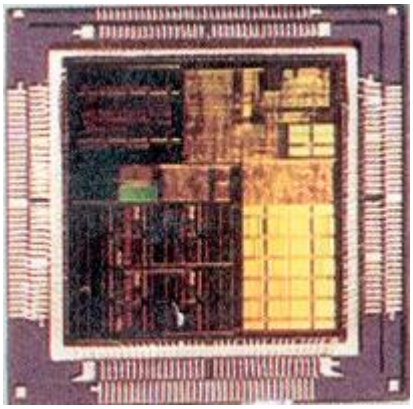
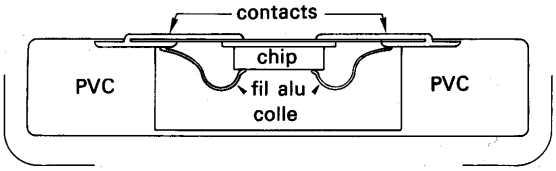
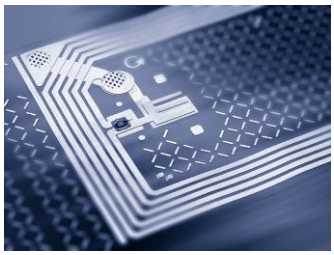
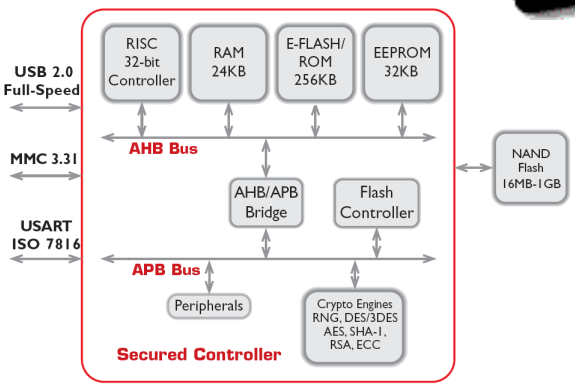
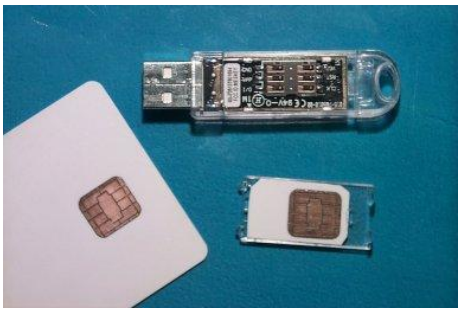


Cartes à puce









TELECOM Paris

IP PARIS

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 premiers distributeurs automatiques de billets (DAB) sont déployés en 1971 et utilisent des cartes bleues munies de piste magnétique (une piste magnétique 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 mémoire) 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 données (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 le composant qui va équiper toutes les cartes à puce.

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 commandés 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 françaises. Le groupement des cartes bancaires (CB) émet une première commande de 12,4 millions de cartes.

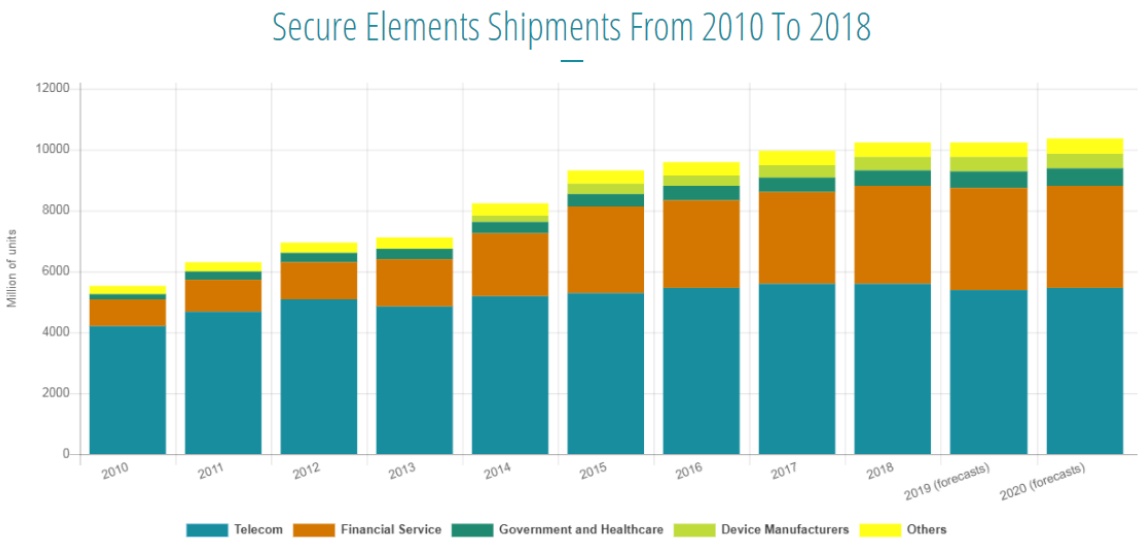
Les standards de base des cartes à puces (ISO 7816) sont introduits à partir de 1988.

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, et sont usuellement dénommés *dotnet card*.

Les marchés.

La plus importante entreprise de cartes à puce GEMALTO (rachetée en 2019 par la société THALES), emploie environ 15,000 personnes. En 2018 son chiffre d'affaire était de 3 milliards d'euros.
Les tableaux suivants illustrent quelques aspects du marché de la carte à puce.



	2016	2017	2018f	2017 vs 2016% growth	2018 vs 2017% growth
Telecom*	5450	5600	5600	2,75%	0,00%
Financial services	2900	3000	3150	3,45%	5,00%
Government - Healthcare	460	485	510	5,43%	5,15%
Device manufacturers**	330	400	470	21,21%	17,50%
Transport	260	280	300	7,69%	7,14%
Pay TV	120	100	95	-16,67%	-5,00%
Others***	90	90	90	0,00%	0,00%
Total	9610	9955	10215	3,59%	2,61%

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.

