# Chapitre 1 : Introduction à la carte à puce

## I.1 Définition :

La carte à puce est un circuit électronique conçu pour manipuler des données (stockage, calcul, etc.). Elle est équipée d'un microprocesseur et d'une mémoire intégrée, et se présente sous forme d'une petite carte. Les cartes les plus répandues sont les cartes bancaires et téléphoniques, contenant les données de l'utilisateur ainsi que des protocoles de communication avec les lecteurs, avec un fort accent sur la sécurité.

microprocesseur:coeur du système informatique(moteur de calculs plus E/S+mémoire externe).HARVARD

contrôleur:coeur du système embarqué(contient le tout dans le même boitier).VAN NEUMAN

*/Les caractéristiques:

-Autonome:la consommation doit être très faible(énergie).

-Sûreté de fonctionnement:le résultat est juste dans le délai attendu.

-sécurité:protéger les informations contre toute intrusion malveillante.

-espace compté:miniaturisation des composants->mémoire très limitée.

-puissance de calcul:juste le nécessaire pour éviter la consommation.



## I.2 Les réalisations industrielles majeures :

**Carte bancaire française (GIE-CB)** : En 2004, 49 millions de cartes étaient en circulation, avec 305 milliards d'euros de transactions.

**Téléphonie mobile (1992-2005)** : 5 millions de cartes SIM vendues.

**Porte-monnaie électronique (1996-2005)** : 100 millions de cartes distribuées dans 18 pays (Suisse, Belgique, Pays-Bas, Suède), dont 80 millions de Geldkartes en Allemagne, Luxembourg, Islande.

**Carte de santé SESAM-Vitale (1996-2005)** : 45 millions de cartes en circulation en 2004.

## I.3 Normes ISO 7816 pour les cartes à contact:

Le standard ISO est la principale norme définissant les caractéristiques des cartes à puce avec contact électrique.

Il y a 15 normes proposées pour les cartes à contact.

**SO 7816-1** : Caractéristiques physiques (dimensions et contraintes).

**ISO 7816-2** : Aspects électriques et position des contacts.

**ISO 7816-3** : Caractéristiques électriques (fréquence, vitesse de communication).
**ISO 7816-4** : Système hiérarchique de fichiers et commandes de gestion des données.
**ISO 7816-5** : Numérotation et enregistrement des identifiants d'applications (AID).
**ISO 7816-6** : Données inter-industrie (nom, date d'expiration).
**ISO 7816-7** : Commandes inter-industrie pour la gestion de bases de données dans la carte à puce.
**ISO 7816-8** : Sécurité de l'architecture et des commandes inter-industrie.
**ISO 7816-9** : Commandes supplémentaires pour la sécurité, telles que Create File, Delete File et recherche.
**ISO 7816-10** : Protocoles de communication pour les cartes synchrones (T=14).
**ISO 7816-11** : Représentation des données, organisation et vérification biométrique.
**ISO 7816-12** : Interface électrique USB et procédures de fonctionnement des cartes à puce.
**ISO 7816-13** : Commandes pour un environnement multi-applicatif, basé sur Global Platform.
**ISO 7816-15** : Stockage standardisé des données cryptographiques, accès aux clés publiques et certificats.



## I.4. Les différents types de cartes à puce:

Il existe plusieurs types de cartes à puce, on cite dessous

### I.4.1. La carte à mémoire:

Les cartes à mémoire, sans processeur, disposent d'une petite capacité (1 à 4 Ko) et sont conçues pour des tâches spécifiques. On distingue :

1. **Cartes à mémoire simple** :
    - Lecture accessible sans protection.
    - Écriture limitée et non réinscriptible.
    - Utilisées pour des tâches simples (ex : cartes téléphoniques).
2. **Cartes à mémoire avec logique câblée** :

    - Incluent des circuits préprogrammés pour des fonctions spécifiques.
    - Peuvent gérer des compteurs électroniques.
    - Plus sécurisées, mais rigides et non évolutives.

Ces cartes sont limitées en comparaison des cartes à microprocesseur en raison de l'absence de processeur.

