stat = SCardDisconnect(hCard,(DWORD)SCARD_LEAVE_CARD)
```
Cette opération est effectuée grâce à la fonction *SCardDisconnect().*

Lecture de l'ATR et test de la présence d'une carte.

```
AtrLen=sizeof(Atr);
stat = SCardState(hCard,&dwState,&dwProtocol,Atr,&AtrLen)
```
La procédure *ScardState()* fournit la valeur de l'ATR de la carte associée à la session hCard et indique également des paramètres utiles tels que la présence ou l'absence de carte dans le lecteur (carte absente en particulier). La manière la plus simple de gérer une session carte et de la refermer dès lors qu'une absence de cartes est détectée.

Echange des APDUs

La primitive ScardTransmit() permet d'envoyer un message de requête (APDU, de longueur Asize)) et retourne le message de réponse mémorisé dans un tableau de caractères (dont la taille est Rsize).

```
stat  = SCardTransmit(hCard,SCARD_PCI_T0,APDU,Asize,NULL,Response,&Rsize);
```

Nous remarquerons que de manière rigoureuse cette fonction gère des TPDUs.

Interface avec le lecteur.

La fonction ScardGetAttrib(),

```
stat=SCardGetAttrib(hCard,
      SCARD_ATTR_VALUE(SCARD_CLASS_IFD_PROTOCOL,Tag),
      (LPBYTE)&Value,
      &AttrLen);
```
lit la valeur des paramètres fonctionnels disponibles pour un lecteur donné. Un attribut de l'interface est identifié par un Tag et un type de classe auquel il appartient (SCARD_CLASS)Le tableau suivant fournit à titre indicatif une liste de paramètres utiles.

Tag	Classe	Taille Max	Valeur de l'attribut
0x300	SCARD_CLASS_ICC_STATE	1 octet	0 si carte absente
0x303	SCARD_CLASS_ICC_STATE	32 octets	ATR
0x202	SCARD_CLASS_IFD_PROTOCOL	4 octets	Horloge actuelle en Mhz
0x203	SCARD_CLASS_IFD_PROTOCOL	4 octets	Paramètre F
0x204	SCARD_CLASS_IFD_PROTOCOL	4 octets	Paramètre D
0x205	SCARD_CLASS_IFD_PROTOCOL	4 octets	Paramètre N
0x206	SCARD_CLASS_IFD_PROTOCOL	4 octets	Paramètre W

Un exemple de code Source

```
#include <windows.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>

#include "winscard.h"

#define ATRLEN 256
char Atr[ATRLEN];
int AtrLen=0    ;

#define MAX_READER 8
char Reader_String[128*MAX_READER]   ;
char Reader_List  [MAX_READER][128]  ;
int  Reader_Nb=0                     ;

int SanityCheck(SCARDCONTEXT *hContext,SCARDCONTEXT *hCard);

main(int argc, char **argv)
{ DWORD dwReaders;
  SCARDCONTEXT hContext=(SCARDCONTEXT)NULL  ;
  SCARDHANDLE hCard    =(SCARDCONTEXT)NULL  ;
  DWORD dwScope= (DWORD)SCARD_SCOPE_SYSTEM  ;
  DWORD dwState,dwProtocol,dwActiveProtocol ;
  LPCVOID pvReserved1=  (LPCVOID) NULL      ;
  LPCVOID pvReserved2=  (LPCVOID) NULL      ;
  LONG stat;

  char APDU[512],Response[512];
  int Rsize,Asize ;
  int i;
  char *pname=Reader_String;;

  Reader_String[0]=0;dwReaders=128*MAX_READER;

  // ####################################
  // Etape 0 Liste des lecteurs Installés
  // ####################################

  SCardListReadersA((IN SCARDCONTEXT)NULL,
                    (IN LPCTSTR)NULL,Reader_String,&dwReaders);

   while(strlen(pname) != 0)
  {  strcpy(Reader_List[Reader_Nb],pname)   ;
     pname += strlen(Reader_List[Reader_Nb]);
       pname ++;
```

```
      Reader_Nb++;
  }

  for(i=0;i<Reader_Nb;i++) printf("%s\n", Reader_List[i]);


  // ##############################
  // Etape 1 - Initialisation PC/SC
  // ##############################
  hContext=(SCARDCONTEXT)NULL ;
  stat = SCardEstablishContext(dwScope,pvReserved1,pvReserved2,&hContext);
  if (stat  != SCARD_S_SUCCESS)
  {printf("Erreur %8.8X PC/SC absent\n",stat); // 0x0000007B
   exit(0);}

  // ############################################################
  // Etape 2 - Attente Insertion Carte sur le premier lecteur #0
  // ############################################################
  hCard=(SCARDCONTEXT)NULL ;
  stat = SCardConnectA(hContext,Reader_List[0],
        (DWORD)(SCARD_SHARE_SHARED|SCARD_PROTOCOL_OPTIMAL),
        (DWORD)SCARD_PROTOCOL_T0,
        &hCard,
        &dwActiveProtocol);

  if (stat != SCARD_S_SUCCESS)
         {printf("Erreur %8.8X Carte Absente\n",stat);
        // 0x80100069 Carte Absente
        // 0x80100009 Nom de lecteur incorrect
        exit(0);}

  // ######################################################
  // Etape 3 - Obtention de l'ATR ou Test Présence Carte
  // ######################################################
   AtrLen=ATRLEN;
   stat = SCardState(hCard,&dwState,&dwProtocol,Atr,&AtrLen);

   // Si la carte est absente retourner à l'étape 2
   if ((stat != SCARD_S_SUCCESS) || (dwState == SCARD_ABSENT))
   {   stat = SCardDisconnect(hCard,(DWORD)SCARD_LEAVE_CARD);
       hCard=(SCARDCONTEXT)NULL ;
         //0x80100069 Carte Absente
         printf("Erreur %8.8X carte absente\n",stat);
         SanityCheck(&hContext,&hCard);exit(0);}

   for(i=0;i<AtrLen;i++) printf("%2.2X ",0xFF & Atr[i]);printf("\n");

   // ######################
   // Etape 4 Envoi de TAPDU
   // ######################

   // Select DF_GSM (7F20)

APDU[0]=(char)0xA0;APDU[1]=(char)0xA4;APDU[2]=(char)0;APDU[3]=(char)0;APDU[
4]=(char)0x02;
   APDU[5]=(char)0x7F;APDU[6]=(char)0x20;
   Asize=7;
   Rsize= sizeof(Response);
   stat =SCardTransmit(hCard,SCARD_PCI_T0,APDU,Asize,NULL,Response,&Rsize);

   // Carte Absente
```

```
   if   (stat != SCARD_S_SUCCESS)
        { printf("Erreur %8.8X carte absente\n",stat);
        SanityCheck(&hContext,&hCard);exit(0);}

   // 9F Le
  for(i=0;i<Asize;i++) printf("%2.2X ",0xFF & APDU[i]);printf("\n")    ;
  for(i=0;i<Rsize;i++) printf("%2.2X ",0xFF & Response[i]);printf("\n");

  if ((Rsize != 2) || (Response[0] != (char) 0x9F) ) // Test du status de
la réponse
  { SanityCheck(&hContext,&hCard);exit(0);}


  // GET_RESPONSE A0 C0 00 00 Le

APDU[0]=(char)0xA0;APDU[1]=(char)0xC0;APDU[2]=(char)0;APDU[3]=(char)0;APDU[
4]=Response[1];
   Asize=5;
   Rsize= sizeof(Response);
   stat= SCardTransmit(hCard,SCARD_PCI_T0,APDU,Asize,NULL,Response,&Rsize);

   // Carte Absente
   if   (stat != SCARD_S_SUCCESS)
   { printf("Erreur %8.8X carte absente\n",stat);
     SanityCheck(&hContext,&hCard);exit(0);}

   //
  for(i=0;i<Asize;i++) printf("%2.2X ",0xFF & APDU[i]);printf("\n");
  for(i=0;i<Rsize;i++) printf("%2.2X ",0xFF & Response[i]);printf("\n");

   // ############################################
   // Etape 5 - Abandon contexte lecteur et PC/SC
   // ############################################

   GetListAttrib(hCard) ;

   if (hCard != (SCARDCONTEXT)NULL)
   stat = SCardDisconnect(hCard,(DWORD)SCARD_LEAVE_CARD) ;

   if (hContext != (SCARDCONTEXT)NULL)
   stat = SCardReleaseContext(hContext);

   hContext=(SCARDCONTEXT)NULL ;
   hCard   =(SCARDCONTEXT)NULL ;

exit(0);
return(0);
}

int SanityCheck(SCARDCONTEXT *hContext,SCARDCONTEXT *hCard)
{ LONG stat ;

if (hCard != (SCARDCONTEXT)NULL)
stat = SCardDisconnect(*hCard,(DWORD)SCARD_LEAVE_CARD) ;
if (hContext != (SCARDCONTEXT)NULL)
stat = SCardReleaseContext(*hContext);

*hContext=(SCARDCONTEXT)NULL; *hCard   =(SCARDCONTEXT)NULL ;
 return(0);
}
```

```
//###################################
//       Paramètres divers
//###################################
int GetListAttrib(SCARDCONTEXT hCard)
{ ULONG Value      ;
  DWORD AttrLen    ;
  LONG  stat       ;
  ULONG Tag        ;

  char IFD[][5]={"T=..","CLK.","F...","D...","N...","W...","IFSC","IFSD",
                 "BWT.","CWT.","EBC.","EBWT"};

  for(Tag=0x201;Tag<=0x20C;Tag++)
  { AttrLen = sizeof(Value);
    stat=SCardGetAttrib(hCard,
                        SCARD_ATTR_VALUE(SCARD_CLASS_IFD_PROTOCOL,Tag),
                        (LPBYTE)&Value,
                        &AttrLen);
  if (stat == SCARD_S_SUCCESS && (AttrLen==4) )
            printf("%s Tag=%4.4X  Value= %8.8X  %d\n",
                   IFD[Tag-0x201],Tag,Value,Value);
  }

  return(0);
}
```

Trace de l'exemple

```
// Liste des lecteurs
Bull SmarTLP3 0
Schlumberger Reflex USB Smart Card Reader 0

// ATR
3B 3F 94 00 80 69 AF 03 07 06 68 00 60 0A 0E 83 3E 9F 16

// Sélection du DF GSM
A0 A4 00 00 02 7F 20
9F 16
A0 C0 00 00 16
00 00 00 00 7F 20 02 00 00 00 00 00 09 11 00 17 0A 00 83 8A 83 8A 90 00

// Paramètres lecteurs divers
T=.. Tag=0201  Value= 00000001  1
CLK. Tag=0202  Value= 00001388  5000
F... Tag=0203  Value= 00000200  512
D... Tag=0204  Value= 00000200  512
N... Tag=0205  Value= 00000000  0
W... Tag=0206  Value= 0000000A  10
IFSC Tag=0207  Value= 00000000  0
IFSD Tag=0208  Value= 000000FE  254
BWT. Tag=0209  Value= 00000000  0
CWT. Tag=020A  Value= 00000000  0
```

Exemple d'usage du package javax.smartcardio de JAVA 1.6

Code source

```
import javax.smartcardio.*;

public class card {

public static void  main(String args[]) {

String ReaderName =
"SCM Microsystems Inc. SCR3310 USB Smart Card Reader 0";

byte []rep = null;

byte [] Select_DF_GSM =
{(byte)0xA0,(byte)0xA4,(byte)0x00,(byte)0x00,(byte)0x02,(byte)0x7f,
(byte)0x20};

byte [] More=
{(byte)0x00,(byte)0xC0,(byte)0x00,(byte)0x00,(byte)0x00 };

Card card =null;
TerminalFactory factory = null;
CardTerminal terminal = null;
ResponseAPDU r=null;
ATR atr=null;
CardTerminals cardterminals=null;
CardChannel channel =null;

factory = TerminalFactory.getDefault();
cardterminals = factory.terminals();
System.out.println("Terminals: " + cardterminals);

// List<CardTerminal> terminals = factory.terminals().list();
// System.out.println("Terminals: " + terminals);

terminal =
(CardTerminal)factory.terminals().getTerminal(ReaderName);
try {
     terminal.waitForCardPresent(1000);
     if(terminal.isCardPresent())
     System.out.println("Card Inserted!!");
     else
     System.out.println("TimeOut reached!!");
     }
catch (CardException e) {};

try { card = terminal.connect("*");}
catch (CardException e) {};

atr = card.getATR()  ;
rep = atr.getBytes() ;
System.out.println("ATR: " + b2s(rep));
channel = card.getBasicChannel();
```

```
try   {channel.transmit(t2a(Select_DF_GSM)); }
catch (CardException e) {};

rep= r.getBytes();
System.out.println("response: " + b2s(rep));

More[4]= rep[1];
try {  r = channel.transmit(t2a(More)); }
catch (CardException e) {};

rep= r.getBytes();
System.out.println("response: " + b2s(rep));

try { card.disconnect(false);}
catch (CardException e) {};

System.exit(0);}

static public CommandAPDU t2a(byte[] t)
{ CommandAPDU a =null ;

if (t.length == 4)
{ a = new CommandAPDU(0xFF & t[0],0xFF & t[1],0xFF & t[2],0xFF &
t[3]); }

else if (t.length == 5)
{if (t[4] != 0)
 a = new CommandAPDU(0xFF & t[0],0xFF & t[1],0xFF & t[2],
 0xFF &  t[3],0xFF & t[4]);
 else
 a = new CommandAPDU(0xFF & t[0],0xFF & t[1],0xFF & t[2],
 0xFF & t[3], 256);
}

else if (t.length > 5)
{ byte b[] = new byte[t.length-5];
  System.arraycopy(t,5,b,0,b.length);
  a = new CommandAPDU(0xFF & t[0],0xFF & t[1],0xFF & t[2],
  0xFF & t[3],b,256);
}
 return a;
}

final static char[] cnv  =
{'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
static public String b2s(byte[] value)
{  String cline="" ;
    int i;
   char[] c= new char [3*value.length];
for(i=0;i<value.length;i++)
    { c[3*i]    = cnv[(((int)value[i]) >> 4)&0xF]  ;
      c[3*i+1]  = cnv[((int)value[i]) & 0xF]      ;
      c[3*i+2]  = ' '                             ; }

cline= String.copyValueOf(c,0,3*value.length) ;
```

```
return(cline);
}
}
```

Run Time

Card Inserted!!
ATR: 3B 9E 95 80 1F C3 80 31 E0 73 FE 21 1B 66 D0 00 17 B4 00 00 A4
response: 9F 23
response: 00 00 00 00 7F 20 02 00 00 00 00 00 16 93 00 20 04 00 83 8A 83 8A 00 03 00 00 00 00 00 00 00 00 2F 06 02 90 00

VII La technologie javacard.

Quelques rappels sur la technologie Java.

La technologie java est organisée autour d'objets (java), matérialisés par des fichiers *ClassFile* (dont l'extension est *.class*) , obtenus après compilation d'un fichier source (.java) par le compilateur *javac*.
Le *code byte* java utilise des index pour référencer les objets, les méthodes (*methods*) et les variables (*fields*). Une classe est identifiée par un nom (*Fully Qualified Name*) dont la partie gauche désigne un *package* (paquetage, un chemin d'accès) et dont la partie droite représente le nom de la classe.
La table *constant_pool* établit une relation entre un index et une information (par exemple le nom d'une classe…). Lors du chargement et de l'exécution d'une classe, la machine virtuelle java (JVM) réalise une nouvelle table (*runtime constant_pool*) qui permet de lier la classe avec l'environnement courant.

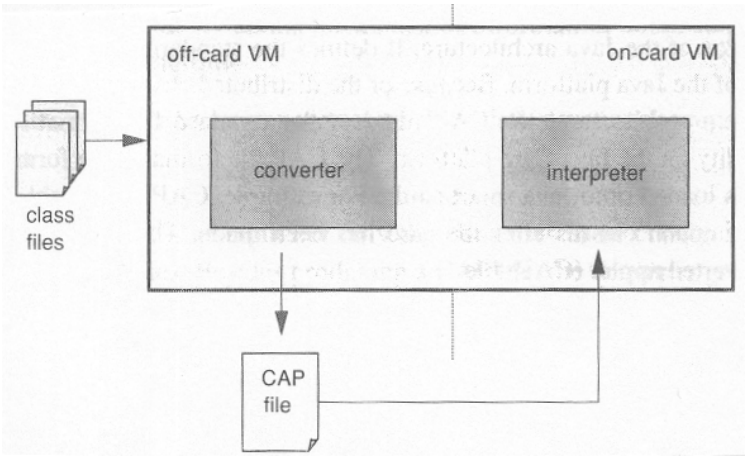
ClassFile {
u4 magic; // 0xCAFEBABE ;
u2 minor_version ; // plus petite version supportée
u2 major_version ; // plus grande version supportée
u2 constant_pool_count ; // nombre d'entrée de la table + 1
cp_info constant_pool[constant_pool_count-1] ; // table des constantes *cp_info*
u2 access_flag; // privilèges d'accès et propriétés de cette classe
u2 this_class ; // index de classe dans la table des constantes
u2 super_class ; // index de la super classe dans la table des constantes.
u2 interfaces_count ; // le nombre d'interfaces
u2 interfaces[interfaces_count] ;// tableau d'index dans la table des constantes.
u2 fields_count ; // nombres de variables
field_info info[fileds_count] ; // Tableau de descripteurs *field_info* des variables
u2_method_count ; // nombre de méthodes
method_info methods[methodes_count] ; // tableau de descripteurs *method_info*
u2 attributes_count ;//nombre des attributs (valeurs variables, code des méthodes …) de la classe.
attributes_info attributes[attributes_count] ; // tableau des valeurs (*attributes_info*) des attributs
}

Dans l'univers java les classes sont stockées dans des répertoires particuliers (identifiés par la variable d'environnement *classpath*) ou dans des serveurs WEB. L'adaptation de la technologie java aux cartes à puce implique en particulier une adaptation des règles de localisation des classes à cet environnement particulier.

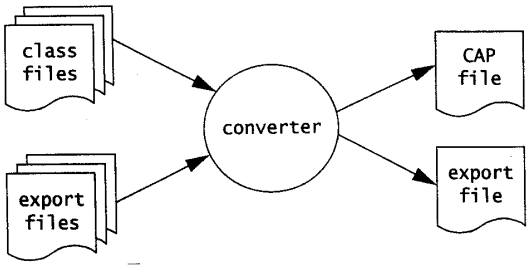
La norme JavaCard 2.x

Le langage *javacard* est un sous ensemble du langage java (paquetage *Java.lang)*. Il supporte les éléments suivants,

Eléments supportés.	Principaux éléments non supportés
Les types primitifs boolean, byte short. Le type primitif int est optionnel	Types primitifs long, double, float, char
Tableau à une dimension	Tableau à plusieurs dimensions
Paquetages, classes, interfaces exceptions. Héritages, objets virtuels, surcharge, création d'objets (new).	Les objets String. Chargement dynamique de classe (new lors du *runtime*)

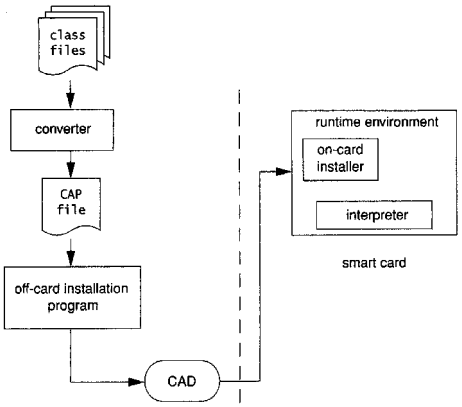


Compte tenu de la puissance de traitement d'une carte à puce la machine virtuelle est constituée par deux entités distinctes, l'une est logée sur une station de travail ou un ordinateur personnel (off-card VM), et l'autre est embarquée dans la carte (on-card VM).

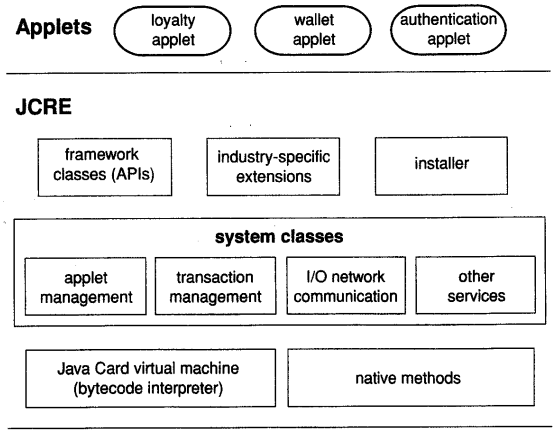


Un ensemble de fichiers java est compilé par un compilateur javac standard qui produit des fichiers .class. Un programme nommé *converter* implémente une partie de la machine virtuelle, il vérifie la comptabilité des classes avec le langage javacard, charge et lient ces objets dans un paquetage. Le résultat de ces opérations est stocké dans un fichier .*CAP* (Converted Applet). Un fichier dit d'exportation (.exp), contient les déclarations des différents éléments du paquetage afin d'en permettre une utilisation ultérieure par un autre paquetage, lors de l'opération de conversion.

Le fichier **.CAP** est chargé dans la carte grâce à la collaboration de deux entités le *off-card installation program* et le *on-card installer*. Cette opération consiste en la segmentation du fichier en une série d'APDUs qui sont généralement signés (et parfois chiffrés) à l'aide de clés secrètes afin d'authentifier leur origine.



Un interpréteur Java (*interpreter*) logé dans la carte réalise l'exécution du *code byte* lors de l'activation d'un Applet.



Le *Java Card Runtime Environnement* (JCRE) est un ensemble de composants résidants dans la carte.

- *Installer*, ce bloc réalise le chargement d'un Applet dans la carte.
- *APIs*, un ensemble de quatre paquetages nécessaires à l'exécution d'un Applet.
  - Java.lang (Object, Throwable, Exception).
  - javacard.framework.
  - javacard.security.
  - javacardx.security.
- Des composants spécifiques à une application particulière.
- Un ensemble de classes (*system classes*) qui réalisent les services de bases, tels que
  - Gestion des Applets
  - Gestion des transactions
  - Communications
  - Autres…
- La machine virtuelle et les méthodes natives associées.

Le dialogue avec l'entité JCRE est assuré à l'aide d'APDUs.

Cycle de développement d'un Applet.

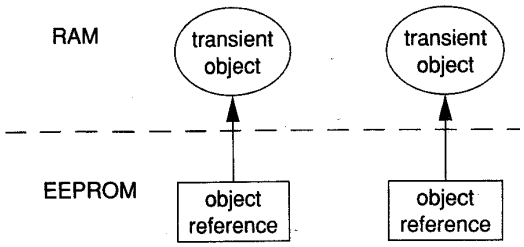
Un Applet est un ensemble de classes regroupées dans un même fichier CAP. Il est identifié de manière unique par un AID (Application Identifier un nombre de 16 octets) qui

conformément à la norme ISO7816-5 comporte un préfixe de 5 octets (RID – Resource IDentifier) et une extension propriétaire de 11 octets (PIX – Proprietary Identifier eXtension).

Un paquetage est également identifié par un AID, il est lié aux paquetages identiques (même AID) lors de son chargement.

L'environnement SUN comporte un émulateur de carte qui permet de tester le bon fonctionnement d'un Applet avant son chargement réel.

Les objets JavaCard.



En raison de la nature des cartes à puce le langage JavaCard supporte deux types d'objets, des objets de type persistant (*persistent object*) stockés dans la mémoire non volatile (E^2PROM), et des objets volatiles (*transient objets*) logés dans la mémoire RAM, dont l'image disparaît à chaque utilisation de la carte.

Par exemple