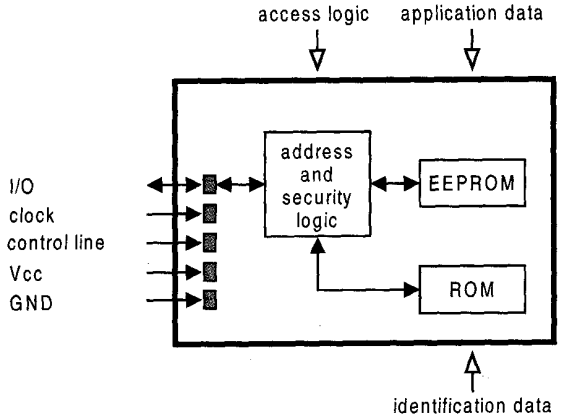
Les principales caractéristiques d'une carte à puce sont les suivantes,

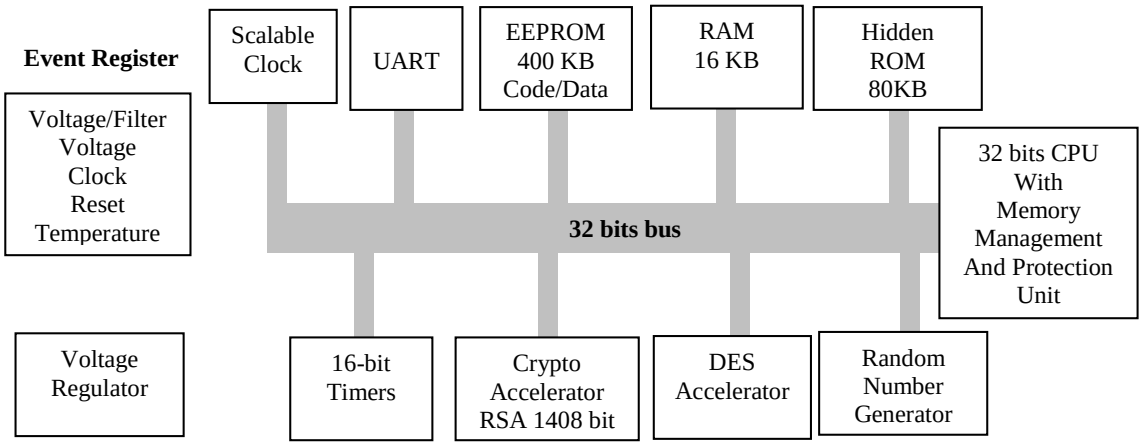
- C'est un objet portable, qui loge des données et des procédures.
- C'est un objet sécurisé
  - Il est difficile de lire les données stockées dans les mémoires de la puce.
  - Le code est exécuté dans un espace de confiance, il n'est pas possible (difficile) d'obtenir les clés associées à des algorithmes 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 milliards 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.

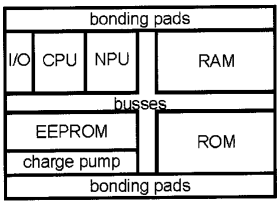
Elles comportent un bloc de sécurité (optionnel) qui contrôle les accès à des mémoires de type ROM ou E^2PROM. Les télécartes de 1^{ière} génération (ou TG1) n'étaient pas sécurisées; les télécartes de 2^{ième} génération (ou TG2) comportent un bloc de sécurité.



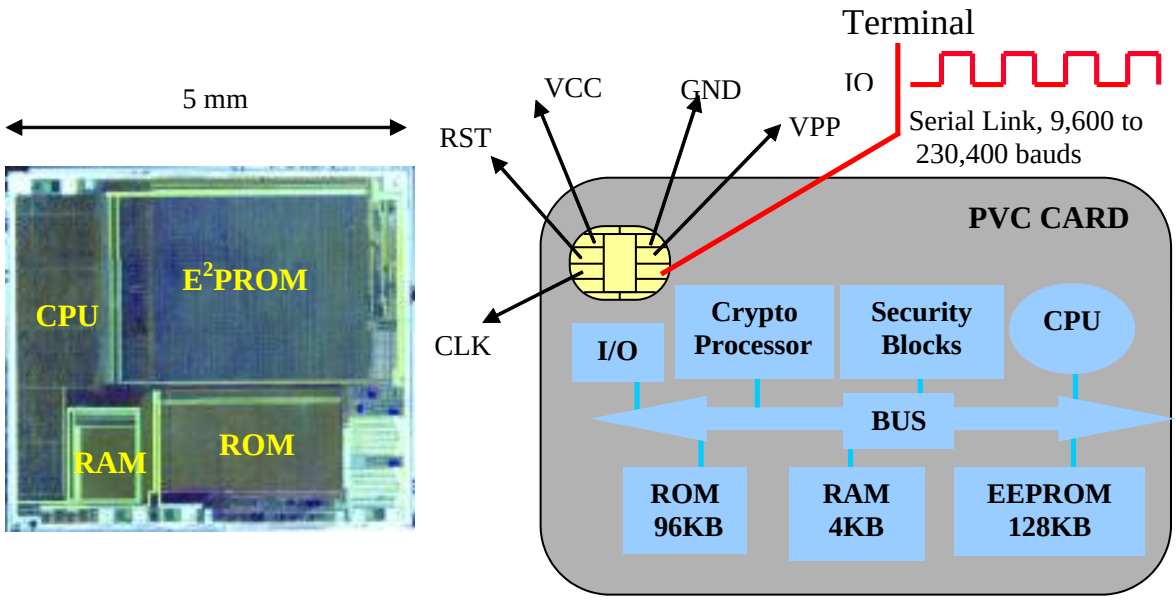


Le microcontrôleur 88CFX4000P

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 pourrait 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.
Les technologies de type mémoires FLASH permettent des capacités de l'ordre du Mo. Les puissances de calculs estimées sont de l'ordre de 100 à 200 MIPS.

Couches de communication 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

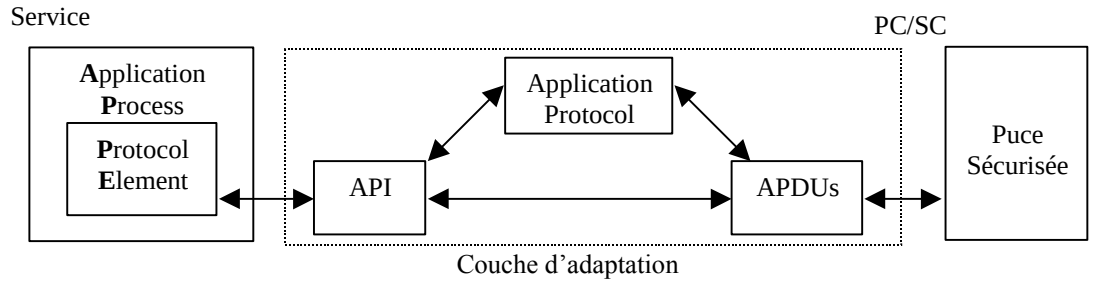
Commandes (APDUs) définis par la norme ISO 7816 pour le protocole T=0.

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 définit également une interface USB à 12 Mbit/s. Le terminal envoie 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 statut (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 embarquée est stockée dans un système de fichiers qui comporte un répertoire racine (MF Master File), des sous répertoires (DF Dedicated File) et des fichiers (EF Elementary File). Chaque élément 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.