### I.4.2. La carte à microprocesseur :

La carte à microprocesseur est la plus avancée, intégrant un microprocesseur capable d'exécuter des applications. Elle permet l'enregistrement, la restitution d'informations et le traitement de données. Ses caractéristiques incluent :

- **Taille de la puce** : 25 mm²
- **Microprocesseur (CPU)** : 8, 16 ou 32 bits avec architecture RISC (horloge interne de 100 à 200 MHz)
- **ROM** : 16 à 24 Ko
- **EEPROM** : 8 à 64 Ko
- **RAM** : 1 Ko
- **Coprocesseur cryptographique**
- **Générateurs de nombres aléatoires**
- **Vitesse de communication** : paramétrable.



**ROM (Read-Only Memory)** : Mémoire persistante et figée, qui ne nécessite pas d'énergie pour conserver les données. Son contenu ne peut pas être modifié et sert à stocker le système d'exploitation de la carte.
**RAM ( Random Access Memory)** : Mémoire de travail temporaire utilisée pour les calculs, avec un accès rapide. Elle est volatile, ce qui signifie qu'elle perd son contenu lorsque la carte est éteinte. Elle peut être lue et écrite indéfiniment.
**EEPROM (Electrically Erasable Programmable Read-Only Memory)** : Mémoire modifiable qui conserve son contenu même hors tension. Elle est utilisée pour stocker des informations susceptibles d'évoluer. Ses inconvénients incluent une endurance limitée (environ 100 000 cycles d'écriture) et une latence, ce qui rend l'accès plus lent.

### I.4.3. La carte à logique câblée :

La carte à logique câblée possède une mémoire et des règles intégrées physiquement dans le silicium. Elle est souvent utilisée pour les cartes téléphoniques et peut réaliser des calculs figés, ce qui limite sa flexibilité et son évolutivité.
Ses caractéristiques incluent :

- Mémoire accessible par circuits préprogrammés.
- Passage de l'EPROM à l'EEPROM pour intégrer des compteurs.
- Gestion de compteurs sous forme de bouliers électroniques.
- Absence de CPU.
- Utilisation d'algorithmes d'authentification et de procédures CBC pour lier authentification et signature des transactions.

### I.4.4. la carte à contact :

La carte à contact, conforme à la norme ISO 7816, utilise des communications série via huit contacts, mais souffre d'une usure due à l'insertion et au retrait fréquents ainsi que des problèmes d'orientation dans le lecteur.



### I.4.5 La carte sans contact :

La carte sans contact, comme la MIFARE, permet des transactions de débit et s'applique à divers domaines tels que le contrôle d'accès. Elle utilise une antenne intégrée et récupère son énergie par couplage capacitif ou inductif. Cependant, elle présente des limitations, notamment une distance de communication d'environ 10 cm, un temps de transaction de 200 ms, et un coût élevé.

### I.4.6 La carte combi (due à l'interface):

C'est une combinaison entre la carte à contact et la carte sans contact. Ces deux possibilités de communication forment une carte idéale.

## I.5. Les domaines d'utilisation de la carte à puce :

Les cartes à puce sont largement utilisées dans les domaines des télécommunications, de la banque, de la santé, des porte-monnaie électroniques, des transports en commun, du contrôle d'accès physique et de l'identification sur Internet.

**Remarque sur les communications:**

Les cartes à puce, compactes et passives, nécessitent une alimentation externe fournie par un lecteur de carte, qui sert d'adaptateur pour la communication avec des ordinateurs.

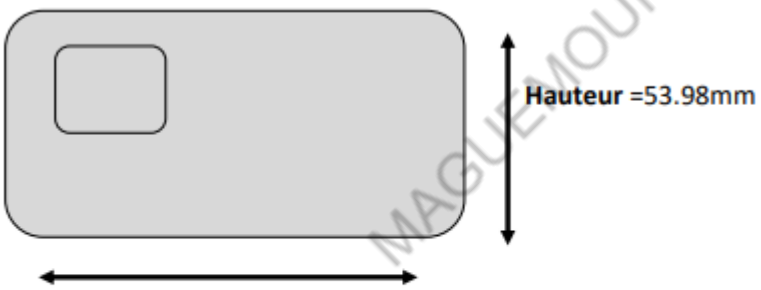
La carte à contact nécessite une insertion dans un lecteur, la carte sans contact utilise des ondes électromagnétiques pour des transactions rapides sans insertion, et la carte à interface combine les deux technologies, permettant une compatibilité avec tous les types de lecteurs.