```
byte[] buffer = JCSystem.makeTransientByteArray(32,JCSystem.CLEAR_ON_DESELECT);
```

réalise la création d'un objet *transient*,détruit lors de la désélection de l'Applet.

Notion d'Atomicité.

Une carte à puce peut manquer d'énergie électrique lors de l'exécution d'un programme. La norme JC introduit une notion d'opération atomique relativement à la mémoire non volatile. Une procédure de transfert mémoire *atomique* est entièrement exécutée ou dans le cas contraire est sans effet.
De telles méthodes sont offerts par la classe javacard.framework.Util,

```
public static short ArrayCopy (byte [] src, short srcOffset, byte[] dest, short DestOffset, short length);
```

Notion de transaction.

Une transaction est une suite d'opérations qui modifient la mémoire non volatile. Cette suite de mises à jour est appliquée (*commit*) uniquement à la fin de la transaction, les modifications générés par une transaction sont appliquées de manière atomique ou ignorées.

Une méthode de la classe JCSystem réalise une telle transaction

```
JCSystem.beginTransaction() ;
// suite de modification dans la mémoire non volatile
JCSystem.commitTransaction() ;
```

Autres méthodes utiles
JCSystem.abortTransaction(), arrête une transaction en cours
JCSystem.getMaxCommitCapacity() , retourne le nombre maximum d'octets pouvant être mémorisés.
JCSystem.getUnusedCommitCapacity(), retourne le nombre d'octets encore disponibles.

Partage d'objets.

Un *contexte* JavaCard est un ensemble d'Applets qui appartiennent au même paquetage. Le système d'exploitation interdit toute interaction entre contextes pour des raisons évidentes de sécurité.
Cependant un tableau de type primitif (bytes, short) avec le préfix *global* peut être accédé en lecture par différents contextes.

Afin de permettre à plusieurs applications d'échanger de l'information, comme par exemple la mise à jour de points de fidélité, on a introduit la notion d'interface partageable (*Shareable Interface Object* – SIO).

Un Applet dit *serveur* implémente un interface partageable. Un Applet dit *client* peut utiliser cette interface.

Exemple d'un Applet serveur

```
Public interface Miles extends Shareable
{public add(short x) ;}
public class AirMiles extends Applet implements Miles
{ static short count=0;
Public add(short x){ count += x ;}
// Méthode à surcharger pour autoriser le partage de l'interface
Public Shareable getShareableInterfaceObject(AID client_aid, byte parameter)
{// Vérification du paramètre AID
  return((Shareable)this;
}}
```

Exemple d'un Applet Client.

```
// Recherche de l'applet serveur identifié par son AID.
// public static AID lookupAID (byte[] buffer, short offset, byte length) ;
server_aid = lookupAID(buffer,0,(byte)16);
// Obtention d'une interface partageable
// public static Shareable getAppletShareableInterfaceObject(AID server_aid,byte parameter) ;
I = getAppletShareableInterfaceObject(server_aid,(byte)0); // retourne SIO ou null
I.add((short)5000) ;
```

Ressources cryptographiques.

L'accès aux fonctions cryptographiques telles que empreintes (*digest, md5,* sha-1…), algorithmes à clés symétriques (DES, AES…) ou à clés dissymétriques (RSA) est un point essentiel de la technologie JavaCard. Ces procédures sont écrites en code natif (assembleur, C) et généralement protégées contre les différentes attaques connues.
Le paquetage javacard.security comporte un ensemble d'interfaces et de classes, qui assurent le lien avec les ressources cryptographiques.

Digest.

On obtient un objet *MessageDigest* à l'aide de la méthode,

```
Public static MessageDigest getInstance(byte algorithm, boolean externalAccess);
```

Par exemple

```
MessageDigest md5 = MessageDigest.getInstance(MessageDigest.ALG_MD5,false) ;
MessageDigest sha  =  MessageDigest.getInstance(MessageDigest.ALG_SHA1,false) ;
```

Les méthodes et réalise les opérations usuelles des fonctions de *hash* telles mise à jour du calcul *update* et calcul final *doFinal.*

```
sha1.update(byte[] buffer, short offset, short length) ;
sha1.doFinal(byte[]buffer, short offset, short length) ;
```

Chiffrement.

La classe *KeyBuilder* permet de construire une clé (*interface Key*) non initialisée. Des interfaces spécifiques sont associées à chaque algorithme.

Exemples.

- DESKey des_key;
  des_key = (DESKey) KeyBuilder.buildKey(KeyBuilder.TYPE_DES, KeyBuilder.LENGTH_DES, false) ;
- RSAPrivateKey rsa_private_key ;
  rsa_private_key =
  (RSAPrivateKey) KeyBuilder.buildKey(KeyBuilder.TYPE_RSA_PRIVATE,
  KeyBuilder.LENGTH_RSA_512, false) ;

Une clé RSA est initialisée à l'aide de méthodes fournies par l'interface RSAPrivateKey

```
rsa_private_key.setExponent(byte[] buffer,short offset, short length) ;
rsa_private_key.setModulus(byte[] buffer,short offset, short length) ;
```

Une clé symétrique DES peut utiliser la méthode setKey de l'interface DESKey.

```
DesKey.setKey(byte[] KeyData, short offset)
```

Un objet Cipher réalise les opérations de chiffrement/déchiffrement.

```
Cipher cipher;
cipher = Cipher.getInstance(Cipher.ALG_DES_CBC_NO_PAD,false);
cipher.init((Key)des_key,cipher.MODE_ENCRYPT);
```

Les méthodes *update* et *doFinal* chiffrent les données.

```
cipher.update(byte[] inBuf, short inOffset, short inLength, byte[] outBuff, short outOffset);
cipher.doFinal(byte[] inBuf, short inOffset, short inLength, byte[] outBuff, short outOffset);
```

Signature.

La mise en œuvre d'une opération de signature est très similaire à celle d'un chiffrement.

On obtient une référence sur un objet signature,
Signature signature;
signature = Signature.getInstance(Signature.ALG_RSA_MD5_RFC2409,false);

La clé associée à la signature est initialisée par une méthode init
signature.init(Key thekey, byte theMode) ou
signature.init(Key thekey, byte theMode, byte[] bArray, short bOffset, short bLength);

Les opérations de signature utilisent les procédures
signature.update(byte[] buf, short offset, short length)
Signature.sign(byte[] buf, short offset, short length, byte[] sig_buf, short sig_offset);

Nombres aléatoires.

La classe *RandomData* réalise la génération de nombre aléatoire.

Exemple.

```
RandomData random_data ;
random_data = getInstance(RandomData.ALG_SECURE_RANDOM) ;
random_data.setSeed(byte[] seed, short offset, short length) ;
random_data.generateData(byte[] random, short offset, short length) ;
```

La classe Applet.

Une application (on emploie parfois le terme *cardlet*) est articulée autour de l'héritage d'une classe Applet.

Public class MyApplet extends Applet {} ;

La classe Applet comporte 8 méthodes.

- Static void install(byte[]bArray, short bOffset, byte bLength)
  Cette méthode est utilisée par l'entité JCRE lors de l'installation d'un applet. Le paramètre indique la valeur de l'AID associé à l'application. Les objets nécessaires à l'application sont crées lors de l'appel de cette procédure (à l'aide de l'opérateur *new*).
- Protected void register()
  Cette méthode réalise l'enregistrement de l'applet avec l'AID proposée par le JCRE.
- Protected void register(byte[]bArray, short bOffset, byte bLength)
  Cette méthode enregistre l'applet avec un AID spécifié dans les paramètres d'appel.
- Protected boolean select()
  Cette méthode est utilisée par le JCRE pour indiquer à l'applet qu'il a été sélectionné. Elle retourne la valeur *true*.
- Protected boolean selectingApplet()
  Indique si un APDU SELECT (CLA=A4) est associé à l'opération de sélection de l'applet.
- Protected void deselect()
  Cette méthode notifie la déselection de l'applet.
- Shareable getShareableIntercaeObject(AID clientAID, byte parameter)
  Cette méthode est surchagée par une application (serveur) qui désire offrir une interface partageable.
- Abstract void process(APDU apdu) throws ISOException
  Cette méthode est la clé de voûte d'un applet. Elle reçoit les commandes APDUs et construit les réponses. L'exception ISOException comporte une méthode ISOException.throwIt(short SW) utilisée pour retourner le status word (SW) d'un ordre entrant.

Le traitement des APDUs

La classe APDU comporte toutes les méthodes nécessaires à la réception des commandes et à la génération des réponses.

Les méthodes les plus employées sont les suivantes

- static byte getProtcol()
  Retourne le type du protocole de transport ISO7816 (0 ou 1). En règle générale seul le protocole T=0 est supporté.
- byte[] getBuffer()
  Cette méthode une référence sur un tampon de réception (situé en RAM). La taille de ce bloc est d'au moins 37 octets (5+32).

- Short setIncomingAnd Receive()
  Cette méthode retourne le paramètre Lc d'une commande entrante. Les Lc octets sont mémorisés à partir de l'adresse 5 du buffer (tampon) de réception.
- Void setOutgoingAndSend(short Offset, short Length)
  Cette méthode transmet un message de réponse contenu dans le tampon de réception. Dans le cas d'une APDU sortante (Lc=0 ET Le#0) le statut 9000 est ajouté à la fin du message. Dans le cas d'une APDU entrante et sortante (Lc#0 et Le#0) le mot de statut 61 Le est généré afin de permettre une lecture ultérieure par un GET_RESPONSE (CLA C0 00 00 Le).

Divers & utile.

- ISOException.throwIt(short SW), permet de quitter la méthode process en générant un mot de statu SW. La classe ISO7816 contient une liste de valeurs utiles.
- Les méthodes

```
Util.arrayCopy(byte[] Source,short OffsetSource,byte[] Destination,short OffsetDestadr, short Length;
ET
Util.arrayCopyNonAtomic(byte[] Source,short OffsetSource,
                        byte[] Destination,short OffsetDestadr, short Length;
```

Réalisent des transferts de mémoire à mémoire avec ou sans atomicité.

- Short Util.makeShort(byte MSB,byte LSB) fabrique un entier short à partir de deux octets.

Utilisation des outils SUN.

Un exemple d'Applet – DemoApp.

```
package Demo;
import javacard.framework.*;

public class DemoApp extends Applet
{
       final static byte  BINARY_WRITE = (byte) 0xD0       ;
       final static byte  BINARY_READ  = (byte) 0xB0       ;
       final static byte  SELECT       = (byte) 0xA4       ;
       final static short NVRSIZE      = (short)1024       ;
       static byte[] NVR               = new byte[NVRSIZE] ;

public void process(APDU apdu) throws ISOException
  { short adr,len;

  byte[] buffer = apdu.getBuffer() ; // lecture CLA INS P1 P2 P3

  byte cla = buffer[ISO7816.OFFSET_CLA];
  byte ins = buffer[ISO7816.OFFSET_INS];
  byte P1  = buffer[ISO7816.OFFSET_P1] ;
  byte P2  = buffer[ISO7816.OFFSET_P2] ;
  byte P3  = buffer[ISO7816.OFFSET_LC] ;

  adr = Util.makeShort(P1,P2) ;
  len = Util.makeShort((byte)0,P3)  ;

  switch (ins) {

              case SELECT: return;

              case BINARY_READ:
                     if (adr < (short)0)
                     ISOException.throwIt(ISO7816.SW_WRONG_LENGTH) ;
                     if ((short)(adr + len) > (short)NVRSIZE)
                            ISOException.throwIt(ISO7816.SW_WRONG_LENGTH) ;
                     Util.arrayCopy(NVR,adr,buffer,(short)0,len);
                     apdu.setOutgoingAndSend((short)0,len);
                     break;

              case BINARY_WRITE:
                     short readCount = apdu.setIncomingAndReceive(); // offset=5 ...
                     if (readCount <= 0)ISOException.throwIt(ISO7816.SW_WRONG_LENGTH) ;
                     if (adr < (short)0)
                            ISOException.throwIt(ISO7816.SW_WRONG_LENGTH) ;
                     if ((short)(adr + len) > (short)NVRSIZE )
                     ISOException.throwIt(ISO7816.SW_WRONG_LENGTH) ;
                     Util.arrayCopy(buffer,(short)5,NVR,adr,len);
                     break;

              default:
                     ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
              }
}

protected DemoApp(byte[] bArray,short bOffset,byte bLength)
{  register();
..//register(byte[] bArray,short bOffset,byte bLength);
  }

public static void install( byte[] bArray, short bOffset, byte bLength )
{  new DemoApp(bArray,bOffset,bLength); }

public boolean select() {return true;}

public void deselect(){}
}
```

Cahier des charges de DemoApp

Cet applet réalise des lectures et des écritures dans la mémoire non volatile (NVR).

- Package Demo
- Applet AID 0x4a:0x54:0x45:0x53:0x54:0x30:0x30
- Package AID 0x4a:0x54:0x45:0x53:0x54:0x30
- TPDUs supportées

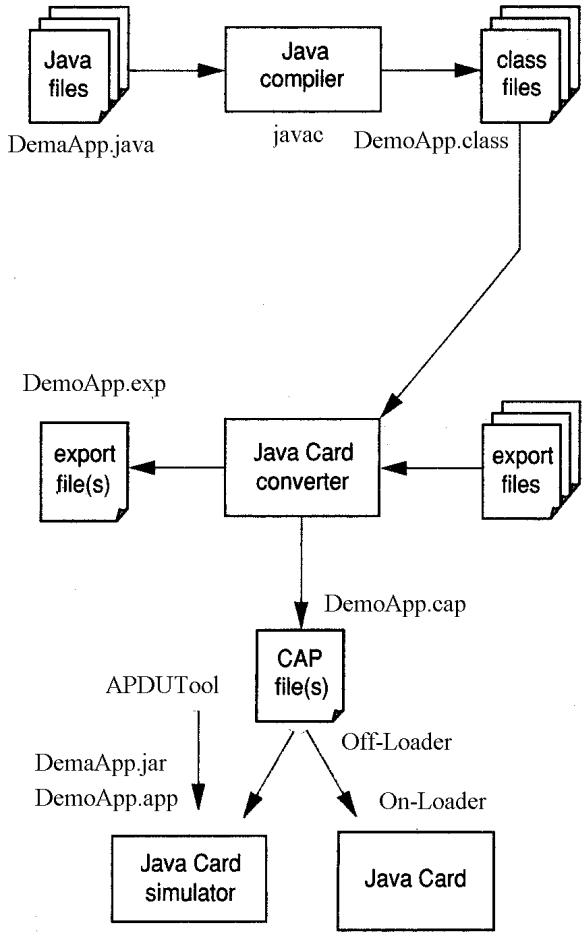
  Read 00 B0 P1 P2 Le.

  Write 00 D0 P1 P2 Lc [Lc octets].

  SelectAID 00 A4 04 00 07 4a 54 45 53 54 30 30.

Le répertoire racine des applets est localisé en c:\reader\Applet.

Le fichier java source est stocké en c:\reader\Applet\Demo\DemoApp.java



Compilation et conversion.

- Cette opération est réalisée par le fichier *batch* compile.bat. Les fichiers produits sont DemoApp.cap, DemoApp.exp et une fichier d'archive DemoApp.jar.