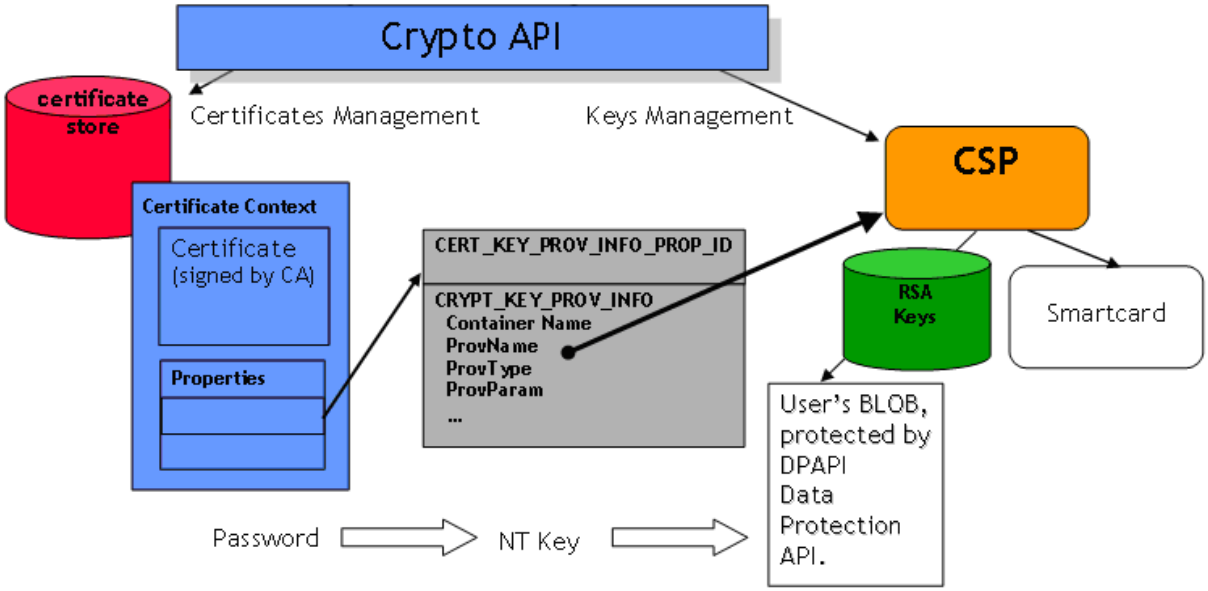
Intégration des cartes à puce aux technologies de l'information.

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.



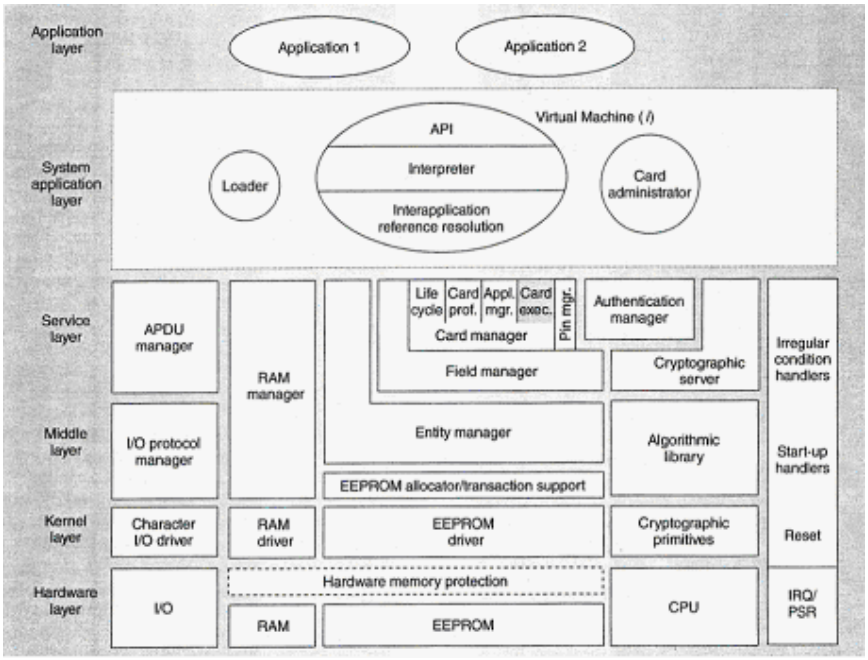
Crypto Service Provider Microsoft

Système d'exploitation d'une carte à puce.

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

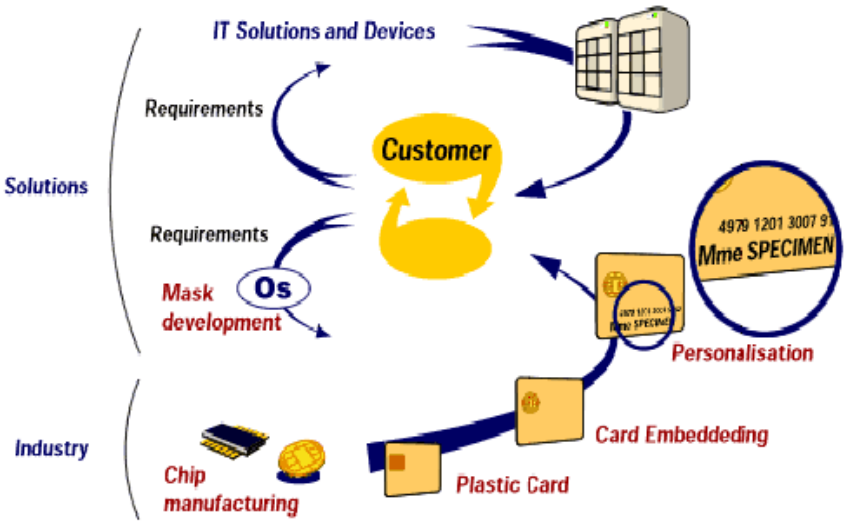
- Un bloc de gestion des ordres (APDUs) transportés par la liaison série.
- Une bibliothèque de fonctions cryptographiques, dont le code est réalisé de telle manière qu'il soit résistant aux attaques logiques connues (*Timing Attack*, DPA, SPA, …).
- Un module de gestion de la RAM
- Un module de gestion 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.



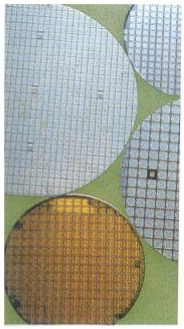
Un exemple de système d'exploitation

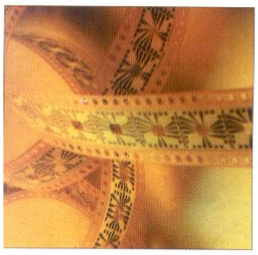
Cycle de vie d'une carte à puce.