## I.6. Caractéristiques physiques de la carte à puce :

Une carte à puce est constituée de trois parties :

Un support en plastique, aux dimensions suivantes :

Un module de contact, assurant la liaison entre la puce et son lecteur.




**Longueur**=85,6 +/-0,12

**Epaisseur** =0.75 mm +/- 0.02

**Figure 5 :** aspects et dimensions normalisées.

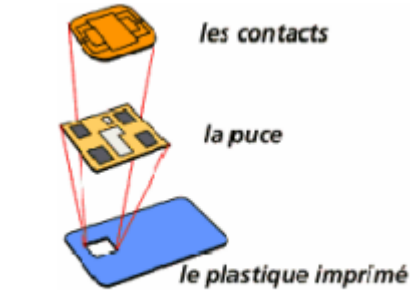



**Figure 4** : les composants d'une carte.

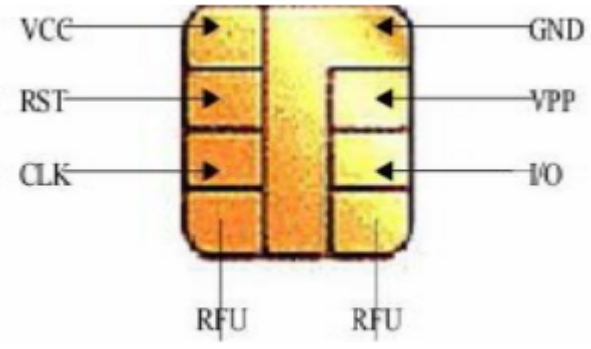



**Figure 6 :** Description des points de contact.

Les cartes à contact doivent être insérées dans un lecteur dans un sens particulier, qui fournit l'alimentation nécessaire. Elles utilisent une communication série sur 8 points de contact selon la norme ISO 7816, mais seules 5 sont généralement utilisées.

- **VCC** : Point d'alimentation, fournissant une tension de 3 à 5 V.
- **RST** : Permet d'initialiser le microprocesseur ; appelé « warm reset » pour réinitialisation sans retirer la carte, et « cold reset » lors d'une coupure de tension.
- **CLK** : Fournit un signal d'horloge à la carte.
- **GND** : Masse, servant de référence de tension à 0 V.
- **I/O** : Utilisé pour la communication entre la carte et le lecteur(half duplexe:communication dans les deux sens).
- **VPP** : Point obsolète sur les modèles modernes, utilisé pour une deuxième source d'alimentation sur les anciens modèles.
- **RFU** : Points réservés pour une utilisation future.

**Puce** : Située sous le module de contact.
**Processeurs** :

- **Processeur central** : Généralement un microcontrôleur 8 bits avec une fréquence d'horloge allant jusqu'à 5 MHz, utilisant des jeux d'instructions comme Motorola 6805 ou Intel 8051, basé sur une architecture RISC.
- **Coprocesseur** : Spécialisé dans des tâches spécifiques, souvent cryptographiques.



**Mémoires** :

- **ROM (Read Only Memory)** : Mémoire non modifiable, utilisée pour stocker des données de manière permanente, typiquement de 32 Ko.
- **EEPROM (Electrical Erasable Programmable Read Only Memory)** : Mémoire modifiable électriquement, conserve les données même hors tension.
- **RAM (Random Access Memory)** : Mémoire volatile utilisée pour le stockage temporaire, perd ses données dès que la carte est hors tension, avec un accès rapide en écriture et en lecture.

**Remarque sur la pile de communication:**

Avec les cartes à puce, la communication entre le terminal et la carte se fait en mode half duplex, ce qui signifie qu'à un instant donné, seule la carte ou le terminal est autorisé à communiquer des données sur le support physique, mais pas les deux en même temps. Ce mécanisme de communication nécessite une synchronisation pour déterminer qui peut communiquer et qui peut écouter. Pour cela, il faut un maître pour initier la communication et un esclave pour la recevoir. Le non-respect de ces règles peut entraîner deux scénarios désastreux :

1. Les deux entités envoient leurs données simultanément sur le support physique, provoquant une collision et la perte des données.
2. Les deux entités écoutent le support physique dans l'attente d'une communication de l'autre, entraînant un blocage indéfini.