```
set JC=c:\jc211
set JDK=c:\JDK

set JCBIN=%JC%\bin
set CLASSPATH=%JCBIN%\api21.jar

set PACK=Demo
set APPLI=DemoApp

REM  compilateur javac
%JDK%\bin\javac.exe -classpath %CLASSPATH% -g %APPLI%.java

set PATH=%PATH%;%JCBIN%
set PATH=%PATH%;%JDK%

REM Converter converter.jar
%JDK%\bin\java -classpath %JCBIN%\converter.jar;%JDK%\lib com.sun.javacard.converter.Converter -config
%APPLI%.opt

cd ..
%JDK%\bin\jar.exe cvf %PACK%\%APPLI%.jar %PACK%\%APPLI%.class
cd %PACK%
```

- Le fichier DemoApp.opt mémorise la liste des options nécessaires au *converter*.

```
-classdir .\..
-exportpath C:\jc211\api21
-applet 0x4a:0x54:0x45:0x53:0x54:0x30:0x30 DemoApp
-out CAP EXP
-nobanner
Demo 0x4a:0x54:0x45:0x53:0x54:0x30 1.0
```

Emulation.

- Le fichier de commande emulator.bat

```
set JC=c:\jc211
set JDK=c:\JDK
set APPLI=DemoApp
REM
set JCBIN=%JC%\bin
REM
REM Emulateur JavaCard
%JDK%\bin\java -classpath %JC%\bin\jcwde.jar;%JC%\bin\apduio.jar;%JC%\bin\api21.jar;.\%APPLI%.jar
com.sun.javacard.jcwde.Main -p 9025 .\%APPLI%.app
```

Le fichier DemoApp.app

```
//              Applet                                          AID
com.sun.javacard.installer.InstallerApplet    0xa0:0x0:0x0:0x0:0x62:0x3:0x1:0x8:0x1
Demo.DemoApp                                  0x4a:0x54:0x45:0x53:0x54:0x30:0x30
```

Test avec APDUtool.

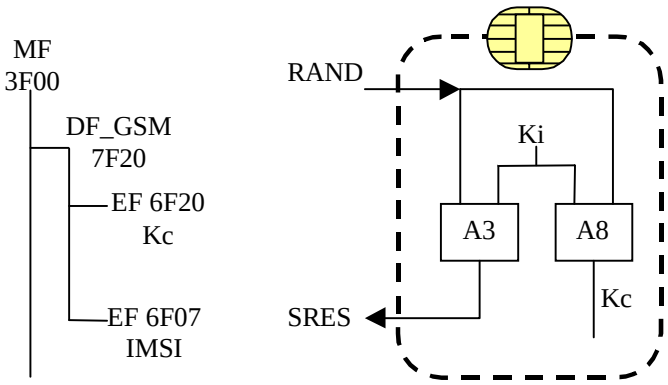
- Le fichier apdutool.bat

```
set JC=c:\jc211
set JDK=c:\JDK
set FILE=script
REM
set JC21BIN=%JC%\bin
REM
set PATH=%PATH%;%JC21BIN%
set PATH=%PATH%;%JDK%

REM path
REM set
%JDK%\bin\java -noverify -classpath %JC21BIN%\apdutool.jar;%JC21BIN%\apduio.jar;%JDK%\lib
com.sun.javacard.apdutool.Main -p 9025 .\%FILE%.scr
```

- Le fichier d'apdu script.rc

```
powerup;
powerup;

// Select the installer applet
0x00 0xA4 0x04 0x00 0x09 0xa0 0x00 0x00 0x00 0x62 0x03 0x01 0x08 0x01 0x7F;
// 90 00 = SW_NO_ERROR

// begin installer command
0x80 0xB0 0x00 0x00 0x00 0x7F;
// 90 00 = SW_NO_ERROR

// create DemoApp applet
0x80 0xB8 0x00 0x00 0x09 0x07 0x4a 0x54 0x45 0x53 0x54 0x30 0x30 0x00 0x7F;
//                              AID=0x4a:0x54:0x45:0x53:0x54:0x30:0x30
// 90 00 = SW_NO_ERROR

//End installer command
0x80 0xBA 0x00 0x00 0x00 0x7F;
// 90 00= SW_NO_ERROR

// Select DemoApp
0x00 0xa4 0x04 0x00 0x07 0x4a 0x54 0x45 0x53 0x54 0x30 0x30  0x7F;
// 90 00 = SW_NO_ERROR

// Write 4 bytes
0xBC 0xD0 0x00 0x00 0x04 0x12 0x34 0x56 0x78 0x7F ;
//90 00

// Read 4 bytes
 0xBC 0xB0 0x00 0x00 0x00 0x04 ;
// 12 34 56 78 90 00

// *** SCRIPT END ***
powerdown;
```

VIII Etude d'une carte SIM – ETSI GSM 11.11.



Une carte SIM réalise les fonctions suivantes,

- **Identification** et **authentification** d'un abonné.
- **Confidentialité** des données inchangées.
- **Sécurisation des accès aux fichiers** à l'aide d'un PIN code.

Elle intègre pour l'essentiel,

- Un identifiant d'abonné GSM (IMSI).
- Une clé secrète (Ki) associée aux algorithmes cryptographiques A3 et A8, qui permettent l'authentification d'un client.
- Un algorithme RUN_GSM_ALGORITHM() qui calcule une signature SRES et une clé de chiffrement Kc à partir d'un nombre aléatoire (RAND).
- Un système de fichier avec sous répertoire DF est 7F20, qui contient en particulier un fichier IMSI (EF_6F07), deux fichiers d'annuaire (*phonebook*) et un fichier stockant les messages SMS.
- 22 APDUs sont supportées.

Une carte comporte deux PIN codes (CHV1 et CHV2, **C**ard **H**older **V**erification Information) et deux codes (PUK) de déblocage associés UNBLOCK CHV1 et UNBLOCK CHV2.

Liste des APDUs disponibles.

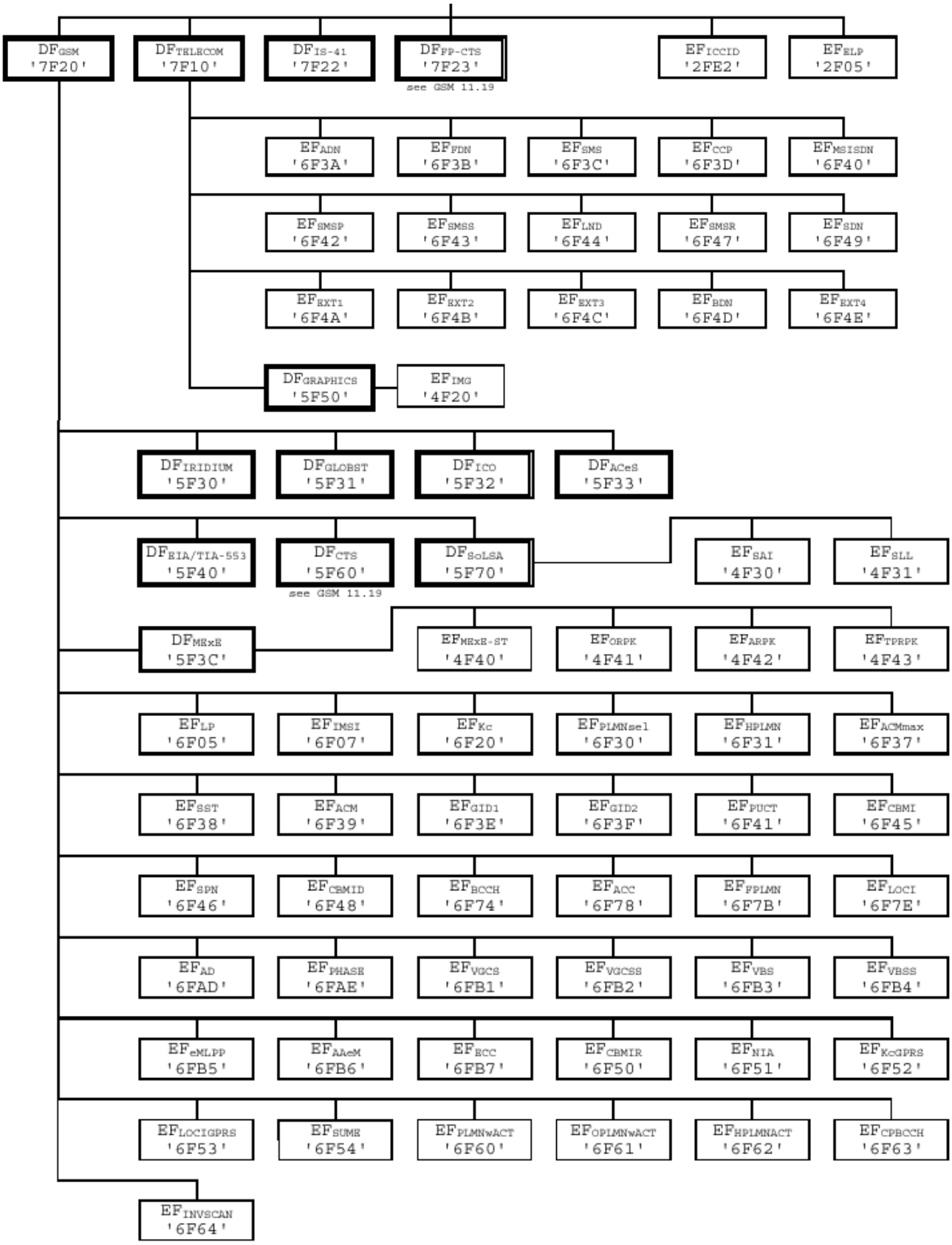
COMMAND	INS	P1	P2	P3	S/R
SELECT	'A4'	'00'	'00'	'02'	S/R
STATUS	'F2'	'00'	'00'	lgth	R
READ BINARY	'B0'	offset high	offset low	lgth	R
UPDATE BINARY	'D6'	offset high	offset low	lgth	S
READ RECORD	'B2'	rec No.	mode	lgth	R
UPDATE RECORD	'DC'	rec No.	mode	lgth	S
SEEK	'A2'	'00'	type/mode	lgth	S/R
INCREASE	'32'	'00'	'00'	'03'	S/R
VERIFY CHV	'20'	'00'	CHV No.	'08'	S
CHANGE CHV	'24'	'00'	CHV No.	'10'	S
DISABLE CHV	'26'	'00'	'01'	'08'	S
ENABLE CHV	'28'	'00'	'01'	'08'	S
UNBLOCK CHV	'2C'	'00'	see note	'10'	S
INVALIDATE	'04'	'00'	'00'	'00'	-
REHABILITATE	'44'	'00'	'00'	'00'	-
RUN GSM ALGORITHM	'88'	'00'	'00'	'10'	S/R
SLEEP	'FA'	'00'	'00'	'00'	-
GET RESPONSE	'C0'	'00'	'00'	lgth	R
TERMINAL PROFILE	'10'	'00'	'00'	lgth	S
ENVELOPE	'C2'	'00'	'00'	lgth	S/R
FETCH	'12'	'00'	'00'	lgth	R
TERMINAL RESPONSE	'14'	'00'	'00'	lgth	S

NOTE: If the UNBLOCK CHV command applies to CHV1 then P2 is coded '00'; if it applies to CHV2 then P2 is coded '02'.

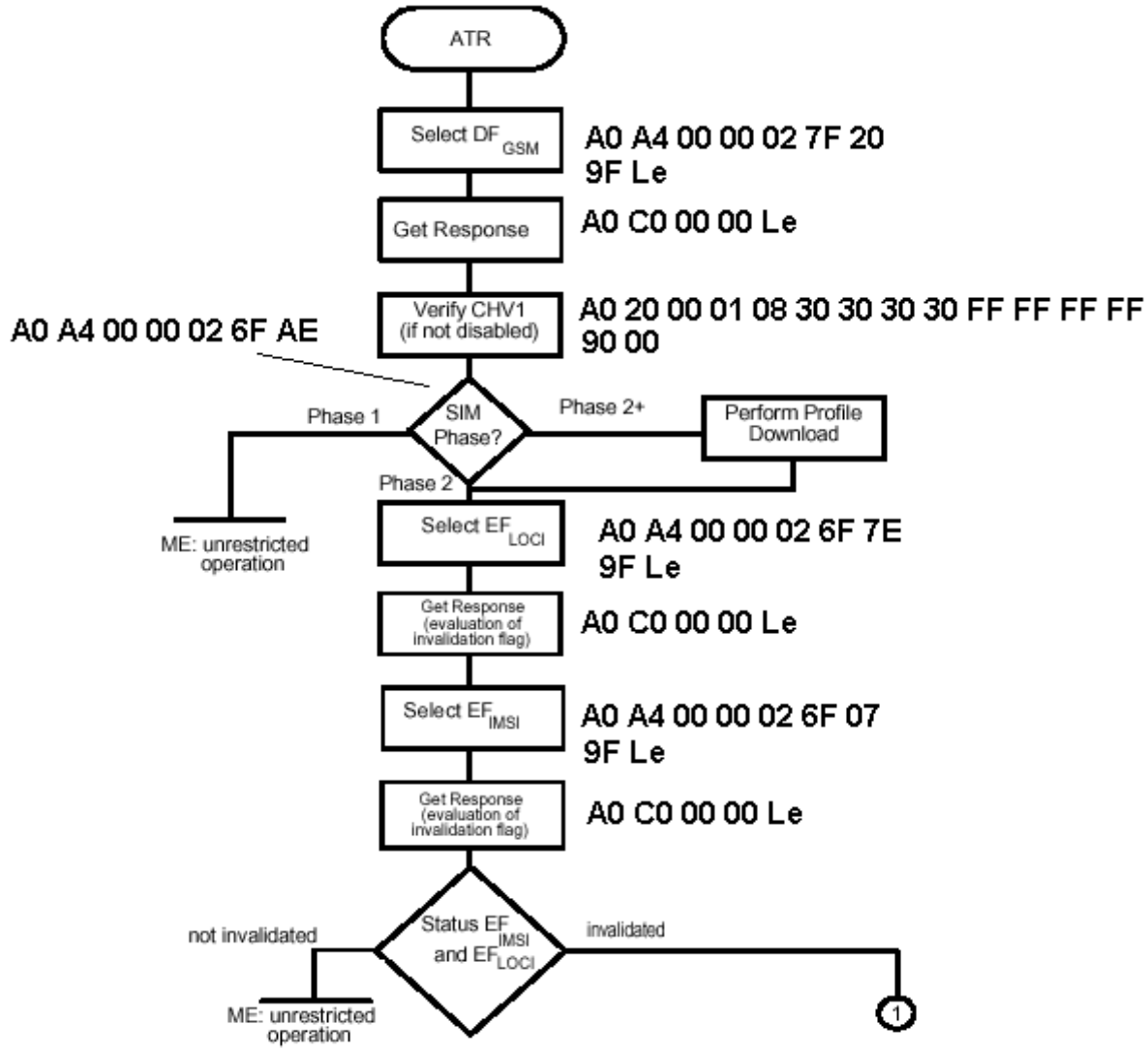
Mots de Status

SW1	SW2	Description
'90'	'00'	- normal ending of the command
'91'	'XX'	- normal ending of the command, with extra information from the proactive SIM containing a command for the ME. Length 'XX' of the response data
'9E'	'XX'	- length 'XX' of the response data given in case of a SIM data download error
'9F'	'XX'	- length 'XX' of the response data
'93'	'00'	- SIM Application Toolkit is busy. Command cannot be executed at present, further normal commands are allowed.
'92'	'0X'	- command successful but after using an internal update retry routine 'X' times
'92'	'40'	- memory problem
'94'	'00'	- no EF selected
'94'	'02'	- out of range (invalid address)
'94'	'04'	- file ID not found - pattern not found
'94'	'08'	- file is inconsistent with the command
'98'	'02'	- no CHV initialized
'98'	'04'	- access condition not fulfilled - unsuccessful CHV verification, at least one attempt left - unsuccessful UNBLOCK CHV verification, at least one attempt left - authentication failed (see note)
'98'	'08'	- in contradiction with CHV status
'98'	'10'	- in contradiction with invalidation status
'98'	'40'	- unsuccessful CHV verification, no attempt left - unsuccessful UNBLOCK CHV verification, no attempt left - CHV blocked - UNBLOCK CHV blocked
'98'	'50'	- increase cannot be performed, Max value reached
'67'	'XX'	- incorrect parameter P3 (see note)
'6B'	'XX[#]'	- incorrect parameter P1 or P2 (see [##])
'6D'	'XX[#]'	- unknown instruction code given in the command
'6E'	'XX[#]'	- wrong instruction class given in the command
'6F'	'XX[#]'	- technical problem with no diagnostic given
NOTE 1: [#] These values of 'XX' are specified by ISO/IEC; at present the default value 'XX'='00' is the only one defined. NOTE 2: [##] When the error in P1 or P2 is caused by the addressed record being out of range, then the return code '94 02' shall be used.		

Fichiers et Répertoires



Mise en œuvre.



Utilisation d'une carte SIM.

Sélection du sous répertoire GSM.

On obtient l'en tête du répertoire.

```
// Select DF_GSM (7F20)
```

A0 A4 00 00 02 7F 20

```
// 9F 17
A0 C0 00 00 17
// 00 00 RFU
// 00 00 Mémoire non allouée dans le sous répertoire (0 octets)
// 7F 20 File_ID (DF_GSM)
// 02 Type 01=MF 02=DF 04=EF
// 00 00 44 44 01 RFU
// 09 Longueur de la fin de l'en tête du répertoire = 9 octets
// 13 File Info, b8=1 CHV1 non requis CHV1 est nécessaire
// 00 Nombre de sous répertoires DF =0
// 11 Nombre de fichiers = 17
// 04 Nombre de codes = 4
```

```
// 00 RFU
// 83 CHV1 Statut*, Code initialisé  3 essais.
// 8A UNBLOCK CHV,  Code initialisé 10 essais.
// 81 CHV2 statut,  Code initialisé  1 essai.
// 8A UNBLOCK CHV2, Code initialisé 10 essais.
// 00 Réservé.
// 90 00
* b8=1 Code initialisé, b7b6 RFU, b4b3b2b1=nombre d'essais restant.
```

Présentation du PIN code.