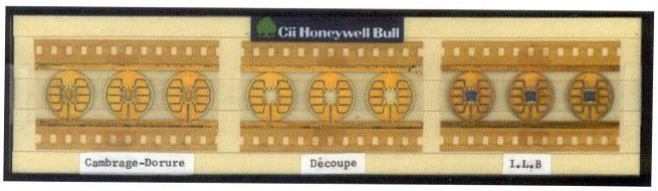


Cycle de vie d'une carte à puce

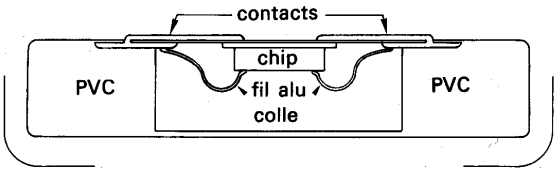
Un système d'exploitation est réalisé par une entreprise spécialisée. Ce logiciel étant ajusté pour un composant électronique particulier, il est appelé ***masque***.
Le masque est stocké dans la ROM du composant lors du processus de **fabrication**. Au terme de cette phase le fondeur de silicium écrit 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 puce

- 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 octet) 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.

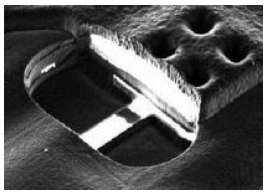
Cette carte est une carte dite d'usage général (*general purpose*) commercialisée dans le courant des années 90 par la société CP8. Elle intégrait des algorithmes cryptographiques DES, RSA ainsi que des mécanismes d'authentification par PIN code et blocage après trois présentations infructueuses. Son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

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).

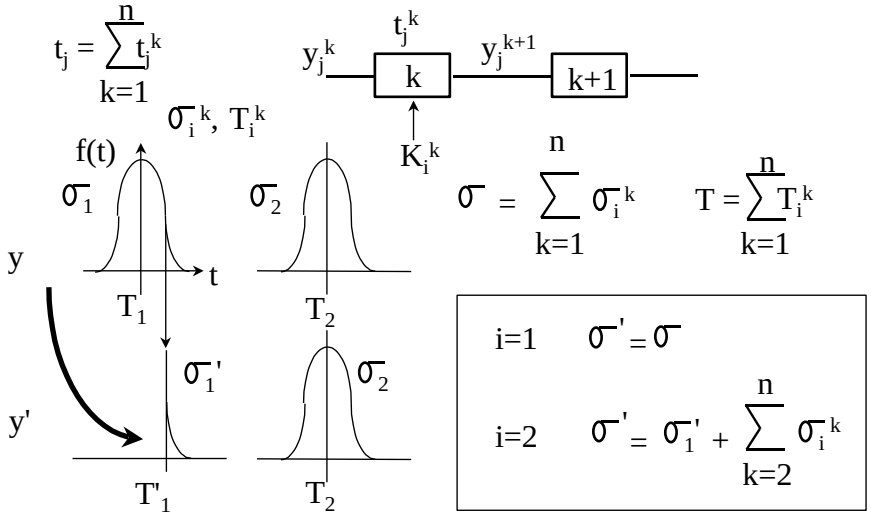
Attaques temporelles (moyenne, écart type). Certaines implémentations logicielles d'algorithmes peuvent présenter des temps de calculs différents en fonction des valeurs calculées et de la clé utilisée.

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.

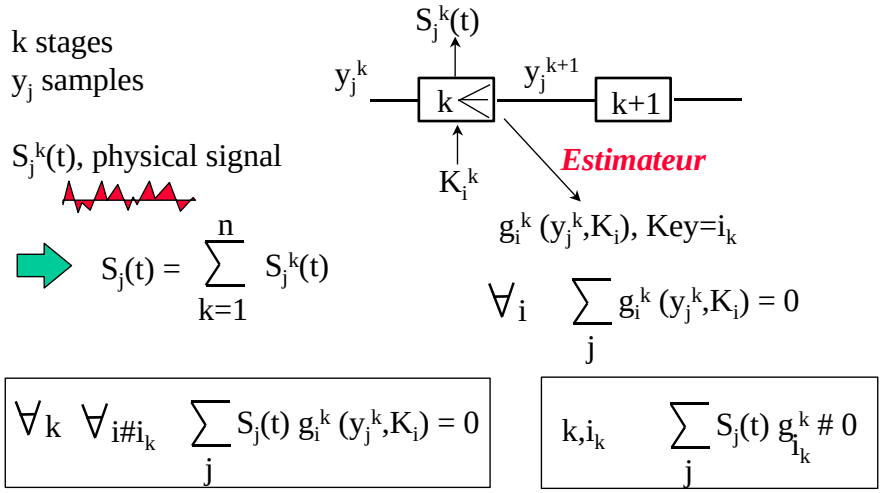
Ainsi un algorithme utilisant une clé parmi $n=2^p$ possible, peut utiliser au moins p instructions différentes pour une clé particulière. Il produit donc des signaux électriques (par exemple puissance consommée) ou radioélectriques qui sont corrélés à la clé opérationnelle.

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.