La communication entre le terminal et la carte s'effectue à travers une pile protocolaire conforme au modèle OSI :

- **Couche physique** : Normalisée par la norme ISO 7816-3, elle décrit la fréquence d'horloge de 1 MHz à 5 MHz et la vitesse des communications, permettant une transmission binaire.
- **Couche transport** : Également normalisée dans la norme ISO 7816-3, elle gère les échanges de données élémentaires appelés TPDU (Transport Protocol Data Unit). Les protocoles de transport les plus utilisés sont :
    - T=0 : Protocole orienté octet.
    - T=1 : Protocole orienté paquet.
- **Couche application** : Les échanges de données élémentaires pour cette couche sont appelés APDU (Application Protocol Data Unit).

    Il existe deux types d'APDU :

    - **Commandes APDU** : Envoyées par le terminal vers la carte.
    - **Réponses APDU** : Envoyées par la carte vers le terminal.

## I.7. Le cycle de vie d'une carte à puce :

Le cycle de vie d'une carte à puce se décompose en cinq étapes :

1. **Fabrication** : Un code applicatif est inscrit en mémoire ROM, définissant les fonctionnalités de base de la carte, telles que la gestion des communications, l'exécution de commandes, la supervision des programmes, et des fonctions de cryptographie (comme DES, RSA, SHA). La carte intègre également une JVM (Java Virtual Machine) pour exécuter des applets.
2. **Initialisation** : À ce stade, des données spécifiques à l'application sont inscrites dans l'EEPROM, accompagnées de caractéristiques de sécurité.
3. **Personnalisation** : Des données relatives à chaque utilisateur, comme le code PIN, les certificats et les clés de cryptographie, sont enregistrées dans l'EEPROM. Ces secrets doivent rester à l'intérieur de la carte pour assurer la sécurité.
4. **Utilisation** : Après un échange d'APDU, la carte traite les commandes. Si elles sont reconnues, les données peuvent être lues ou écrites en EEPROM, avec un renvoi d'APDU de réponse. Sinon, un code d'erreur est renvoyé.
5. **Mort** : La carte peut devenir obsolète par invalidation logique, saturation mémoire, perte ou vol.



Ce schéma illustre exactement les différentes étapes du processus de la fabrication de la carte:

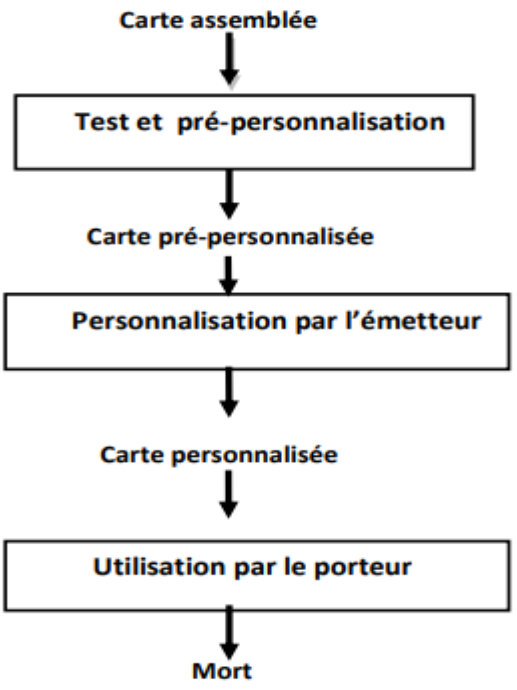



**Figure7 :** processus de fabrication de la carte

## I.8. Communication carte/lecteur :

La communication entre le lecteur et la carte à puce se déroule en mode semi-duplex, permettant à soit le lecteur, soit la carte d'envoyer des données à la fois, mais pas les deux. La carte utilise des paquets APDU (Application Protocol Data Unit) pour échanger des commandes et des réponses. Dans ce modèle maître-esclave, le lecteur, en tant que maître, fournit l'énergie nécessaire, tandis que la carte, en tant qu'esclave, attend toujours une commande APDU de l'hôte, qu'elle exécute avant de répondre.