```
// Verify P2=1=CHV1 PIN ='0000'
A0 20 00 01 08 30 30 30 30 FF FF FF FF
// 90 00
```

Lecture du fichier EF_PHASE

```
// Select EF_Phase
```

A0 A4 00 00 02 6F AE

```
// 9F 0F
A0 C0 00 00 0F
// 00 00 00 01 6F AE 04 00 04 FF 44 01 01 00 00 90 00
A0 B0 00 00 01
// 02 90 00 2=Phase2 (3=Phase2+)
```

Lecture IMSI

```
// Select IMSI (6F7E)
A0 A4 00 00 02 6F 7E
// 9F 0F
A0 C0 00 00 0F
// 00 00 RFU
// 00 09 longueur du fichier = 9 octets.
// 6F 07 File_ID
// 04 file Type 01=MF 02=DF 04=EF
// 00 b7=1 Commande INCREASE supporté.
// 14 FF 14 *octets 9,10,11, Lecture CHV1, Mise à jour ADM.
// 01 Statut du fichier ** non invalidé.
// 01 taille d'un enregistrement.
// 00 structure 00=Transparent, 01=Linear Fixed, 03=Cyclic.
// 00 ?
// 90 00

*Conditions d'accès 0=ALWAY 1=CHV1 2=CHV2 3=RFU 4…E=ADM F=NEW
                  b8b7b6b5   | b4b3b2b1
     octet 9  = READ;SEEK    | UPDATE
     octet 10 = INCREASE     | RFU
     octet 11 = REHABILTATE  | INVALIDATE
** Statut du fichier
     b0=1 non invalidé
     b3=1, lecture et mise à jour possible si le fichier est
     invalidé
     b3=0, lecture et mise à jour interdite si le fichier est
     invalidé.
     b2,b4,b4,b6,b7,b8 RFU
```

```
// Read Binary
A0 B0 00 00 09
// 08 29 80 01 44 72 65 20 61
// 90 00
```

Run_Gsm_algo.

```
// Run_Gsm_Algo(RAND)
A0 88 00 00 10 01 23 45 67 89 AB CD EF 01 23 45 67 89 AB CD EF
// 9F 0C
A0 C0 00 00 0C
// FE 67 7C 9D              SRES (4 octets)
// B8 DD F1 B1 DE 27 18 00   KC (8 octets)
// 90 00
```

Accès à la Table des Services.

Cette table contient la liste des services disponibles sur la carte SIM. 2 bits sont alloués par service. Les bits de plus faibles poids sont assignés aux services dont les numéros sont les plus petits.

4p+ 4 4p+ 3 4p+ 2 4p+ 1

Un octet = b8b7 b6b5 b4b3 b2b1 du fichier EF_SST code quatre services, compris entre 4p+1 et 4p+4.

bi = service disponible, bi+1= service activé.

```
// SELECT EF_SST Sim_Service_Table
A0 A4 00 00 02 6F 38
// 9F 0F
A0 C0 00 00 0F
// 00 00 00 05 6F 38 04 00 14 FF 44 01 01 00 00 90 00
```

A0 B0 00 00 05

```
// FF 3C FF 7F 3D 90 00
// Service 1 CHV1 Disable Function
// Service 2  ADN Abreviated Dialling Numbers, annuaire
// Service 3  FDN Fixed Dialling      Numbers, annuaire
// Service 4  SMS Short Message       Storage, message SMS
```

Répertoire DF Telecom.

```
// Select DF_Telecom 7F10
A0 A4 00 00 02 7F 10
// 9F 17
A0 C0 00 00 17
// 00 00 1A 95 7F 10 02 00 00 44 44 01 09 13 00 0D
// 04 00 83 8A 81 8A 00 90 00
6805 octets et 13 fichiers.
```

Messages SMS.

```
// Select EF SMS 6F3C
A0 A4 00 00 02 6F 3C
// 9F 0F
```

A0 C0 00 00 0F

```
// 00 00 08 F0 6F 3C 04 00 11 F4 44 01 02 01 B0 90 00
Taille totale = 08F0 = 2280 octets
```

```
Taille d'un enregistrement = B0 = 176 octets.
2280/176 = 13 Enregistrements

// Read Record
// P1 00= Current Record
// P2 02=Next 03=Previous 04=Absolute
A0 B2 01 04 B0
// 03  b1=used/free b2= from_network/to_be_read/read b3=to_be_sent
// 07 91 33 06 09 20 92 F0 04 00 91 41 00 10 70
// 21 11 10 25 00 30 D2 32 FC ED 26 97 EB 72 1D 28
// 06 72 BF EB F6 72 B8 8E C2 A7 40 ED F2 7C 1E 3E
// 97 51 F3 94 0B 24 0D C3 E1 65 76 59 0F 8A C9 66
// FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
//…tout à FF
// 90 00

// Update Record #10
A0 DC 0A 04 B0 01 23 45 67 89 FF FF FF FF FF FF FF FF FF FF FF FF  FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
Tout à FF
// 90 00

// Select EF_ADN (6F3A) - Abreviated Dialing Number
A0 A4 00 00 02 6F 3A
// 9F 0F
A0 C0 00 00 0F
// 00 00 0A A0 6F 3A 04 00 11 F4 22 01 02 01 22 90 00
// 2720 octets/34 = 80 records

// Read Record #1
A0 B2 01 04 22
// 50 41 53 43 41 4C FF FF FF FF FF FF FF FF FF FF FF FF FF FF
// 06 BCD Number Length = 1 + 5
// 81 TON & NPI = GSM
// 60 28 75 73 83 FF FF FF FF FF FF FF // 06 82 57 37 38
// 90 00

// Select EF_FDN (6F3B) Fixed Dialing Number
```

A0 A4 00 00 02 6F 3B

```
// 9F 0F
A0 C0 00 00 0F
// 00 00 02 A8 6F 3B 04 00 12 F4 44 01 02 01 22 90 00
// 680 /34 = 20 records

// Read Record #1
A0 B2 01 04 22
// 52 05 70 6F 6E 64 65 75 72 FF FF FF FF FF FF FF FF FF FF FF FF
// X Alpha identifier
// 03 BCD Length = 1 + 2
// 81 TON & NPI
// 21 F3 FF FF FF FF FF FF FF FF // SSC string = 123
// FF
// FF
// 90 00
```

IX - Le modèle Application Sim Tool Kit (STK)

Introduction

Ce modèle est décrit par les deux documents suivants :

- "*Digital cellular telecommunications system (Phase 2+);Specification of the SIM Application Toolkit for the Subscriber Identity Module - Mobile Equipment (SIM-ME) interface)*", ETSI TS 101 267 - 3GPP TS 11.14
- "*Digital cellular telecommunications system (Phase 2+);GSM API for SIM toolkit stage*" , ETSI TS 101 476 - 3GPP TS 03.19

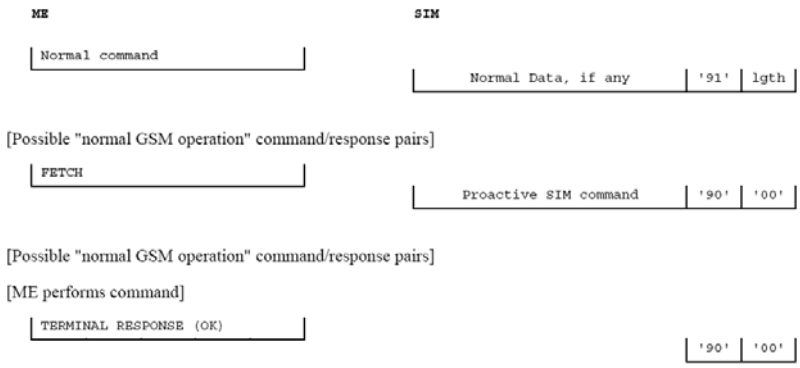
Commandes proactives

On désigne par commande proactive (*Proactive Command*) une action contrôlée par la carte SIM mais exécutée par le mobile. On dresse ci-dessous une liste non exhaustive de commandes proactives.

- **DISPLAY TEXT**, affiche un texte sur l'écran du mobile.
- **GET INKEY**, affiche un texte sur l'écran du mobile et attend la frappe d'un caractère en retour
- **GET INPUT**, affiche un texte sur l'écran du mobile et attend une réponse en retour.
- **GET READER STATUS**, délivre de l'information pour un lecteur de carte additionnel disponible sur le mobile.
- **MORE TIME**, la carte SIM indique qu'elle a besoin de plus de temps pour traiter une requête précédente. Le terminal acquitte cette notification par une commande TERMINAL RESPONSE (OK).
- **PERFORM CARD APDU**, la carte SIM demande au terminal de router une commande APDU vers un lecteur de carte additionnel.
- **PLAY TONE**, le mobile diffuse une sonnerie ou tout signal sonore approprié.
- **POLL INTERVAL**, indique l'intervalle de temps entre les commandes STATUS produites par le mobile durant les phases inactives (IDDLE mode) de la carte SIM. Le mode POLLING est désactivé par la commande POLLING OFF.
- **POWER OFF CARD**, termine une session avec une carte externe.
- **POWER ON CARD**, démarre une session avec une carte externe et renvoie l'ATR de cette dernière.
- **PROVIDE LOCAL INFORMATION,** le mobile renvoie des informations relatives au pays et réseau dans lequel il fonctionne.
- **REFRESH**, demande d'initialisation de la carte SIM et/ou indication que le contenu de fichiers EF a été modifié.
- **RUN AT COMMAND**, envoi d'une commande AT au mobile et attente d'une réponse AT.
- **SELECT ITEM**, la SIM a précédemment fourni une liste d'options (ITEMs) et attends le choix de l'utilisateur.
- **SEND DTMF,** envoi d'un signal DMTF durant en phase d'établissement d'appel.
- **SEND SHORT MESSAGE**, envoi d'une message SMS vers le réseau.
- **SEND USSD**, envoi d'une chaîne USSD vers le réseau.
- **SET UP CALL**, prise en charge d'un appel téléphonique.
- **SET UP EVENT LIST,** la SIM fournit une liste d'évènements auxquels il souscrit.
- **SET UP IDLE MODE TEXT**, une chaîne de caractère utilisé par le mobile en mode en mode "stand-by mode text. »
- **SET UP MENU**, une liste de choix est incorporée dans la structure des menus du mobile.

- **TIMER MANAGEMENT**, demande de gestion d'une horloge (démarrage, arrêt, et lecture de la valeur courante)

La figure suivante illustre le mécanisme de base d'une commande proactive. La SIM exécute une requête du terminal et indique par le mot STATUS « 91 lgth » qu'une commande de longueur *lgth* est disponible. Le terminal collecte cette commande grâce à l'instruction FETCH. Il exécute cette dernière et notifie sa réponse par le message TERMINAL RESPONSE.



Un dialogue proactif GSM 11.14 typique.

COMMAND	INS	P1	P2	P3	S/R
SELECT	'A4'	'00'	'00'	'02'	S/R
STATUS	'F2'	'00'	'00'	lgth	R
READ BINARY	'B0'	offset high	offset low	lgth	R
UPDATE BINARY	'D6'	offset high	offset low	lgth	S
READ RECORD	'B2'	rec No.	mode	lgth	R
UPDATE RECORD	'DC'	rec No.	mode	lgth	S
SEEK	'A2'	'00'	type/mode	lgth	S/R
INCREASE	'32'	'00'	'00'	'03'	S/R
VERIFY CHV	'20'	'00'	CHV No.	'08'	S
CHANGE CHV	'24'	'00'	CHV No.	'10'	S
DISABLE CHV	'26'	'00'	'01'	'08'	S
ENABLE CHV	'28'	'00'	'01'	'08'	S
UNBLOCK CHV	'2C'	'00'	see note	'10'	S
INVALIDATE	'04'	'00'	'00'	'00'	-
REHABILITATE	'44'	'00'	'00'	'00'	-
RUN GSM ALGORITHM	'88'	'00'	'00'	'10'	S/R
SLEEP	'FA'	'00'	'00'	'00'	-
GET RESPONSE	'C0'	'00'	'00'	lgth	R
TERMINAL PROFILE	'10'	'00'	'00'	lgth	S
ENVELOPE	'C2'	'00'	'00'	lgth	S/R
FETCH	'12'	'00'	'00'	lgth	R
TERMINAL RESPONSE	'14'	'00'	'00'	lgth	S

NOTE: If the UNBLOCK CHV command applies to CHV1 then P2 is coded '00'; if it applies to CHV2 then P2 is coded '02'.

Résumé des commandes SIM dans la norme GSM 11.11

Les évènements

Les évènements notifiés par le terminal à la carte SIM (réception d'un SMS, choix d'un élément de menu, appel téléphonique entrant…) sont transportés par des APDUs de type ENVELOPE (*INS=C2*). Le premier TAG contenu dans cette commande identifie la nature de l'évènement.

Le contenu du message ENVELOPE est une liste d'objets encodés au format ASN.1, c'est à dire sous une forme qualifiée de TLV,
- un TAG (T), un octet qui identifie l'objet ;
- la longueur (L) de la valeur ;
- et le contenu (V) de cet objet.

Le terminal indique à la SIM la liste des services STK (commandes proactives, évènements, …) qu'il supporte à l'aide de la commande TERMINAL PROFILE qui contient une liste d'octets, chaque bit mis à un indiquant la disponibilité d'un service particulier.

Par exemple A0 10 00 00 04 EF FF FF FF exprime la disponibilité de 31 services (le bit de poids fort doit être mis à 0).

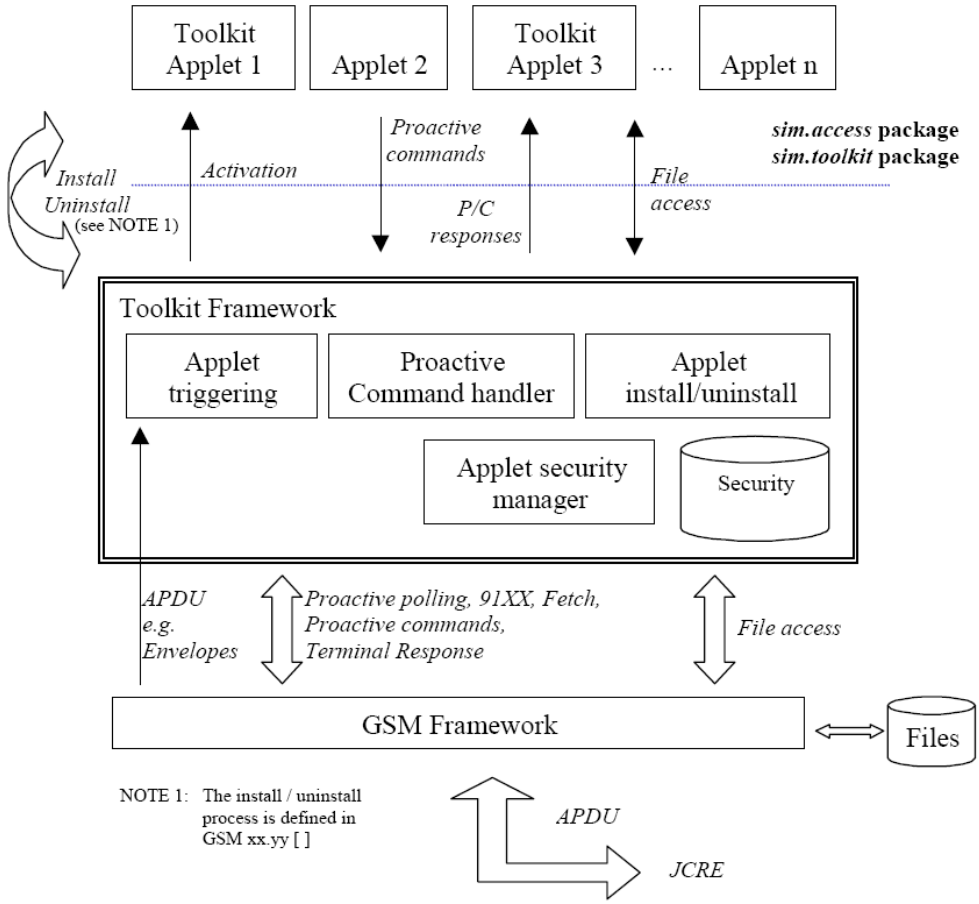
Exemple de dialogue STK

```
// Select DF GSM (7F20)
A0 A4 00 00 02 7F 20
// 9F 23
A0 C0 00 00 23
// 90 00

// Unblock CHV1
// A0 2C 00 00 10 37 38 39 38 31 32 31 34 31 31 31 31 FF FF FF FF

// Verify CHV1 "1111"
A0 20 00 01 08 31 31 31 31 FF FF FF FF
// 90 00

// Select EF_Phase
A0 A4 00 00 02 6F AE
// 9F 13
A0 C0 00 00 13
// 00 00 00 01 6F AE 04 00 04 FF 44 01 01 00 00 90 00

A0 B0 00 00 01
// 02 90 00   Phase_2   03<=> phase 2 and PROFILE DOWNLOAD REQUIRED (2+)

// Terminal Profile
A0 10 00 00 04 EF FF FF FF
// 90 00

// 11.14: When the ME receives a Short Message with:
// protocol identifier = SIM data download, and
// data coding scheme = class 2 message,
// then the ME shall pass the message transparently to the SIM
// using the ENVELOPE (SMS-PP DOWNLOAD)


// 11.14 ENVELOPE (INS=C2) SMS-PP download TAG=D1, Length  = 1D
// OBJECT Device-identity TAG=02, length=02
// - Source      (1 octet): 82 (ME) 83 (Network)
// - Destination (1 octet): 81 (SIM)
// OBJECT SMS-TPDU TAG=8B (ou 0B), Length=17
// - Message SMS (GSM 03.40)

A0C20000 1F D1 1D 82 02 83 81 8B 17 04 00 A1 7F F6 99 01 01 01 02 03 40 0A
01 14 00 06 0D 20 00 00 00 00
// 91 76
```

```
// FETCH
A0 12 00 00 76
D0 68 // Proactive SIM command TAG=D0
81 03 01 13 00
// OBJECT Command-details TAG=81 Length=03
// - Command-Number=01
// - Type-of-command=13=SEND-SHORT-MESSAGE  Command-Qualifier=0
// 82 02 81 83
// OBJECT Device-identity TAG=02, length 02,
// - source=82=SIM
// - destination=83=Network
0B 5D // OBJECT SMS-TPDU TAG=0B, Length = 5D
//message SMS(GSM 03.40)
01 A5 0B
91 21 43 65 87 90 F8 00 04 50 02 14 00 50 0D 80
00 00 00 46 16 03 01 00 41 01 00 00 3D 03 01 3F
AA 2B 6A 08 BD D2 85 B4 3D 1F 3B C9 71 5F C9 F8
5F C4 53 FE 58 F3 A9 E0 7F F3 97 CD 65 39 22 00
00 16 00 04 00 05 00 0A 00 09 00 64 00 62 00 03
00 06 00 13 00 12 00 63 01 00
90 00

// Terminal Response
A0 14 00 00 0C 81 03 01 13 01 82 02 82 81 83 01 00
// OBJECT Command-details TAG=81
// OBJECT Device-identity TAG=82
// OBJECT Command-result  TAG=83 00=OK
// 9000
```

Paquetages JAVA

La norme GSM 3.19 introduit la notion de *Toolkit Applet* dont les services sont hérités des paquetages *sim.toolkit* et *sim.access*. Le premier permet de s'enregistrer à des évènements particuliers et de produire des commandes proactives, le deuxième offre des facilités pour accéder au système de fichier de la carte SIM depuis un applet embarqué JAVA.



SIM ACCESS

L'accès aux services de sim.access s'effectue à l'aide d'un objet JAVA SIMView dont une instance est obtenue par le constructeur de l'applet embarqué

SIMView myView= SIMSystem.getTheSIMView();

La classe SIMView réalise des opérations classiques (sélection, lecture, écriture) avec le système de fichier de la SIM, telles que par exemple

- *select(short, byte[], short, short)*, sélectionne au répertoire DF ou un fichier EF et retourne son entête FCI
- readBinary(short, byte[], short, short), lecture des données d'un fichier linéaire
- readRecord(short, byte, short, byte[], short, short) lecture d'un fichier à enregistrements
- *updateBinary(short, byte[], short, short)* écriture dans un fichier linéaire
- *updateRecord(short, byte, short, byte[], short, short)* écriture dans un fichier à enregistrement

```
public class MyApplet extends Applet {

private SIMView theGsmApplet;
private byte[] buffer;

public MyApplet () {
// get a reference to the GSM interface
theGsmApplet = SIMSystem.getTheSIMView();
// create the exchange buffer
buffer = new byte[32];
}

public static void install(byte bArray[],short bOffset, byte bLength)
throws ISOException {
// create and register the applet
MyApplet myAppletRef = new MyApplet();
myAppletRef.register();
}

public void getADN(short adnNumber) {
// select EF ADN in DF GSM
theGsmApplet.select((short)SIMView.FID_DF_TELECOM);
theGsmApplet.select((short)SIMView.FID_EF_ADN);
// reads the record from EF ADN and put it in the exchange buffer
theGsmApplet.readRecord((short)adnNumber,
                        (byte)SIMView.REC_ACC_MODE_ABSOLUTE_CURRENT,
                        (short)0,
                        (byte[])buffer,
                        (short)0,
                        (short)32);
    }
 }
```

SIM Tool Kit

Un applet SIM Tool Kit implémente une interface *ToolkitInterface*, c'est-à-dire qu'il contient obligatoirement une méthode

public void processToolkit(byte event)

qui réalise le traitement des événements notifiés par le terminal à la SIM au moyen d'APDUs ENVELOPE.

Le constructeur d'un tel applet s'enregistre pour les événements qu'il désire traiter à l'aide d'un objet *ToolkitRegistry*,

```
ToolkitRegistry reg  = ToolkitRegistry.getEntry();
// register to the EVENT_UNFORMATTED_SMS_PP_ENV
reg.setEvent(EVENT_UNFORMATTED_SMS_PP_ENV);
```

Dès lors les événements sollicités seront routés par le système d'exploitation de la SIM vers la méthode processToolkit(*byte event*).