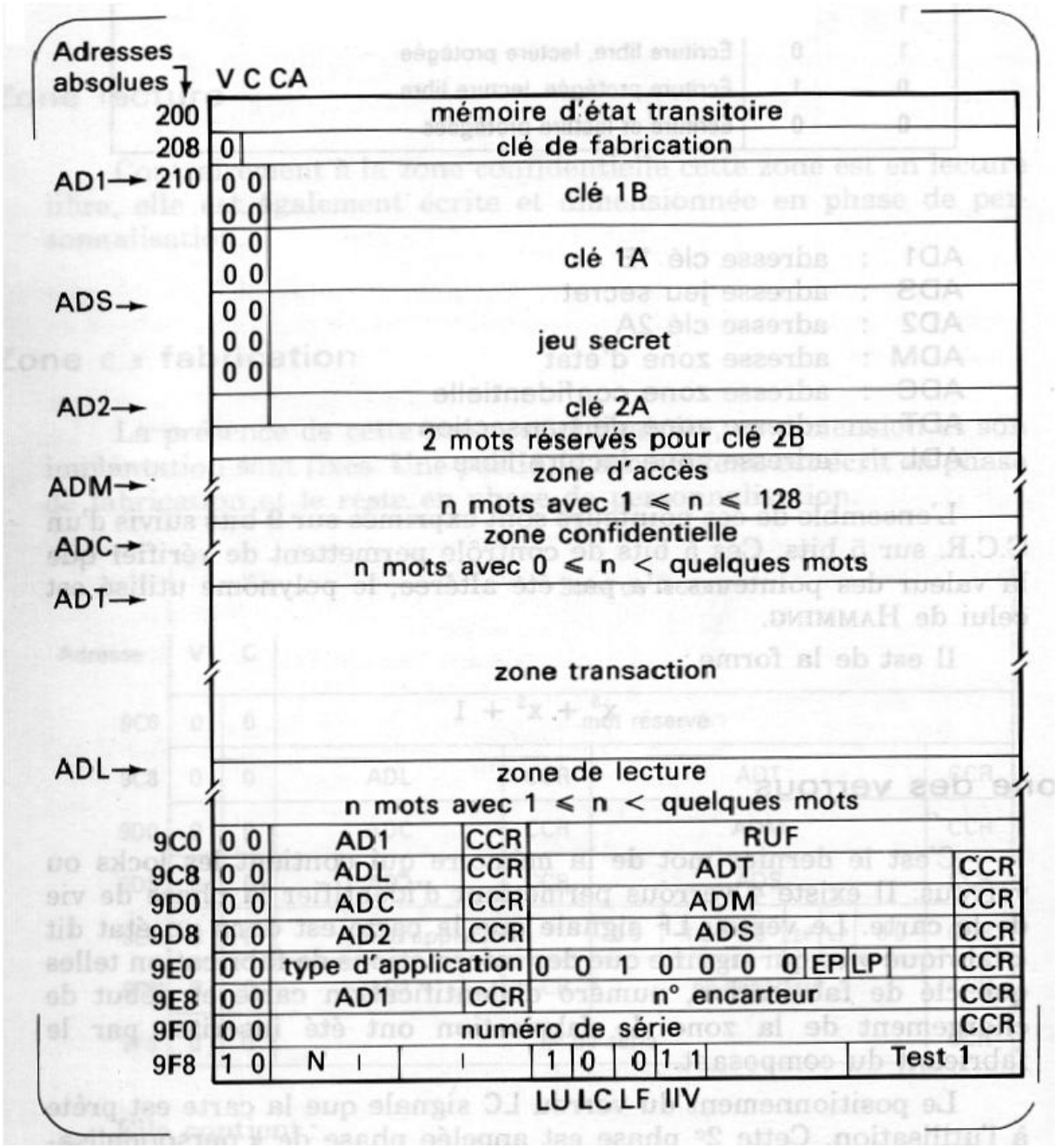


Attaque par *Ecart Type*. L'idée est d'utiliser une sous population de la sous clé (i) d'un étage de calcul (k) qui modifie l'écart type du temps de calcul du système



Attaque par corrélation.

Etapes de fabrication d'une carte à puce



Organisation de la mémoire d'une carte bancaire B0'

II 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.

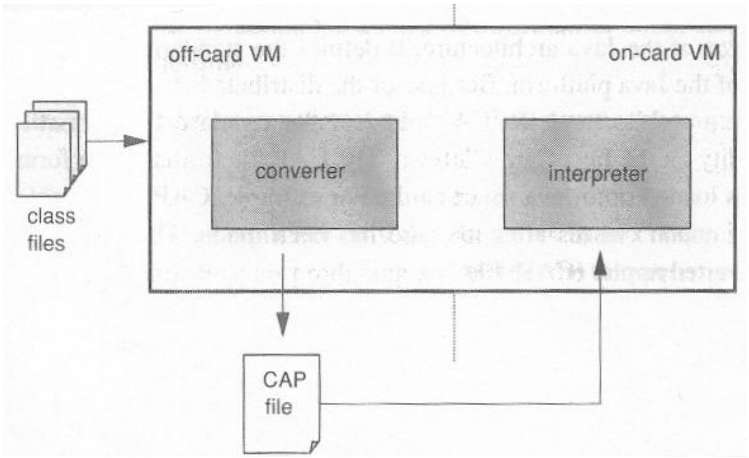
ClassFile {
u4 magic; // 0xCAFEBABE ;
u2 minor_version; // plus petite version supportée
u2 major_version; // plus grande version supportée
u2 constant_pool_count; // nombre d'entrée de la table + 1
cp_info constant_pool[constant_pool_count-1] ; // table des constantes *cp_info*
u2 access_flag; // privilèges 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.

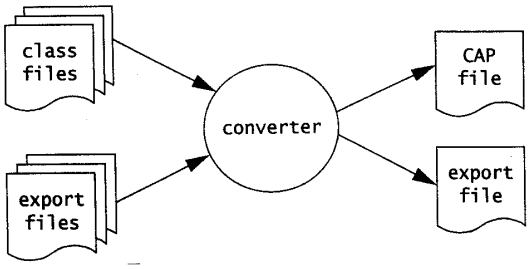
La norme JavaCard 2.x

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*)

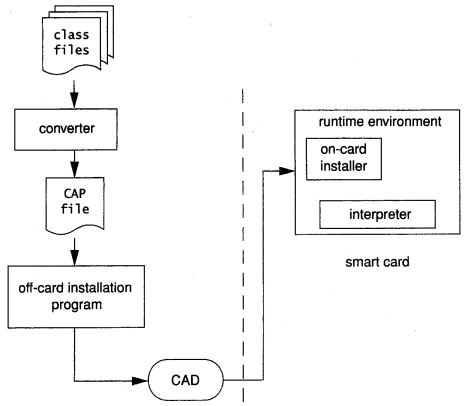


Compte tenu de la puissance de traitement d'une carte à puce la machine virtuelle est constituée par deux entités distinctes, l'une est logée sur une station de travail ou un ordinateur personnel (off-card VM), et l'autre est embarquée dans la carte (on-card VM).

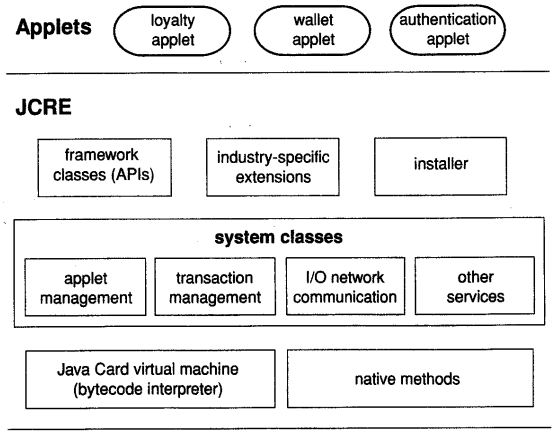


Un ensemble de fichiers java est compilé par un compilateur javac standard qui produit des fichiers .class. Un programme nommé *converter* implémente une partie de la machine virtuelle, il vérifie la comptabilité des classes avec le langage javacard, charge et lient ces objets dans un paquetage. Le résultat 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.

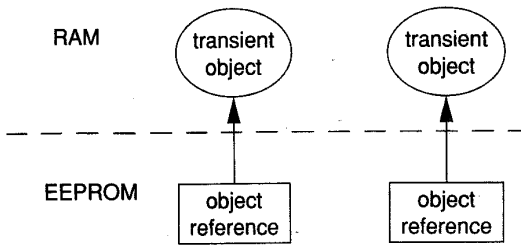
Cycle de développement d'un Applet.