Les APDU sont échangées comme suit:

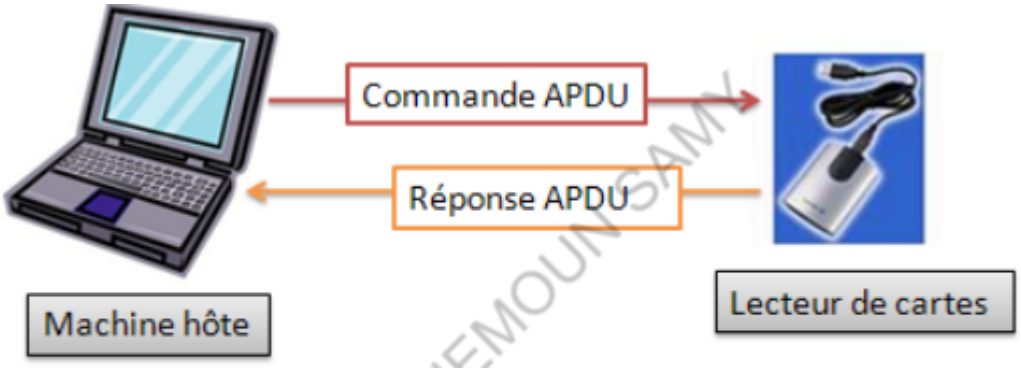





**Figure 8** : ISO 7816-4 : le protocole APDU

### I.8.1. Le protocole APDU :

Le protocole de niveau application de la norme OSI comprend deux types de messages APDU : la commande APDU (C-APDU), envoyée par le CAD vers la carte, et la réponse APDU (R-APDU), renvoyée de la carte au CAD.

**Structure d'une commande APDU :**

| Commande APDU | | | | | | |
|---|---|---|---|---|---|---|
| ENTETE OBLIGATOIRE | | | | | CORPS OPTIONNEL | |
| CLA | INS | P1 | P2 | Lc | Champ de données | Le |

L'en-tête APDU se compose de plusieurs champs :

- **CLA (1 octet)** : classe d'instruction, indiquant la structure et le format pour une catégorie de commandes et de réponses(carte bancaire:0XBC,0X80:porte monnaie électronique,0X10:application SIM).
- **INS (1 octet)** : code d'instruction, spécifiant l'instruction de la commande.
- **P1 (1 octet)** et **P2 (1 octet)** : paramètres de l'instruction.
- **Lc (1 octet)** : nombre d'octets dans le champ de données de la commande(envoyé à la carte pour exécuter l'instruction contenue dans l'entête.
- **Le (1 octet)** spécifie le nombre d'octets attendus dans la réponse APDU(attendu par le terminal).
- **Champ de données** : contient les données à envoyer à la carte, correspondant à la valeur de Lc

**Structure d'une réponse APDU :**

| Réponse APDU | | |
|---|---|---|
| Corps optionnel | Partie obligatoire | |
| Champ de données | SW1 | SW2 |