```
ProactiveHandler         myProHdlr  = null;
EnvelopeHandler          myEnvHdlr  = null;
ProactiveResponseHandler myRespHdlr = null;

public void processToolkit( byte event) throws ToolkitException
{ short offset, len;
myProHdlr = ProactiveHandler.getTheHandler();
myEnvHdlr = EnvelopeHandler.getTheHandler() ;

switch (event) {
//Proactive Commands For Your Tree Starts Here
case EVENT_UNFORMATTED_SMS_PP_ENV:

// On recherche la première occurrence du tag TAG_SMS_TPDU dans ENVELOPE
if (myEnvHdlr.findTLV(TAG_SMS_TPDU ,(byte)1) != TLV_NOT_FOUND)
{// Recopie de la valeur (le SMS) dans un tableau data à l'index 0
myEnvHdlr.copyValue((short)0,
                     data,(short)0,
                    (short) myEnvHdlr.getValueLength();
}

// offset = position des données du SMS
offset = (short) myEnvHdlr.getTPUDLOffset();
len    = Util.makeShort((byte)0,myEnvHdlr.getValueByte((short)offset));
// Recopie des données du SMS dans un tableau data
myEnvHdlr.copyValue(offset,data,(short)0,(short)(len+1)) ;

// Initialize 3 octets contenus dans les tags 81 et 82
// public void init(byte type,byte qualifier,byte dstdevice)

myProHdlr.init((byte)0x13,(byte)0x00,(byte)x83);

// Rajoute le TAG_SMS et un message SMS
myProHdlr.appendTLV(TAG_SMS_TPDU,sms,
                   (byte)0,(byte)sms.length);


// result value in the TERMINAL-RESPONSE APDU
byte result = myProHdlr.send();

// Pour obtenir les TAGs présents dans la réponse
myRespHdlr  = ProactiveResponseHandler.getTheHandler();


break;

default: //Add Your Default Action Here
         break;
}
}
```

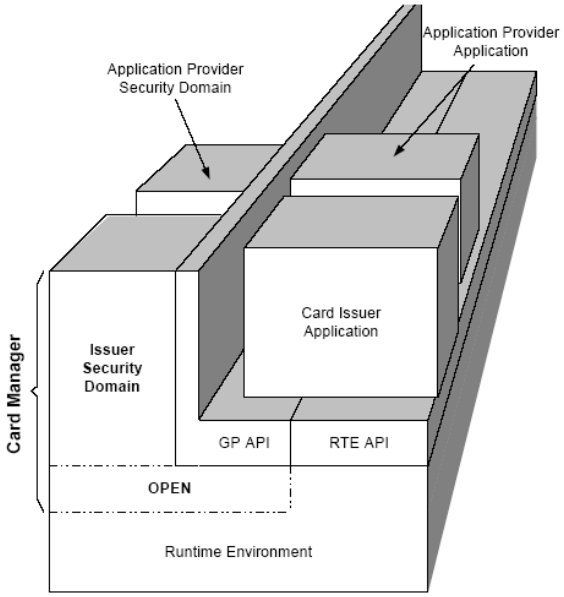
X - **Global Platform**

Introduction

L'architecture Global Platform comportent des composants qui réalisent une interface entre les applications et un système d'administration externe (ou *off-card management*), indépendante de la plateforme matérielle utilisée.

Les applications embarquées disposent d'un environnement d'exécution (*Runtime Environment*) qui dispose d'un jeu d'APIs (*Application Programming Interface*) permettant d'utiliser les services Global Platform.

Le *Card Manager* est l'entité logicielle centrale dans un composant (carte à puce) GP. Cette dernière gère des clés cryptographiques spécifiques à l'émetteur de la carte (*Card Issuer*) et aux fournisseurs d'applications (*Application Provider*)



Runtime Environment

On désigne par *Runtime Environment* le contexte logique d'une application exécutée sous le contrôle d'un système d'exploitation, qui gère simultanément plusieurs entités logicielles (autrement dit des programmes). Les services GP sont rendus accessibles à l'aide d'API (GP API) spécialisées

Card Manager.

Cette entité se divise en trois sous ensembles fonctionnels, l'environnement Global Platform (*OPEN*), le domaine de l'émetteur de la carte (*Issuer Domain*), les méthodes de vérification du porteur de la carte (*Card Holder Verification Methods*).

GlobalPlatform Environment

Le *Card Manager* réalise un ensemble de services dénommé *GlobalPlatform Environment* (OPEN), qui offre les fonctionnalités suivantes
- Gestion des APIs utilisées par les applications
- Traitement des commandes, c'est-à-dire routage des APDUs
- Gestion des canaux logiques, assurant la confidentialité et de l'intégrité des informations échangées.
- Gestion du contenu de la carte

Issuer Security Domain

C'est une entité logique qui assure les droits de l'émetteur de la carte et qui contrôle les mécanismes de chargement, d'installation, et de destruction des applications embarquées.

Cardholder Verification Management

Ce sous ensemble vérifie l'identité du porteur de la carte, typiquement à l'aide d'un PIN code.

Security Domains

Un domaine de sécurité est une application embarquée qui administre le cycle de vie d'un ensemble d'applications (chargement – installation – destruction). Il est associé à un AID (Application IDentifier) et possède également un cycle de vie. Une telle entité assure des fonctions de sécurité telles que le stockage de clés cryptographiques, le chiffrement et le déchiffrement, la génération et la vérification de signatures.

Le domaine de sécurité de l'émetteur de la carte (*Issuer Security Domain*) contrôle le chargement des applications *Issuer*. Il permet également d'installer d'autres domaines (*Application Provider Security Domains*) utilisés pour contrôler le cycle de vie des applications de type *Provider*.

Un domaine de sécurité est associé à un jeu d'APDUs qui définit de manière concrète les services supportés.

GlobalPlatform API

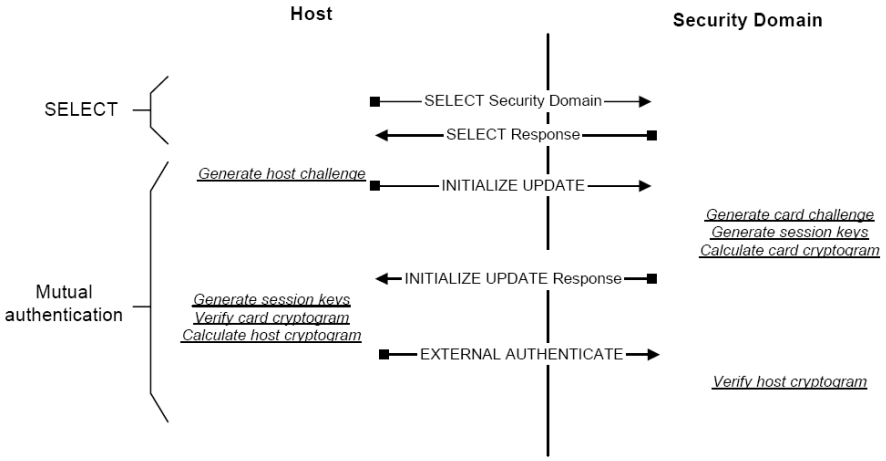
Cette interface délivre des services GP aux applications, par exemple la vérification du porteur de carte, des fonctions de sécurité, ou la gestion du cycle de vie.

Card Content

On entend par contenu de la carte un fichier chargé et exécutable. L'opération d'installation consiste à créer dans la mémoire du composant une instance de cette application. Il est également possible de détruire ultérieurement cette instance ou le fichier initialement chargé.

Sécurité des communications

La sécurité des communications entre un système hôte (un terminal qui utilise une carte) et un domaine de sécurité s'appuie sur trois procédures
- Une mutuelle authentification entre le système hôte et le domaine de sécurité
- Un mécanisme d'intégrité et de signature des messages échangés
- La confidentialité (c.a.d le chiffrement) des données



Une clé DEK (*Data Encryption Key*) pré-installée permet de transporter de manière chiffrée des données sensibles (secret partagé ou clé RSA)

Deux clés S-ENC(*Secure Channel Encryption Key*) et S-MAC (*Secure Channel Message Authentication Code Key*) assurent respectivement la confidentialité et l'intégrité des messages transportés par le canal sécurisé (*Secure Channel*). Elles sont crées dynamiquement lors de la procédure d'authentification.

Les clés de sessions sont calculées à partir d'une clé dénommée *Secure Channel base key* (de 16 octets) qui s'applique à une algorithme triple DES en mode CBC.

Un chiffrement DES s'applique à la partie données (après P3) de la commande APDU. Un premier octet 80 est ajouté aux données, puis une série d'octets nuls (00) est rajoutée afin d'obtenir une taille multiple de 8.

Le MAC d'une commande APDU est produit en ajoutant la longueur du MAC (soit 8 octets) au champ P3 (Lc) de la commande. L'octet 80 est ajouté à la fin de la commande puis un ensemble d'octets nul (00) est rajouté de telle sorte que la longueur totale soit un multiple de 8. Le MAC est calculé sur cette suite d'octets.

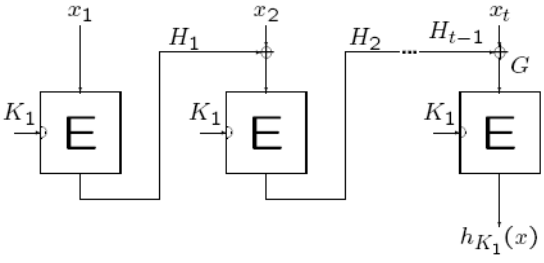
Protocole 01

Dans le cas de la carte un bloc B16 de 16 octets est construit par concaténation du host-challenge (8 octets) et du card-challenge (8 octets)

Dans le cas du système hôte carte un bloc B16 de 16 octets est construit par concaténation du card-challenge (8 octets) et du host-challenge (8 octets).

Un bloc de 8 octets ('80 00 00 00 00 00 00 00') est ajouté au bloc précédent (B16), ce qui conduit à un bloc (B24) de 24 octets.

Un algorithme triple DES-CBC utilisant la clé S_ENC avec un IV de 8 octets nul est appliqué à la valeur B24 pour produire une signature selon la méthode ISO 9797-1 (MAC Algorithm 1 with output transformation 1).



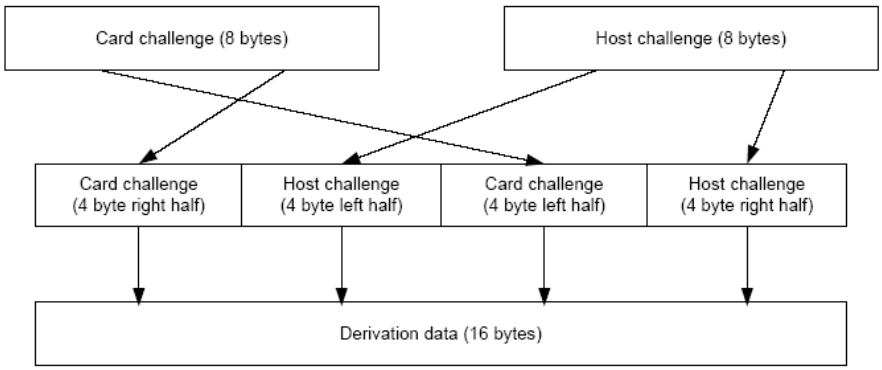


Figure D-3: Session Key - Step 1 - Generate Derivation data

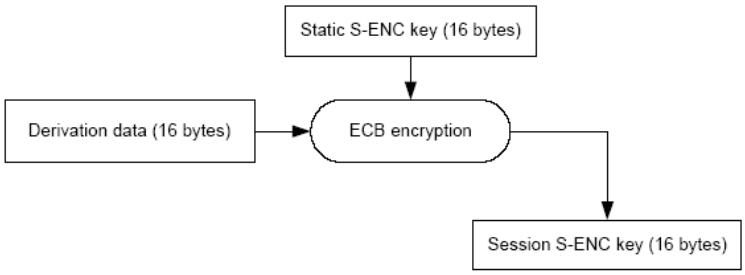


Figure D-4: Session Key - Step 2 - Create S-ENC Session Key

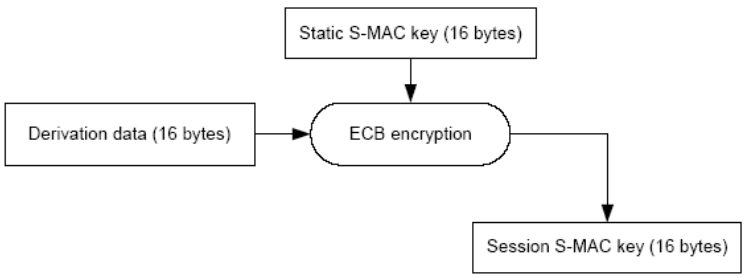


Figure D-5: Session Key – Step 3 - Create S-MAC Session Key

Protocole 02

Dans le cas du protocole 02 les deux premiers octets du card-challenge forment un champ nommé Sequence-Counter. Ce paramètre est utilisé pour le calcul de différentes clés. MAC.

La clé R-MAC s'applique aux messages produits par le système hôte
La clé C-MAC s'applique aux messages produits par la carte

Les cryptogrammes sont calculés de manière identique au cas du protocole 01

Clé	Constante
C-MAC	0101
R-MAC	0102
S-ENC	0182
DEK	0181

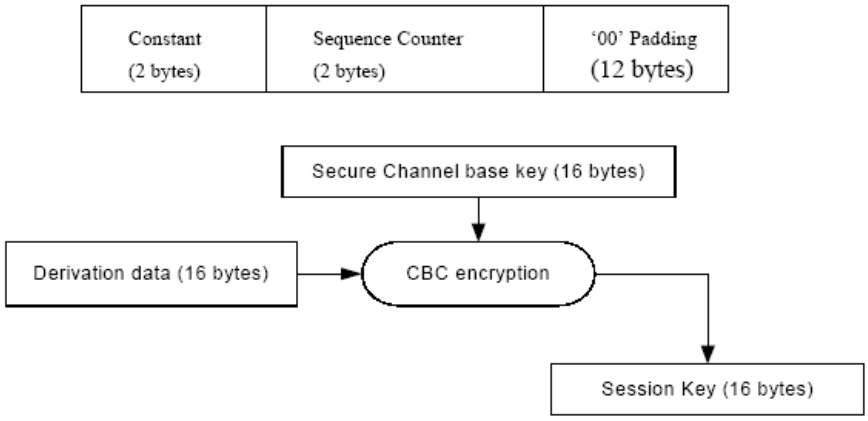


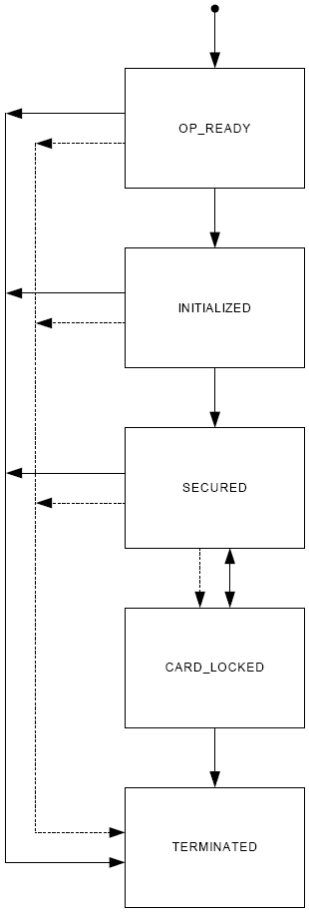
Figure E-2: Create Secure Channel Session Key from the Base Key

Le cycle de vie de la carte

Le cycle de vie d'une carte se divise en cinq états

1. OP_READY
2. INITIALIZED
3. SECURED
4. CARD_LOCKED
5. TERMINATED

Les deux premiers états (OP_READY et INITIALIZED) sont actifs lors des phases dites de pré émission de la carte. Les états restant seront observés lors des phases qualifiées de post-émission.



L'état OP_READY indique que l'environnement d'exécution et le domaine de sécurité de l'émetteur (*issuer*) sont disponibles. Ce dernier peut recevoir, exécuter, et répondre aux commandes APDUs générées par le système hôte.

L'état INITIALIZED est actif lors de la production de la carte. Il permet de transférer des données initiales dans la carte (les clés du domaine de sécurité de l'émetteur par exemple). La transition de l'état OP_READY à INITIALIZED est irréversible.

Dans l'état SECURED, l'entité *Card Manager* contrôle la politique de sécurité en phase de post émission (par exemple le chargement, installation, activation de domaines de sécurité ou d'applications, …). La transition de l'état INITIALIZED à SECURED est irréversible.

Dans l'état CARD_LOCKED, la carte est contrôlée uniquement par le domaine de sécurité de l'émetteur. La transition entre l'état SECURED et CARD_LOCKED est réversible.

L'état TERMINATED, signifie la fin du Cycle de Vie de la carte. La transition à cet état peut s'effectuer à partir de n'importe quel état mais est irréversible.

Cycle de vie d'une application

Une application comporte trois états :

1. INSTALLED
2. SELECTABLE
3. LOCKED

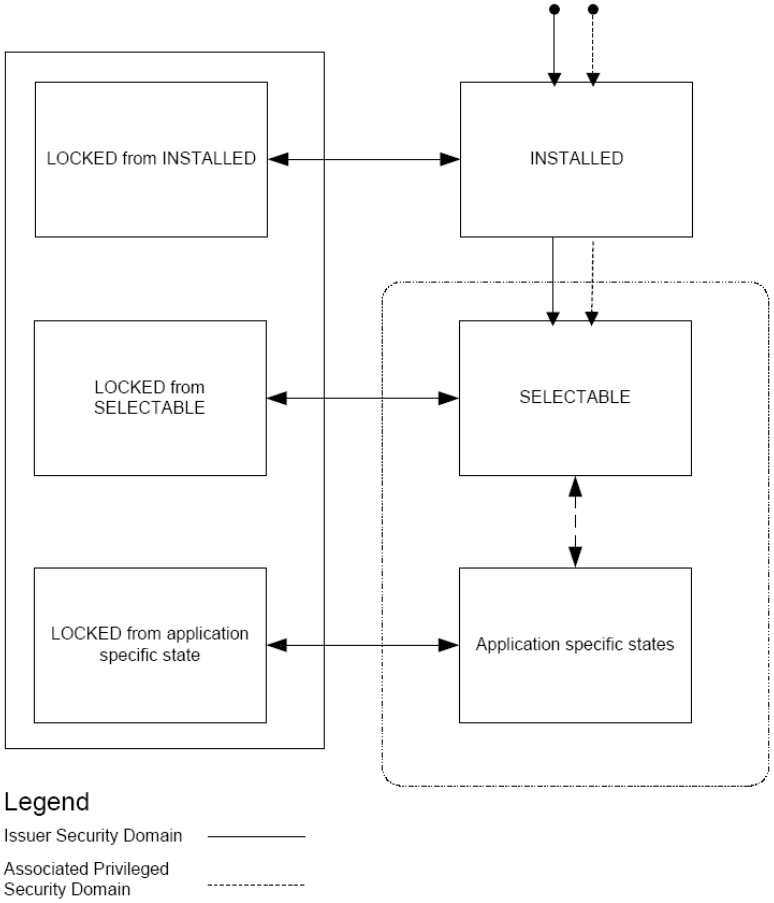
Cependant, dans l'état SELECTABLE une application peut gérer un sous état propre, qualifié de SPECIFY.

Dans l'état INSTALLED, le code et les données de l'application sont chargés en mémoire.

L'état SELECTABLE indique que l'application est prête à recevoir des commandes (APDUs) du système hôte; elle est responsable de la gestion de son cycle de vie. La transition de l'état INSTALLED à SELECTABLE est irréversible.

L'état LOCKED interdit la sélection et l'exécution de l'application. La transition vers l'état LOCKED est réversible et contrôlé par le domaine de sécurité l'émetteur.

L'entité OPEN peut détruire une application, quelque soit son état.



Cycle de vie d'un domaine de sécurité

Le cycle de vie d'un domaine de sécurité se divise en quatre états.

1. INSTALLED
2. SELECTABLE
3. PERSONALIZED
4. LOCKED

Contrairement au cas des applications, il n'existe pas d'états propres , et donc non normalisé, du domaine de sécurité.

L'état INSTALLED indique que le domaine de sécurité est accessible depuis une entité authentifiée.

Dans l'état SELECTABLE le domaine de sécurité reçoit typiquement ses clés cryptographiques. Il ne peut pas être sélectionné, et n'est associé à aucune application. La transition de l'état INSTALLED à SELECTABLE est irréversible.

L'état PERSONALIZED, signifie que le domaine de sécurité possède toutes les données (clés..) nécessaires et peut gérer des applications. La transition de l'état SELECTABLE à PERSONALIZED est irréversible.

L'état LOCKED indique que le domaine de sécurité est hors service. La transition entre des autres états vers l'état LOCKED est réversible.

L'entité OPEN peut détruire un domaine de sécurité, quelque soit son état.

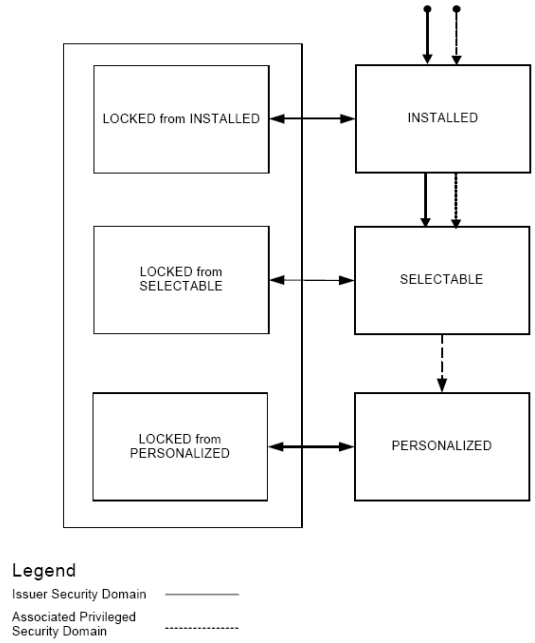
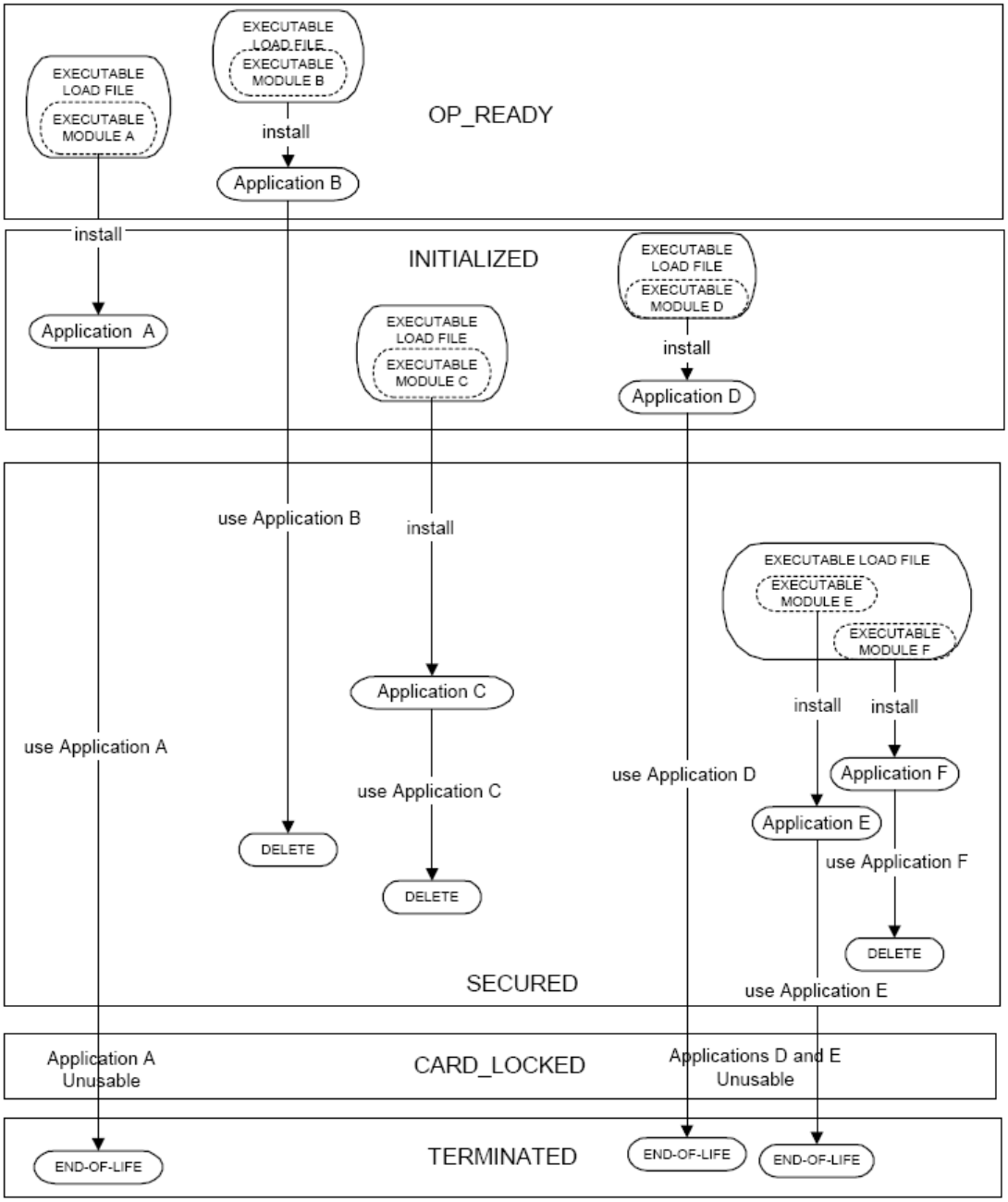


Illustration du cycle de vie d'une carte et de ses applications



Liste des commandes APDUs

Command	OP_READY			INITIALIZED			SECURED			CARD_LOCKED			TERMINATED		
	ISD	DM SD	SD	ISD	DM SD	SD	ISD	DM SD	SD	ISD	DM SD	SD	ISD	DM SD	SD
DELETE Executable Load File															
DELETE Executable Load File and related Application(s)															
DELETE Application	✓			✓			✓								
DELETE Key															
GET DATA	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓		
GET STATUS	✓			✓			✓			✓					
INSTALL [for load]															
INSTALL [for install] (*)	✓	✓		✓	✓		✓	✓							
INSTALL [for make selectable] (*)	✓	✓		✓	✓		✓	✓							
INSTALL [for extradition]															
INSTALL [for personalization]															
LOAD															
PUT KEY	✓			✓			✓								
SELECT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓					
SET STATUS	✓			✓			✓			✓					
STORE DATA	✓			✓			✓								

DELETE. Destruction d'un objet tel que application ou clé.

GET DATA. Lecture d'une information identifiée par un TAG, plus particulièrement une clé

GET STATUS, Lecture d'informations telles que liste d'applications, liste de domaine de sécurité, ou état d'un cycle de vie géré par un domaine de sécurité.

INSTALL. Commande adressée à un domaine de sécurité pour gérer les différentes étapes de l'installation d'une application

LOAD. Chargement d'un fichier. Cette commande est généralement précédé de l'APDU INSTALL [for load] qui indique des options de chargement.

PUT KEY. Création mise à jour ou destruction de clés

SELECT. Sélection d'une application

SET STATUS. Modification de l'état d'un cycle de vie

STORE DATA. Transfert de données vers une application ou une domaine de sécurité

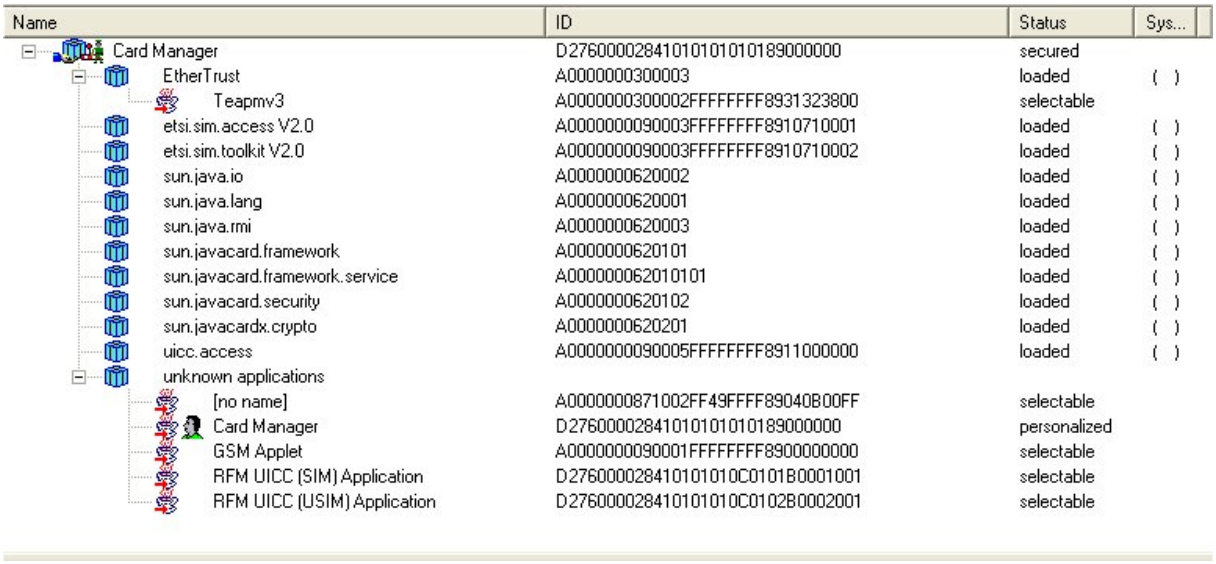
```
ATR=3B E6 00 FF 81 31 FE 45 4A 43 4F 50 32 31 07       ;....1.EJCOP21.