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 (byte, 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,* sha1…), algorithmes à clés symétriques (DES, AES…) ou à clés asymétriques (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

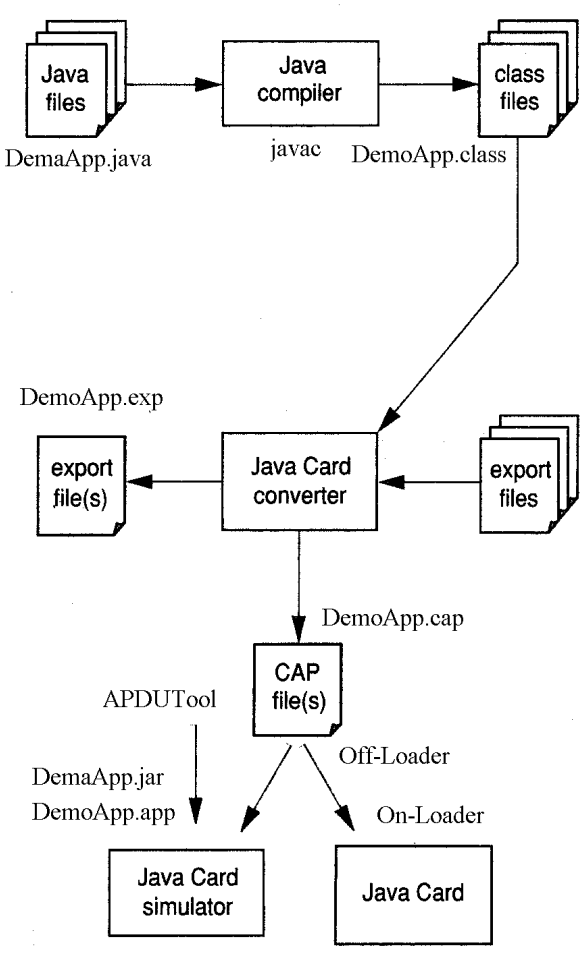
  Read 00 B0 P1 P2 Le.

  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    0xa0:0x0:0x0:0x0:0x0:0x62:0x3:0x1:0x8:0x1
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;
```

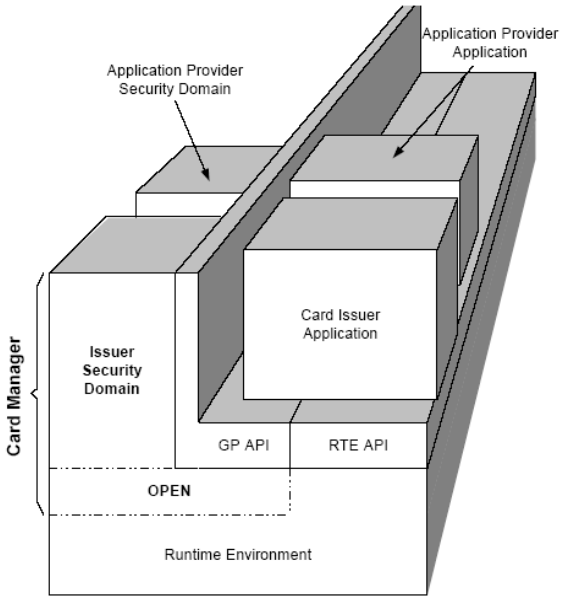
Global Platform

Introduction

L'architecture Global Platform comporte des composants réalisant 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 des droits 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

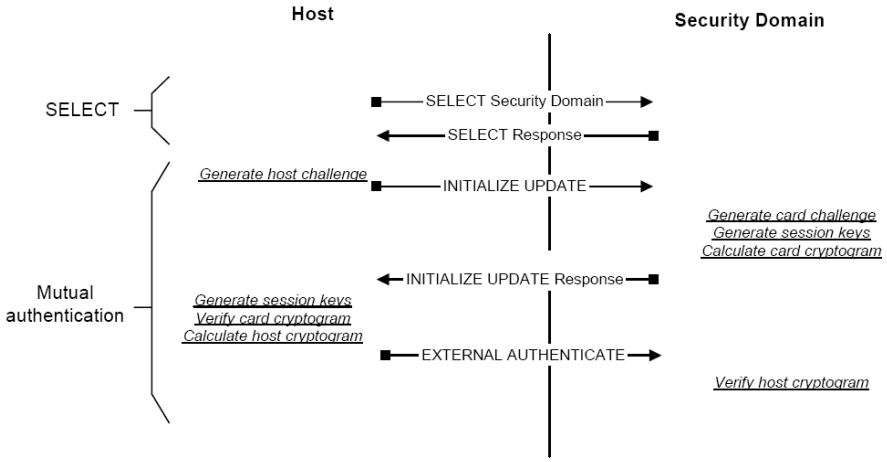
Cette interface délivre des services GP aux applications, par exemple la vérification du porteur de carte, des fonctions de sécurité, ou la gestion du cycle de vie.

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*) préinstallée 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 session sont calculées à partir d'une clé dénommée *Secure Channel base key* (de 16 octets) qui s'applique à un 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.

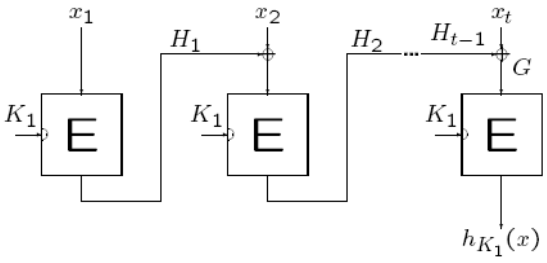
Protocole 01

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 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 nuls est appliqué à la valeur B24 pour produire une signature selon la méthode ISO 9797-1 (*MAC Algorithm 1 with output transformation 1*).



Méthode ISO 9797-1 (*MAC Algorithm 1 with output transformation 1*)

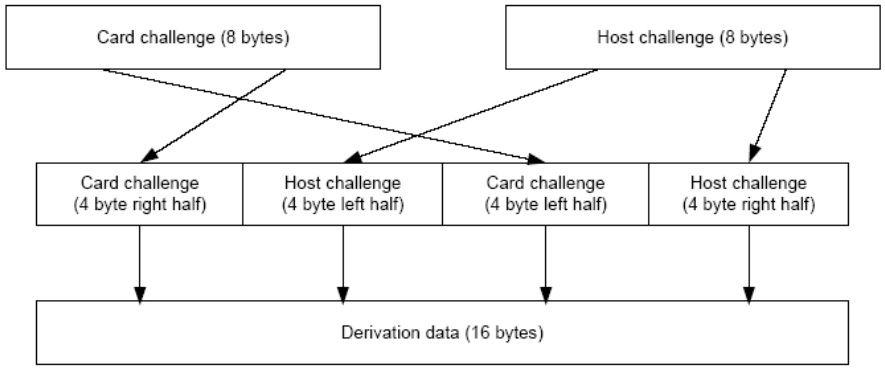