Select Card Manager A0 00 00 00 03 00 00 00 00

=> 00 A4 04 00 08 A0 00 00 00 03 00 00 00 00
<= 6F 19
   84 08 A0 00 00 00 03 00 00 00
   A5 0D 9F 6E 06 40 51 23 05 21 14 9F 65 01 FF
   90 00

initialize-update  CLA=80 INS=50
P1=00 (key version), P2=00 P3=08 = length of the host challenge = 9D B1 90 58 6D 84
B6 96

 => 80 50 00 00 08 9D B1 90 58 6D 84 B6 96
 <= 00 00 23 25 00 47 30 90 18 09 FF 01 57 99 34 CB
    BC AE 75 9B 90 4C 79 38 1B 9A E2 79 90 00

Key diversification data 10 bytes 00 00 23 25 00 47 30 90 18 09
Key information          02 bytes FF 01    FF=Key Version Number 01=Channel
Protocol identifier,
Card challenge           08 bytes 57 99 34 CB BC AE 75 9B
Card cryptogram          08 bytes 90 4C 79 38 1B 9A E2 79
Total

EXTERNAL AUTHENTICATE
P1 = Security level = 00 = No secure messaging expected.
P2 = 0

 => 84 82 00 00 10 29 E5 5B 81 89 02 99 E0 E8 4A 14
    89 66 54 7A 6C
 <= 90 00    Successful execution of the command

Host cryptogram and MAC = 29 E5 5B 81 89 02 99 E0
                           E8 4A 14 89 66 54 7A 6C

DELETE
P1= always 0 P2= 0 = Delete Object(AID=4F 07 4A 54 45 53 54 30 30)
APPLET = JTEST00

 => 80 E4 00 00 09 4F 07 4A 54 45 53 54 30 30          .....O.JTEST00
 <= 00 90 00

DELETE
P1= always 0 P2= 0 = Delete Object(AID=4F 07 4A 54 45 53 54 30)
PACKAGE = JTEST0
 => 80 E4 00 00 08 4F 06 4A 54 45 53 54 30             .....O.JTEST0
 <= 00 90 00

INSTALL
P1= 02 = For Load P2= always zero
 => 80 E6 02 00 13 06 4A 54 45 53 54 30 08 A0 00 00    ......JTEST0....
    00 03 00 00 00 00 00 00
 <= 06 4A 54 45 53 54 30 A0 00 00 00 03 00 00 00 00
    00 00 00 00 90 00

  06 Length of Load File AID
  4A 54 45 53 54 30      AID
  08 Length of Security Domain AID
  A0 00 00 00 03 00 00 00 00 00   Security Domain AID


LOAD  CLA=80 INS=E8 P1=80=more block P1=00=last block P2=bloc number

80 E8 00 00 17
80 E8 00 01 22
80 E8 00 02 0E
```

```
80 E8 00 03 0E
80 E8 00 04 15
80 E8 00 05 80
80 E8 00 0E 80
80 E8 00 0F 10
80 E8 00 10 31
80 E8 00 11 1A
80 E8 00 12 80
80 E8 80 13 05

INSTALL
P1=0C = for load  P2= always 0

06 Length of Load File AID
4A 54 45 53 54 30 Load File AID
07 Length of Executable Module AID
4A 54 45 53 54 30 30 Executable Module AID
07 Length of Application AID
4A 54 45 53 54 30 30 Application AID
01 Length of Application Privileges
00 Application Privileges
02 Length of install parameters field
C9 00 Install parameters field
00 Length of Install Token

 => 80 E6 0C 00 1D 06 4A 54 45 53 54 30 07 4A 54 45    ......JTEST0.JTE
    53 54 30 30 07 4A 54 45 53 54 30 30 01 00 02 C9    ST00.JTEST00....
    00 00
 <= 90 00

GET STATUS
P1= 80 = Issuer Security Domain only. p2= 02= Response data structure
4F00 = according to the reference control parameter P1

 => 80 F2 80 00 02 4F 00 00
 <= 08 A0 00 00 00 03 00 00 00 01 9E 90 00

Length of AID 08
AID A0 00 00 00 03 00 00 00
Life Cycle State 01
Application Privileges 9E

GET STATUS
P1= 40 = Applications and Security Domains only, p2= 02= Response data structure
4F00 = according to the reference control parameter P1

 => 80 F2 40 00 02 4F 00
 <= 07 4A 54 45 53 54 30 30 07 00 90 00               .JTEST00....

Length of AID 07
AID  4A 54 45 53 54 30 30
Life Cycle State 07
Application Privileges 00

GET STATUS
P1= 10 = Executable Load Files and their Executable Modules only. p2= 02= Response
data structure
4F00 = according to the reference control parameter P1

 => 80 F2 10 00 02 4F 00
 <= 6A 86

GET STATUS
P1= 20 = Executable Load Files only. p2= 02= Response data structure
4F00 = according to the reference control parameter P1
 => 80 F2 20 00 02 4F 00
 <= 07 A0 00 00 00 62 00 01 01 00 07 A0 00 00 00 62    .....b.........b
    01 01 01 00 07 A0 00 00 00 62 01 02 01 00 07 A0    .........b......
```

```
    00 00 00 62 02 01 01 00 07 A0 00 00 00 03 00 00    ...b............
    01 00 08 A0 00 00 01 67 41 30 01 01 00 07 A0 00    .......gA0......
    00 01 32 00 01 01 00 07 A0 00 00 00 03 53 50 01    ..2..........SP.
    00 05 A0 00 00 00 63 01 00 06 4A 54 45 53 54 30    ......c...JTEST0
    01 00 90 00                                        ....

Card Manager AID   :  A000000003000000
Card Manager state :  OP_READY

    Application:  SELECTABLE (--------) "JTEST00"
    Load File  :      LOADED (--------) A0000000620001   (java.lang)
    Load File  :      LOADED (--------) A0000000620101   (javacard.framework)
    Load File  :      LOADED (--------) A0000000620102   (javacard.security)
    Load File  :      LOADED (--------) A0000000620201   (javacardx.crypto)
    Load File  :      LOADED (--------) A0000000030000   (visa.openplatform)
    Load File  :      LOADED (--------) A000000167413001 (FIPS 140-2)
    Load File  :      LOADED (--------) A0000001320001
                                        (org.javacardforum.javacard.biometry)
    Load File  :      LOADED (--------) A0000000035350   (Security Domain)
    Load File  :      LOADED (--------) A000000063       (PKCS15)
    Load File  :      LOADED (--------) "JTEST0"
```

```
class gp
{ //  => 80 50 00 00 08 9D B1 90 58 6D 84 B6 96
  //  <= 00 00 23 25 00 47 30 90 18 09
  //     FF 01
  //     57 99 34 CB BC AE 75 9B
  //     90 4C 79 38 1B 9A E2 79
  //     90 00

  //  => 84 82 00 00 10
  //     29 E5 5B 81 89 02 99 E0
  //     E8 4A 14 89 66 54 7A 6C
  //  <= 90 00  Successful execution of the command

  // String card_challenge =  "57 99 34 CB BC AE 75 9B";
  // String host_challenge =  "9D B1 90 58 6D 84 B6 96";

  String card_challenge_H = "57 99 34 CB";
  String card_challenge_L = "BC AE 75 9B";

  String host_challenge_H = "9D B1 90 58";
  String host_challenge_L = "6D 84 B6 96";

  String card_cryptogram = "90 4C 79 38 1B 9A E2 79";
  String host_cryptogram = "29 E5 5B 81 89 02 99 E0";

public void test()
{
// clés VISA
_DES enc1 = new _DES(Reader.a2b("40 41 42 43 44 45 46 47"));
_DES enc2 = new _DES(Reader.a2b("48 49 4A 4B 4C 4D 4E 4F"));
_DES mac1 = new _DES(Reader.a2b("40 41 42 43 44 45 46 47"));
_DES mac2 = new _DES(Reader.a2b("48 49 4A 4B 4C 4D 4E 4F"));

String host_challenge  = host_challenge_H + host_challenge_L ;
String card_challenge  = card_challenge_H + card_challenge_L ;
String derivation_data = card_challenge_L + host_challenge_H + card_challenge_H +
host_challenge_L ;

byte[] b24c = Reader.a2b(host_challenge + card_challenge + "80 00000000000000");
byte[] b24h = Reader.a2b(card_challenge + host_challenge + "80 00000000000000");

byte[] s_enc_h = enc1.cipher(enc2.uncipher(enc1.cipher(Reader.a2b(card_challenge_L
+ host_challenge_H))));

byte[] s_enc_l = enc1.cipher(enc2.uncipher(enc1.cipher(Reader.a2b(card_challenge_H
+ host_challenge_L))));

byte[] s_mac_h = mac1.cipher(mac2.uncipher(mac1.cipher(Reader.a2b(card_challenge_L
+ host_challenge_H))));

byte[] s_mac_l = mac1.cipher(mac2.uncipher(mac1.cipher(Reader.a2b(card_challenge_H
+ host_challenge_L))));

_DES S_ENC_1 = new _DES(s_enc_h);
_DES S_ENC_2 = new _DES(s_enc_l);
_DES S_MAC_1 = new _DES(s_enc_h);
_DES S_MAC_2 = new _DES(s_enc_l);

byte[] card_crypto = mac(S_ENC_1,S_ENC_2,b24c);
byte[] host_crypto = mac(S_ENC_1,S_ENC_2,b24h);
byte[] host_mac    = mac(S_MAC_1,S_MAC_2,Reader.a2b("84 82 00 00 10" + "29 E5 5B 81
89 02 99 E0" + "80 00 00"));

System.out.println("CARD_CRYPTO: "+ Reader.b2s(card_crypto,0,card_crypto.length));
System.out.println("HOST_CRYPTO: "+ Reader.b2s(host_crypto,0,host_crypto.length));
System.out.println("HOST_MAC: "+ Reader.b2s(host_mac,0,host_mac.length));
}
```

Un exemple d'outil Global Platform



Name	ID	Status	Sys...
Card Manager	D276000028410101010189000000	secured	
EtherTrust	A0000000300003	loaded	( )
Teapmv3	A0000000300002FFFFFFFF8931323800	selectable	
etsi.sim.access V2.0	A0000000090003FFFFFFFF8910710001	loaded	( )
etsi.sim.toolkit V2.0	A0000000090003FFFFFFFF8910710002	loaded	( )
sun.java.io	A0000000620002	loaded	( )
sun.java.lang	A0000000620001	loaded	( )
sun.java.rmi	A0000000620003	loaded	( )
sun.javacard.framework	A0000000620101	loaded	( )
sun.javacard.framework.service	A0000000620101 01	loaded	( )
sun.javacard.security	A0000000620102	loaded	( )
sun.javacardx.crypto	A0000000620201	loaded	( )
uicc.access	A0000000090005FFFFFFFF8911000000	loaded	( )
unknown applications			
[no name]	A0000000871002FF49FFFF89040B00FF	selectable	
Card Manager	D276000028410101010189000000	personalized	
GSM Applet	A0000000090001FFFFFFFF8900000000	selectable	
RFM UICC (SIM) Application	D276000028410101010C0101B0001001	selectable	
RFM UICC (USIM) Application	D276000028410101010C0102B0002001	selectable	

XI La carte EMV

Une carte EMV fournit un environnement PSE, ou *Payment Systems Environment*. Pour des raisons historiques, liés aux problèmes de migration d'un parc important de cartes de paiement issues d'un précédent standard (la norme B0') déployé par les banques Françaises au cours des années 80, il est parfois nécessaire de réaliser un reset à chaud (ou *warm reset*) afin d'accéder aux ressources EMV. Ainsi les cartes CB sont identifiées par un double ATR, le premier est retourné après une mise sous tension (*cold reset*), le second est obtenu après une nouvelle remise à zéro (*warm reset*). Par exemple dans le cas d'une carte B0' – EMV émise en 2004 :

Reset à froid, ColdATR = 3F65250891046C9000
Reset à chaud WarmATR = 3F65000091046C9000

Au sujet des Data Objects.

Dans la norme EMV les informations élémentaires (ou *data element*) sont organisées sous forme de Data Objects (ou DO) décrits par la norme ISO 7816. Les éléments d'information sont identifiés grâce à des balises (ou *tags*), selon les règles du langage ASN.1, dans lequel une donnée est codée sous la forme d'une série d'octets Type Longueur Valeur (ou TLV).

Type est représenté par un octet ou plusieurs octets. Lorsque les cinq bits de poids faibles du premier octet sont égaux à un, le code du tag comporte plusieurs octets dont le dernier possède un bit de poids fort forcé à zéro.

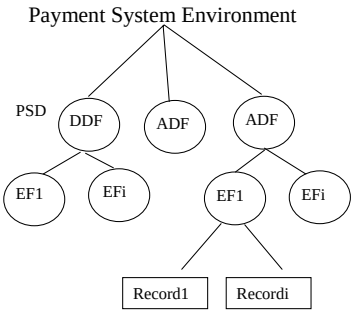
Par exemple le tag 6A (0110 1100) est codé par un seul octet, alors que le tag 9F02 (1001 1111) comporte deux octets.

Lorsque le bit de poids fort de la longueur est égal à zéro, la longueur de la valeur associée au tag est codée par les 7 bits restant. Dans le cas contraire les bits de poids faibles représentent la taille de la longueur.

Ainsi 6A 02 1234 est un DO dont le type est 6A, la longueur 2 octets et la valeur 1234.
6A 81 02 1234 est une notation équivalente, le deuxième octet 81 indique que la longueur de la taille de l'objet est représentée par un octet.

Un DOL ou *Data Object List* est un identifiant d'une liste d'attributs, exprimée sous la forme d'une liste de tags (un ou deux octets) et de la taille attendue (1 octet). Le contenu d'un DOL est une liste de valeurs concaténées, les champs types et longueur étant omis.

Le système de fichier d'une carte EMV



Le système de fichier d'une carte comporte des répertoires DDF (*Directory Definition File*) et des répertoires ADF (*Application Definition File*) dédiés à des applications particulières, qui contiennent des fichiers AEF (*Application Elementary File*). Les éléments d'information mis en œuvre par les applications sont partiellement décrits par la norme EMV, cette dernière fournit donc une architecture de référence, mais permet un certain degré de liberté aux applications de paiements.

Le répertoire PSD

Un répertoire DDF particulier, le PSD (pour *Payment System Director)* nommé «1PAY.SYS.DDF01», permet d'obtenir le mode d'emploi de la carte EMV. Il est activé à grâce à la commande SELECT. Par exemple

```
//SELECT("1PAY.SYS.DDF01")
>> 00 A4 04 00 0E 315041592E5359532E4444463031
<< 612A
>> 00C000002A
<< 6F 28
       84 0E 315041592E5359532E4444463031
       A5 16
        88 01 01
        5F2D 04 6672656E
        9F11  01 01
        BF0C 05 DF60020B05
     9000
```

Après sélection du répertoire PSD, on obtient un FCI (*File Control Information*), c'est-à-dire une suite d'information codée par des *Data Objects*. Ainsi dans notre exemple le tag 84 de longueur 14 octets, répète le nom du répertoire («1PAY.SYS.DDF01»), le tag A5 de longueur 22 octets est une suite de données propriétaires. Le Tag 88 renseigne la valeur du Short File Identifier (*SFI*) du premier enregistrement, le tag 5F2D le langage préféré (*fren*) , le tag 9F11 le mode de codage du nom de l'application préférée, le tag BF0C des données additives gérée par l'émetteur de la carte.

Le répertoire PSD comporte une suite d'enregistrements de taille variable, collecté grâce à la commande READ RECORD. Dans cette dernière les cinq bits de poids fort de l'octet P2 désignent le SFI du fichier, les trois derniers bits codés à 100 indiquent que P1 désigne un numéro d'enregistrement.

```
// ReadRecord#1  - Record=01 (P1=01), SFI = 00001 (P2= 00001 100)
>> 00 B2 01 0C 00
<< 6C19
```

```
>> 00 B2 01 0C 19
<< 7017 6115
      4F07 A0000000421010
      5002 4342
     9F12 024342
     8701 01
     9000

// ReadRecord#2 - Record=01 (P1=01), SFI = 00001 (P2= 00001100)
>>  00 B2 02 0C 00
<<  6C1D
>>  00 B2 02 0C 1D
>>  701B 6119
      4F07 A0000000031010
      5004 56495341
      9F12 0456495341
      8701 02
      9000

//Read Record# 03
>>  00 B2 03 0C 00
<<  6A83, Erreur, fin d'enregistrement
```

Chaque enregistrement est une liste de *Data Object*, le premier TAG a pour valeur '70' (*Directory Record*) le second '61' (*Application Template*). Le tag 4F désigne l'AID d'une application de paiement, les tags 50 et 9F précisent respectivement l'étiquette et le nom de cette application, le tag 87 indique un ordre de préférence. Dans notre exemple il y a deux applications dont les AIDs sont A0000000421010 et A0000000031010, les noms et étiquettes sont 'CB' et 'EMV', avec un ordre de préférence CB puis EMV.

Sélection d'une application EMV

A partir de la liste des AIDs et de leurs attributs, le terminal active une application (c'est à dire un ADF) grâce à la commande SELECT :

```
//  SELECT(A0000000421010)

>> 00 A4 04 00 07 A0000000421010
<< 612C
>> 00 C0 00 00 2C
>> 6F 2A
     84 07 A0000000421010
     A5 1F
      50 02 4342
      87 01 01
      5F2D 04 6672656E
      9F11 01 01
      9F12 02 4342
      BF0C 05
      DF60 02 0B05
```

Il collecte en retour un FCI particulier qui contient des informations liées à l'application de paiement telles que,

- L'AID de l'application (tag 84)
- Une suite de données propriétaires (tag A5)
    - L'étiquette de l'application (tag 50, valeur 'CB')

- Un indicateur de priorité (tag 87)
- Le langage préféré (tag 5F2D, valeur ‘fren’)
- le mode de codage du nom de l’application préférée (tag 9F11)
- le nom préféré de l’application (tag 9F12, valeur ‘CB’)
- des données additives (tag BF0C)

La commande GET PROCESSING OPTION démarre une session EMV