Figure D-3: Session Key - Step 1 - Generate Derivation data

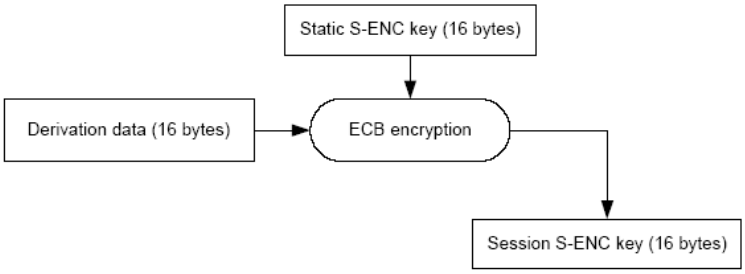


Figure D-4: Session Key - Step 2 - Create S-ENC Session Key

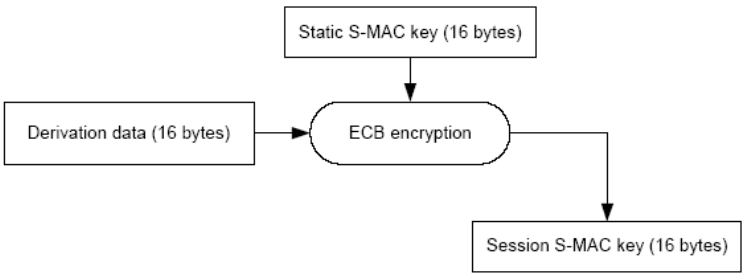


Figure D-5: Session Key – Step 3 - Create S-MAC Session Key

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

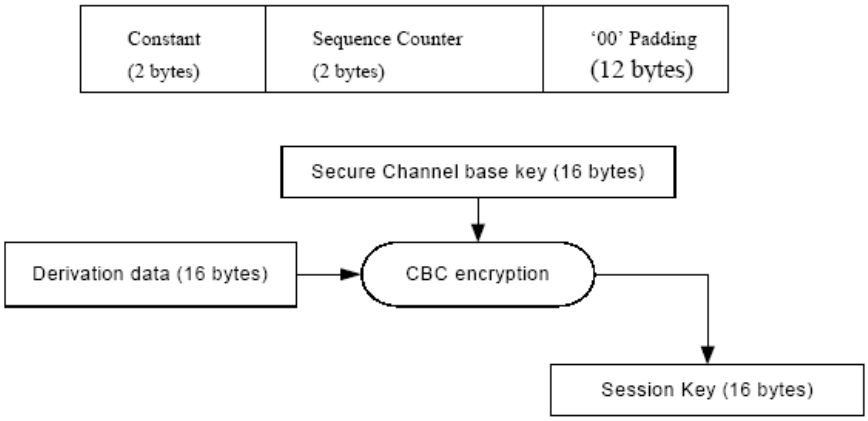


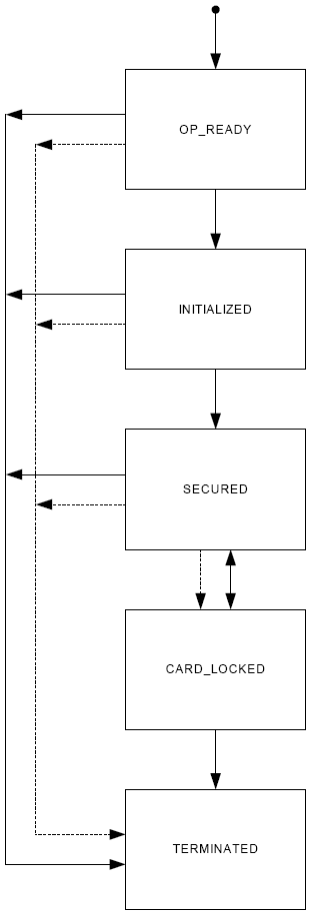
Figure E-2: Create Secure Channel Session Key from the Base Key

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

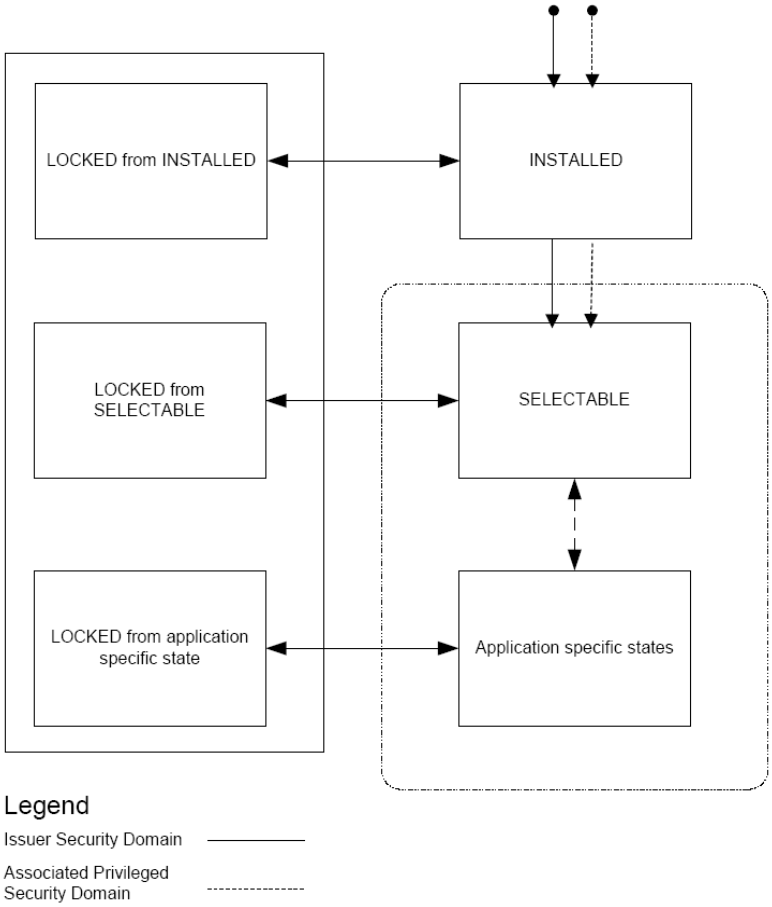
Cependant, dans l'état SELECTABLE une application peut gérer un sous état propre, qualifié de SPECIFY.

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.

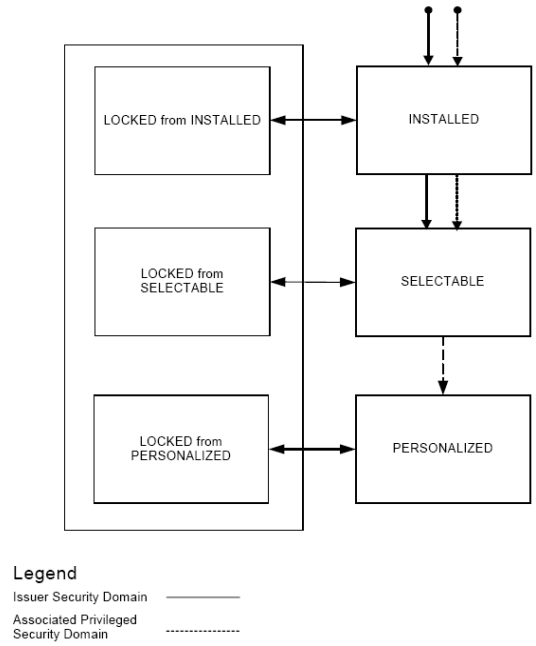
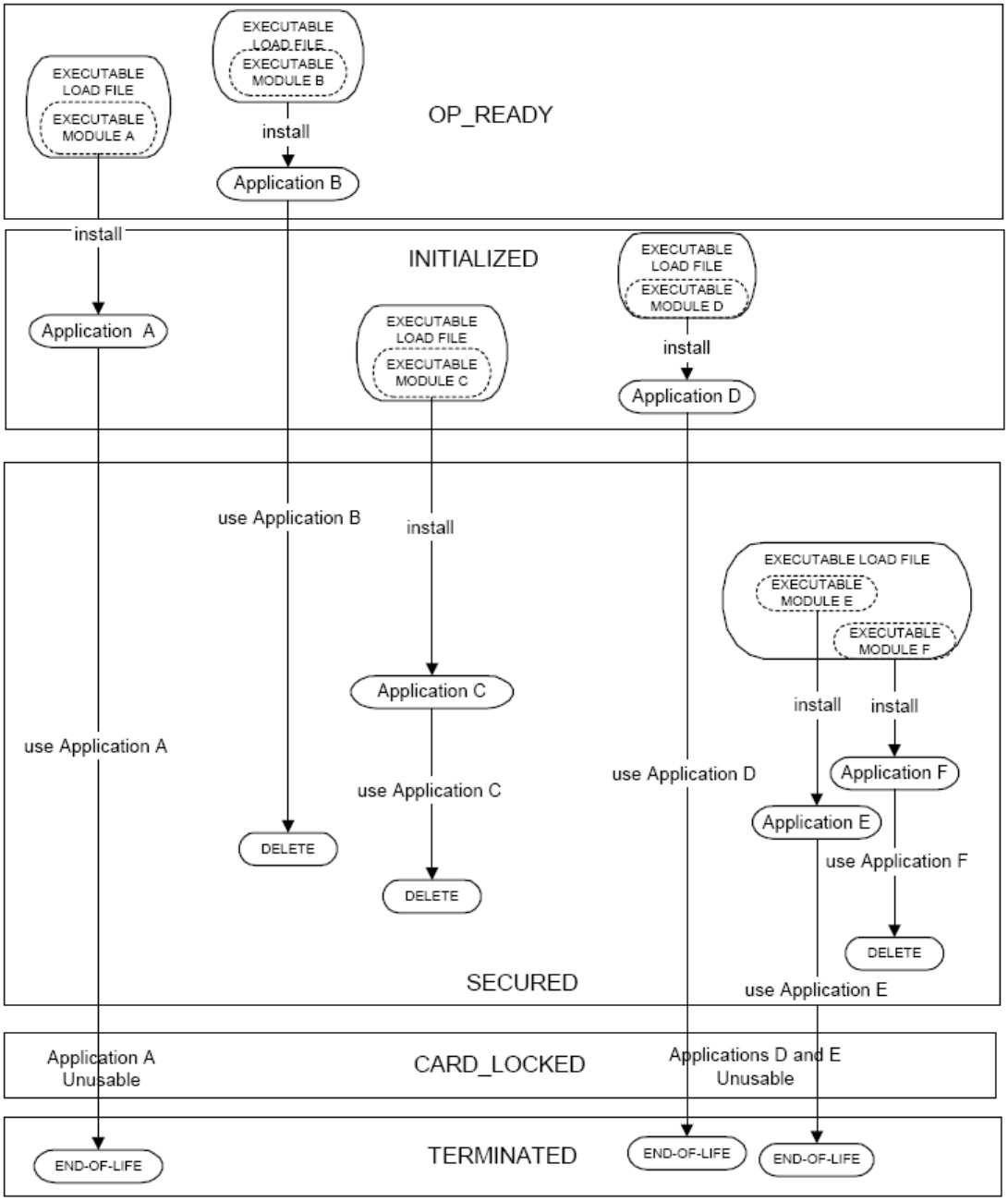


Illustration du cycle de vie d'une carte et de ses applications



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 qu'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 précédée 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

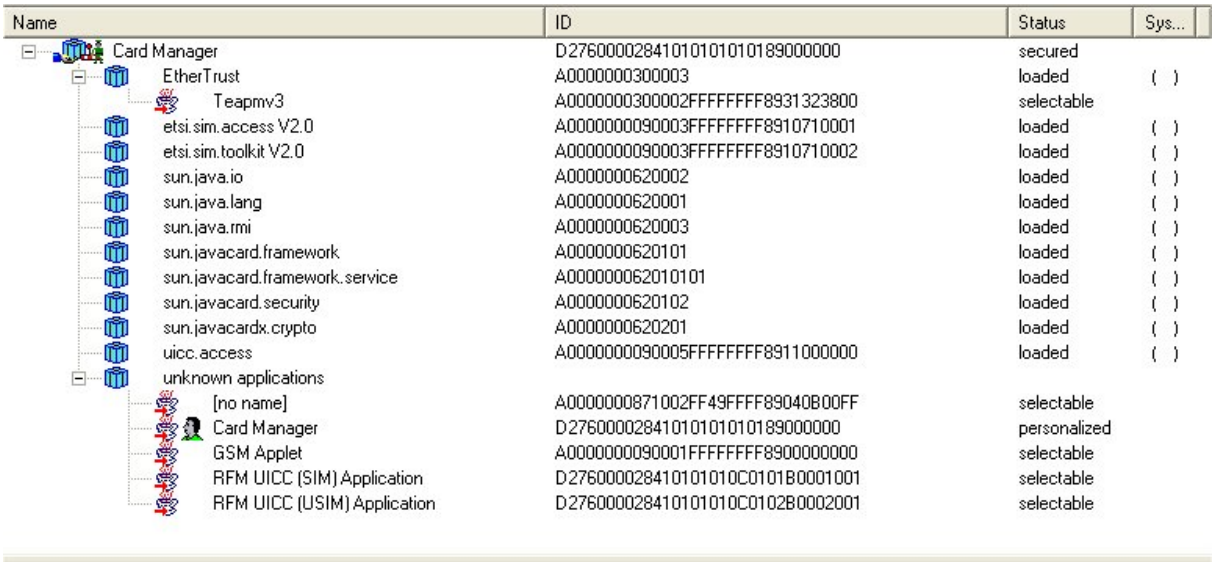
```
ATR=3B E6 00 FF 81 31 FE 45 4A 43 4F 50 32 31 07       ;....1.EJCOP21.


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 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" + "2 9 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	