```
GET-PROCESSING-OPTION(8300)
>> 80 A8 00 00 02 8300 –
<< 610C
>> 00 C0 00 00 0C
<< 80 0A
     5C00 08010301 10010501
     9000
```

L’argument 8300 requiert la collecte de l’objet PDOL (*Processing Data Object List*), dont le contenu est encapsulé par le tag 80 de la réponse. Cette valeur consiste en la concaténation de deux champs AIP (*Application Interchange Profile*, 2 octets) et AFL (Application File Locator).

AIP - Application Interchange Profile

Le premier octet de l’attribut AIP précise certaines options fonctionnelles de l’application de paiement
- b8, Toujours zéro
- b7, Authentification statique (SDA), disponible
- b6, Authentification dynamique (DDA), disponible
- b5, Vérification du PIN porteur disponible
- b4, *Terminal Risk Management* demandé
- b3, Authentification de l’émetteur (*Issuer Authentication*) supportée.
- b2, La combinaison DDA et GENERATE AC est disponible
- b1, Toujours zéro

Le *Terminal Risk Management* est un ensemble de procédures de contrôles réalisées par le terminal de paiement afin de limiter les risques de fraudes.

Dans notre exemple, une carte EMV émise en 2004 seule l’authentification statique est disponible (AIP=5C00). Une valeur 3C00 de AIP, observée sur les cartes émises après 2006, indique la disponibilité d’une authentification dynamique DDA.

AFL- Application File Locator

AFL est une liste de fichiers et d’enregistrements utilisés par l’application de paiement; chaque élément de cette liste comporte quatre octets :
- Le premier est le SFI du fichier multiplié par 8
- Le deuxième est le numéro du premier enregistrement
- Le troisième est le numéro du dernier enregistrement
- Le dernier octet indique le nombre d’enregistrements utilisés pour l’authentification off line (SDA) à partir du premier enregistrement (indiqué par le deuxième octet).

Dans notre exemple :

- 08, SFI=1, 01 numéro du 1ier enregistrement, 03 numéro du dernier enregistrement, 01 enregistrement pour l'authentification off line. Le contenu du premier enregistrement, encapsulé par un tag 70 (*Application Elementary File Data Template*), est le tag '57'.
- 10, SFI=2, 01 numéro du 1ier enregistrement, 05 numéro du dernier enregistrement, 01 enregistrement pour l'authentification SDA off line. Le contenu du 1ier enregistrement, encapsulé par un tag 70 (*Application Elementary File Data Template*), est une liste de tags 5F25, 5F24, 5A, 5F34, 9F07, 8E, 9F0D, 9F0F, 5F28.

Grâce aux indications fournies par SFI, il est possible de déduire la liste des objets associée à l'authentification statique. Cette structure est signée (cette valeur est pointée par le tag 93, *Signed Static Application Data*) par la clé de l'émetteur de la carte (Issuer), ou par la clé de la carte (*ICC*) lorsque cette dernière est disponible.

L'application CB comporte donc deux fichiers AEF, le premier (SFI=1) se divise en 3 enregistrements et le second (SFI=2) en cinq enregistrements.

La commande 00 B2 P1 0C 00 (P2= 00001.100) permet de lire les enregistrements du premier fichier (P1 variant de 1 à 3). La commande 00 B2 P1 14 00 (P2= 00010.100) les composants du deuxième fichier (P1 variant de 1 à 5). P2 représente la valeur de SFI multipliée par 8 à laquelle on ajoute 4, P2= 8.SFI + 4.

Chaque enregistrement est une liste de Data Object. Dans le 1er enregistrement du fichier AEF-1, on retrouve, dans le tag 57 le numéro de la carte de paiement. Dans le 2ième enregistrement du fichier AEF-2 le tag 5F20 renseigne le nom du porteur. Le 4ième enregistrement stocke signature des données applicative (tag 93, *Signed Static Application Data*), le certificat de la clé publique de l'émetteur (tag 90, *Issuer Public Key Certificate*) est contenu dans le 5ième enregistrement).

Authentification statiques et dynamiques

Hiérarchie des clés RSA

Une carte EMV comporte au plus quatre certificats et deux clés privées RSA.

Le certificat de l'autorité de certification (Certification Authority, CA) est identifié par un index, pointé par le tag 8F.

On trouve à l'adresse https://partnernetwork.visa.com/cd/vsdc/pubkeys.jsp la liste des clés publiques RSA gérées par la compagnie VISA. La taille de ces dernières varie entre 1024, 1152, 1408, et 1984 bits; l'exposant publique ayant pour valeur 3.

L'émetteur de la carte (ou *Issuer*) possède un certificat délivré par l'autorité de certification. Les cartes qui supportent seulement le mode SDA - *Static Data Authentication* comportent uniquement ce certificat et l'index de l'autorité de certification.

Les cartes supportant la fonctionnalité dite DDA - *Dynamic Data Authentication* possèdent un certificat ICC (*IC Card*) délivré par l'émetteur (*Issuer*) de la carte. Une clé privée (*ICC private key*) permet de délivrer des cryptogrammes (signature) d'authentification dynamique.

De manière optionnelle la carte stocke un certificat signé par la clé privée de la carte (ICC private key). Il permet le chiffrement du PIN (*ICC PIN encipherment*) par le terminal de paiement avec la clé publique, le déchiffrement étant réalisé par la carte grâce à la clé privée correspondante.

Certificats EMV

Un certificat EMV est une valeur de N octets chiffré avec une clé privée de N octets (c'est-à-dire avec un modulo de N octets).

La structure des données avant chiffrement est la suivante :

- Un octet d'entête de valeur '6A'
- un message (MSG1) comportant exactement N-22 octets
- le calcul d'une empreinte H du contenu de MSG1 et d'un optionnel message MSG2.
    - H = sha1 (MSG1 | MSG2)
- Un dernier octet de valeur 'BC'.

Les clés publiques (Issuer et ICC) sont codées à l'aide des trois tags suivants :
- un tag contenant l'exposant publique
    - Tag= '9F32', Issuer Public Key Exponent
    - Tag='9F47', ICC Public Key Exponent
- Un tag contenant une signature, et fournissant au plus N-22 octets du modulo
    - Tag='90', Issuer Public Key Certificate
    - Tag='9F46', ICC Public Key Certificate
- Un tag fournissant sous forme non chiffrée les octets les moins significatifs du modulo
    - Tag='92', Issuer Public Key Remainder
    - Tag='9F48', ICC Public Key Remainder

La structure du champ MSG1 du certificat de l'émetteur est la suivante
- Un premier octet indiquant le format du certificat, toujours '02'
- Les huit chiffres les plus à gauche (soit 4 octets) du numéro de la carte (ou PAN, pointé par le tag '5A'), par exemple '45330182'
- Deux octets indiquant la date de validité, par exemple '1299' pour décembre 1999
- Un numéro de série du certificat codé sur trois octets '123456'
- L'identifiant de l'algorithme de la fonction de digest (1 octet=01 pour sha1)
- L'identifiant de l'algorithme de chiffrement (1 octet=01 pour RSA)
- La longueur du modulo de la clé publique (1 octet)
- La taille de l'exposant de la clé publique (1 octet)
- NCA-36 octets les plus significatifs du modulo de la clé *Issuer*

Le champ MSG2 comporte le reste du modulo (N-NCA-36 octets) concaténé à la valeur de l'exposant publique.

Par exemple pour une clé publique **CA** de 128 octets et une clé ***Issuer*** de 127 octets
MSG1 = "02" + "45 33 01 82" + "12 99" + "12 34 56" + "01" + "01" + "7F" + "01"
| 128-36= 92 octets du modulo Issuer

MSG2= 127-92= 35 octets du modulo Issuer | 03 = exposant de la clé publique Issuer

La calcul de l'empreinte est réalisé sur la concaténation de MSG1 et MSG2 H= sha1(MSG1|MSG2). La signature (V^3 modulo $CA_{1024bits}$) est calculée à partir de la valeur de 128 octets V=6A|MSG1|H|BC.

La structure du champ MSG1 d'un certificat ICC est la suivante

- Un premier octet indiquant le format du certificat, toujours '04'
- Les chiffres du numéro (typiquement 16 chiffres soit 8 octets) de la carte (ou PAN, pointé par le tag '5A'), complétés par des caractères F, afin d'obtenir une longueur totale de 10 octets, par exemple '45 33 01 82 86 33 57 98 FF FF'
- Deux octets indiquant la date de validité, par exemple '1299' pour décembre 1999
- Un numéro de série du certificat codé sur trois octets '123456'
- L'identifiant de l'algorithme de la fonction de digest (1 octet=01 pour sha1)
- L'identifiant de l'algorithme de chiffrement (1octet=01 pour RSA)
- La longueur du modulo de la clé publique (1 octet)
- La taille de l'exposant de la clé publique (1 octet)
- NI-42 octets les plus significatifs du modulo de la clé ICC

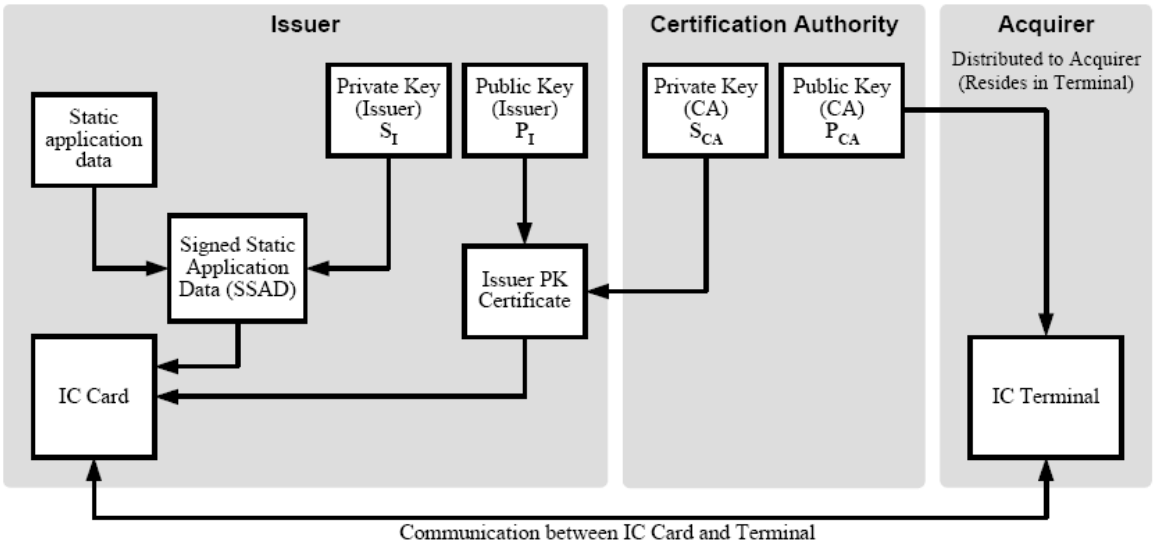
Le champ MSG2 comporte le reste du modulo (N-NI-42 octets) concaténé à la valeur de l'exposant publique et une liste de DO (Type Longueur Valeur) identifiée par la structure *AFL* ou par un tag optionnel, le *Static Data Authentication List* ('9F4A').

Par exemple pour une clé publique ***Issuer*** de 144 octets et une clé ***ICC*** de 112 octets
MSG1 =
- "04" + "4533018286335798FFFF" + "12 99" + "12 34 56" + "01" + "01" + "70" + "01"
- 144-42= 102 octets les plus significatifs du modulo Issuer

MSG2= 112-102= 10 octets du modulo Issuer | 03 = exposant de la clé publique Issuer | liste de DOs

La calcul de l'empreinte est réalisé sur la concaténation de MSG1 et MSG2 H= sha1(MSG1|MSG2). La signature (V^3 modulo $Issuer_{1152bits}$) est calculée à partir de la valeur de 144 octets V=6A|MSG1|H|BC.

Authentification statique



L'authentification statique (SDA) est une signature réalisée à l'aide de la clé privée de l'émetteur, elle est pointée par le tag '93' *Signed Static Application Data.* Elle résulte du chiffrement d'une suite de TAGs, identifiée par la structure AFL ou par un tag optionnel, le *Static Data Authentication List* ('9F4A').

Le champ MSG1 comporte les éléments suivants,

- Un premier octet précise le format de la signature, toujours '03'
- Un octet indique l'identifiant de la fonction de hash (01= sha1)
- Deux octets contiennent le code de l'émetteur (*Issuer Code*).
- NI-26 octets de bourrages, dont la valeur est 'BB'. NI est la taille en octet du modulo de la clé publique de l'émetteur.

Le champ MGS2 contient une liste de DO, exprimée sous la forme type longueur valeur.

Authentification dynamique

Le DDA, ou DDA (*Dynamic Data Authentication* est une mécanisme de signature à l'aide la clé privé ICC.

Le DDOL ou *Dynamic Data Authentication Data* désigne une liste de valeurs dont la carte EMV produit la signature DDA. Le DDOL est associé au tag 9F49, présent dans le 2^{ième} enregistrement du deuxième fichier (AEF-2) d'une application CB. Une valeur 9F3704 désigne une valeur de 32 bits produite par un générateur de nombre aléatoire (*Unpredictable Number Generator*).

Le champ MSG1 comporte les éléments suivants :
- Le premier octet indique le format de signature, toujours '05'
- Un octet identifie la fonction de hash (01=sha1)
- Un octet renseigne la longueur des données LDD dont on désire réaliser la signature
- LDD octets. Le premier octet précise la longueur d'un nombre aléatoire généré par la carte (ICC Dynamic Number).
- NICC – LDD -25 octets de bourrage (de valeur 'BB'), ou NICC désigne la taille du modulo de la clé publique ICC.

MSG2 contient les valeurs pointées par l'objet DDOL.

La commande INTERNAL AUTHENTICATE, dont les données sont codées selon le format précisé par le DDOL permet d'obtenir le résultat du calcul DDA. L'octet P1 codé à 0 indique que l'algorithme de calcul est implicite.

```
>> 00 88 00 00 04 355C833A
<< 61 72
>> 00C0 0000 72
>> 80 70
1D3B8830D24DB766C005E3423822DE4F0
64F72B1D829DAB4252D8AB893A326E56D
CE10A6859461C6C447F9A04702912FCD3
DEDA6C1F891BE3C5A20C2BE4B5CCF8492
BE94A50855AE0897C985D763F61C870C9
56888EA4E1AEADA1EB56838EF9D4C28FA
B9E626397DAE9DCCF8AE16B8B29000
```

Valeur déchiffrée avec la clé publique ICC

```
6A05010908E8F1089C26F48F8EBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBB96D35A9F8E
7940EAA22998781683721487D1460CBC
```

Mode DDA combiné avec ACG

Le terminal délivre une commande GENERATE AC, associée au DOL CDOL1, la carte en retour produit signature.

Le champ MSG1 comporte les éléments suivants :
- Le premier octet indique le format de signature, toujours '05'
- Un octet identifie la fonction de hash (01=sha1)
- Un octet renseigne la longueur des données LDD dont on désire réaliser la signature
- LDD octets.
    - Le premier octet la longueur du ICC Dynamic Number.
    - 2 à 8 octets générés par la carte.
    - Un octet représentant le Cryptogram Information Data
    - 8 octets associés aux tickets TC ou ARQC
- NICC – LDD -25 octets de bourrage (de valeur 'BB'), ou NICC désigne la taille du modulo de la clé publique ICC.

MSG2 contient les valeurs pointées par l'objet DDOL, soit 4 octets aléatoires générés par le terminal.

Certificat de transaction

Une transaction EMV utilise plusieurs types de structures de données
- TC, pour Transaction Certificate.
- AAC pour Application Authentication Cryptogram
- AAR pour Application Authentication Reject
- ARQC pour Authorization Request Cryptogram
- ARPC pour Authorization Response Cryprogram

La commande GENERATE AC (AC signifiant *Application Cryptogram*) supporte trois options fonctionnelles pour le terminal :
- Demande à la carte d'une génération de TC (transaction of line).
- Demande à la carte d'une génération d'un ARQC (transaction on line).
- Demande à la carte d'une génération d'un AAC (abandon de transaction)

Selon le type de transaction le terminal délivre une ou deux commandes GENERATE AC, les données associées à ces commandes sont respectivement décrites par deux structures *Card Risk Management Data Object List*, CDOL1 (tag '8C') et CDOL2 (tag '8D').

Par exemple pour une carte EMV valide en 2006, on observe les valeurs suivantes,
- Pour CDOL 1
    * 9F02 06, le montant de la transaction, par exemple 000000000123 indique un montant de 1,23 £ avec un *currency code* de 826

* 9F03 06 , autre montant, la monnaie (*cashback*) soit toujours zéro (000000000000) dans le cas d'une transaction EMV
* 9F1A 02, le code national du terminal de paiement, par exemple 0250
* 95 05, le résultat de la vérification de la transaction par le terminal, soit une liste de cinq octets nuls (0000000000) si la transaction est sans problèmes.
* 5F2A 02, le *Transaction Currency Code* (par exemple 0826), c'est-à-dire le format de codage du montant de la transaction.
* 9A03, la date de ranscation par exemple 010105 pour le 1ier janvier 2005.
9C01, le type de la transaction (00 pour EMV ?)
9F37 04, un nombre aléatoire de quatre octets (par exemple 12345678)

- Pour CDOL2
* 8A02, le code de la réponse d'autorisation, les deux caractères Y1 (5931 en hexadécimal) indiquent l'accord de la transaction off line.
Z1 = refus off line
Y2 = accord
Z2= refus
* la liste des objets pointés par CDOL1

La liste des valeurs pointées par CDOL1 dans nos exemples est donc,
000000000123 000000000000 0250 0000000000 0826 010105 00 12345678
Et pour CDOL2 on obtient,
5931 000000000123 000000000000 0250 0000000000 0826 010105 00 12345678

La commande GENERATE AC (CLA= 80, INS= AE) précise dans l'octet P1 le type de cryptogramme généré par la carte EMV, soit
- 00 pour AAC
- 40 pour TC
- 80 pour ARQC
Le bit b6 (P1=b8…b1= 0x20) indique si le terminal désire la combinaison DDA/AC, c'est-à-dire la génération d'un certificat DDA
L'octet P2 est toujours codé à zéro

La réponse à GENERATE AC peut être codée selon deux formats, le premier identifié par le tag '80', et le deuxième associé à un DDA, par un tag '70'
Dans le cas du premier format, la réponse est codée selon les règles suivantes
- le premier octet, dénommé *Cryptogram Information Data* précise le type de cryptogramme produit et diverses informations
b8b7= 00, AAC
b8b7=01, TC
b8b7=10, ARQC
b8b7=11, AAR
b4=1, problème détecté
b3b2b1, raison du problème
000, pas d'information disponible
001, service non autorisé
010, nombre de tentatives de présentation du PIN dépassé.
011, échec de l'authentification de l'émetteur
- 2 octets, représentant la valeur d'un compteur de transaction (ATC), *Application Transaction Counter* (Tag='9F36')

- 8 octets représentant un cryptogramme de l'application (AC) c'est-à-dire une signature MAC d'un ensemble de données, produite à l'aide d'une clé secrète
- Des données propres à l'émetteur de la carte (tag=' 9F10') de longueur variable.

Chaque carte EMV possède une clé secrète de 16 octets, dénommée *ICC Master Key* (MK) et un compteur de transaction (ATC) de deux octets. Une clé de session, *ICC Session Key* SK, est dérivée à partir des valeurs MK et ATC.

SK = F(ATC, MK)

La génération d'un *MAC* (Message Authentication Code) associé à une message M, utilise la clé SK associé à l'algorithme de chiffrement symétrique DES

MAC = G(SK, M)

Lorsque le terminal décide de mener une transaction off line, il délivre une requête de génération de TC. La carte répond par un TC si elle accepte, mais peut également imposer un contrôle on line (ARQC) ou refuser la transaction (AAR ou AAC).

Lorsque le terminal décide de mener une transaction on line, il délivre à la carte une requête de génération d'un ARQC. La carte répond par un ARQC si elle accepte la transaction on line, elle peut refuser cette opération via un AAR ou un AAC.

Si le terminal refuse la transaction il délivre à la carte une demande de génération d'un AAC.

Dans le carte d'un contrôle on line de la transaction la carte génère un TC, puis le terminal reçoit la réponse de l'émetteur de la carte qui vérifie l'authenticité de ce cryptogramme. Par la suite le terminal met fin à cette transaction par une demande de génération d'un TC (en cas de succès) ou par une demande de génération d'un AAC en cas d'échec.

Exemple1, transaction off line

```
// Select EMV
>> 00 A4 04 00 07 A0000000031010
<< 61 30
>> 00 C0 00 00 30
// getProcessingOptions
>> 80 A8 00 00 02 8300
<< 6114
>> 00 C0 00 00 14
<< Verify PIN
>> 0020 0080 08 24 1234FF  FFFFFFFF
// GENERATE AC = TC
>> 80 AE 40 00 1D 000000000123 000000000000 0250 0000000000 0826 010105
   00 12345678
<< 6114
>> 00 C0 00 00 14
<< 80 12 40 00 9E 98 68 B3 7A F4 BE 5B A6 06 38 0A
   03 94 00 00 90 00

// 80 tag = format1, length= 12
// Cryptogram Information Data = 9F27 1 octet =         40 = TC
// Application Transaction Counter ATC 9F36 length=2    00 9E
// Application Cryptogram 9F26 length=8                 9868B37AF4BE5BA6
// Issuer Application data 9F10 Length=variable         06380A03940000
```

Exemple 2. Demande d'un TC, la carte demande un ARQC

```
// Select EMV
```

```
>> 00 A4 04 00 07 A0000000031010
<< 61 30
>> 00 C0 00 00 30
<< 6F 2E 84 07 A0 00 00 00 03 10 10 A5 23 50 04 56
   49 53 41 87 01 02 5F 2D 04 66 72 65 6E 9F 11 01
   01 9F 12 04 56 49 53 41 BF 0C 05 DF 60 02 0B 05
   90 00
// getProcessingOptions
>> 80 A8 00 00 02 8300
<< 6114
<< 00 C0 00 00 14
>> 80 12 5C 00 08 01 02 01 18 01 01 00 10 02 03 00
   20 01 03 01 90 00
// GENERATE AC(TC)
>> 80 AE 40 00 1D 000000000123 000000000000 0250 0000000000 0826 010105 00
   12345678
<< 6114
>> 00 C0 00 00 14
<< 80 12 80 00 AC BA 90 7C C7 FD 0B E0 CB 06 38 0A
   03 A0 A8 00 90 00
// 80 tag = format1, length= 12
// Cryptogram Information Data = 9F27 1 octet =         80 = ARQC
// Application Transaction Counter ATC 9F36 length=2    00AC
// Application Cryptogram 9F26 length=8                 BA907CC7FD0BE0CB
// Issuer Application data 9F10 Length=variable         06380A03A0A800
```

Demande de génération d'un TC => OK

```
>> 80 AE 40 00 1F 59 32 00 00 00 00 01 23 00 00 00
   00 00 00 02 50 00 00 00 00 00 08 26 01 01 05 00
   12 34 56 78
<< 61 14
>> 00 C0 00 00 14
<< 80 12 40 00 AC 90 97 86 BC 8F 32 FA 33 06 38 0A 03 60 AC 00 90 00
```

Exemple 3, demande d'un ARQC

```
// Select EMV
>> 00 A4 04 00 07 A0000000031010
<< 61 30
>> 00 C0 00 00 30
<<  6F 2E 84 07 A0 00 00 00 03 10 10 A5 23 50 04 56
    49 53 41 87 01 02 5F 2D 04 66 72 65 6E 9F 11 01
    01 9F 12 04 56 49 53 41 BF 0C 05 DF 60 02 0B 05
    90 00
// getProcessingOptions
>> 80 A8 00 00 02 8300
<< 6114
>> 00 C0 00 00 14
>> 80 12 5C 00 08 01 02 01 18 01 01 00 10 02 03 00
   20 01 03 01 90 00
// Verify PIN
>> 0020 0080 08 24 1234FF  FFFFFFFF
// GENERATE AC = ARQC
>> 80 AE 80 00 1D 000000000123 000000000000 0250 0000000000 0826 010105 00
>> 12345678
<< 6114
>> 00 C0 00 00 14
<< 80 12 80 00 A1 CE 3A 82 76 0B 46 E5 36 06 38 0A
   03 A4 A0 00 90 00
```

```
// 80 tag = format1, length= 12
// Cryptogram Information Data = 9F27 1 octet =         80 = ARQC
// Application Transaction Counter ATC 9F36 length=2    00A1
// Application Cryptogram AC 9F26 length=8              CE3A82760B46E536
// Issuer Application Data IAD 9F10 Length=variable     06380A03A4A000

>> 80 AE 40 00 1F 5931 000000000123 000000000000 0250 0000000000 0826
   010105 00 12345678
<< 6114
>> 00 C0 00 00 14
<< 80 12 40 00 A1 C3 86 B6 28 7F D9 02 8A 06 38 0A
   03 64 A4 00 90 00

// 80 tag = format1, length= 12
// Cryptogram Information Data = 9F27 1 octet =         40 = TC
// Application Transaction Counter ATC 9F36 length=2    00A1
// Application Cryptogram 9F26 length=8                 C386B6287FD9028A
// Issuer Application Data 9F10 Length=variable         06380A0364A400
```

Issuer Authentication

Cette option est validée par le un bit AIP (*Application Interchange Profile*). Après la génération par la carte d'un ARQC (*Authorization Request Cryptogram*) le terminal transmet cette valeur à l'émetteur (*Issuer*) et reçoit en retour un message ARPC (*Authorization Response Cryptogram*), de longueur 8 octets. Cette information est transmise à la carte grâce à une commande EXTERNAL AUTHENTICATE ; cette dernière vérifie

Presentation du PIN

La commande de présentation (VERIFY) d'un PIN code de 4 chiffres (par exemple 1234) est codée de la manière suivante,

00200080 08 24 1234 FF FFFFFFFF