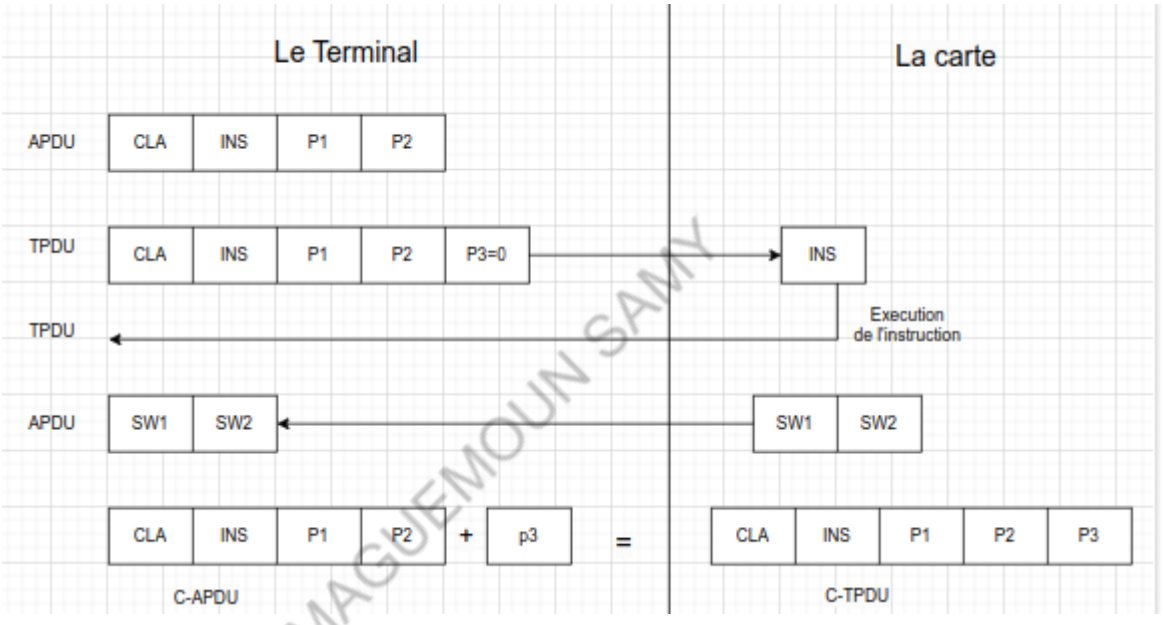


Le champ de données de la réponse APDU a une taille déterminée par **Le** dans la commande APDU correspondante. Les codes d'état (SW1 et SW2) peuvent indiquer différentes situations :

- **0x90 0x00** : Succès
- **0x6E 0x00** : Erreur CLA
- **0x6D 0x00** : Erreur INS
- **0x6B 0x00** : Erreur P1, P2
- **0x67 0x00** : Erreur de longueur (LEN)
- **0x98 0x04** : Mauvais PIN
- **0x98 0x40** : Carte bloquée

**Scénarios de l'échange APDU entre le terminal et la carte:**
Puisque certaines parties des commandes et réponses APDU sont optionnelles ,plusieurs scénarios de communication sont envisageable:

- **Envoie de commandes sans données utiles:**
  Le programme du terminal construit une commande APDU de base, puis la couche transport ajoute un octet supplémentaire P3 (TPDU), souvent invisible au niveau applicatif, qui est mis à 0. Cette commande TPDU est envoyée à la carte, qui répond en renvoyant le code d'instruction au niveau TPDU. On voit donc seulement la commande APDU envoyée et la réponse de la carte après exécution : si tout est correct, le code est 0x9000, sinon un code d'erreur est renvoyé.
  C-TPDU=C-APDU+1octet TPDU

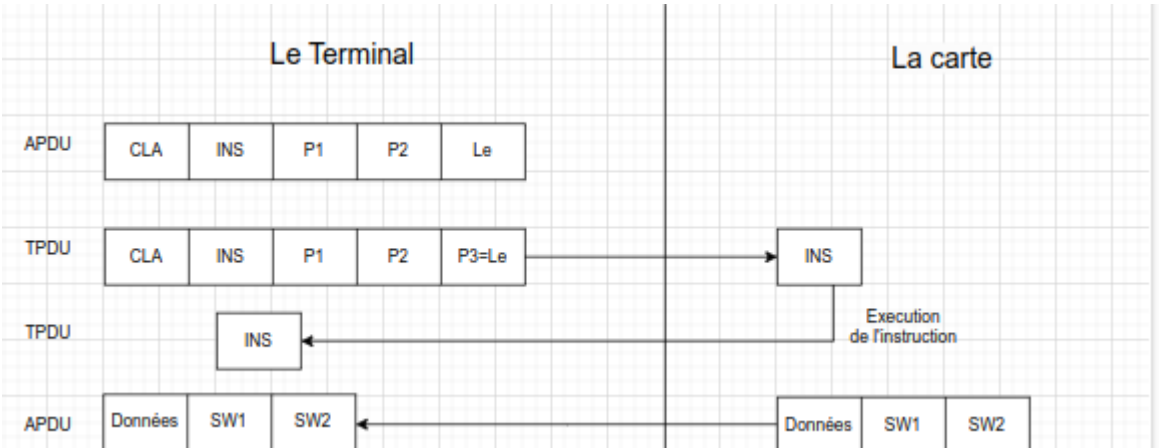




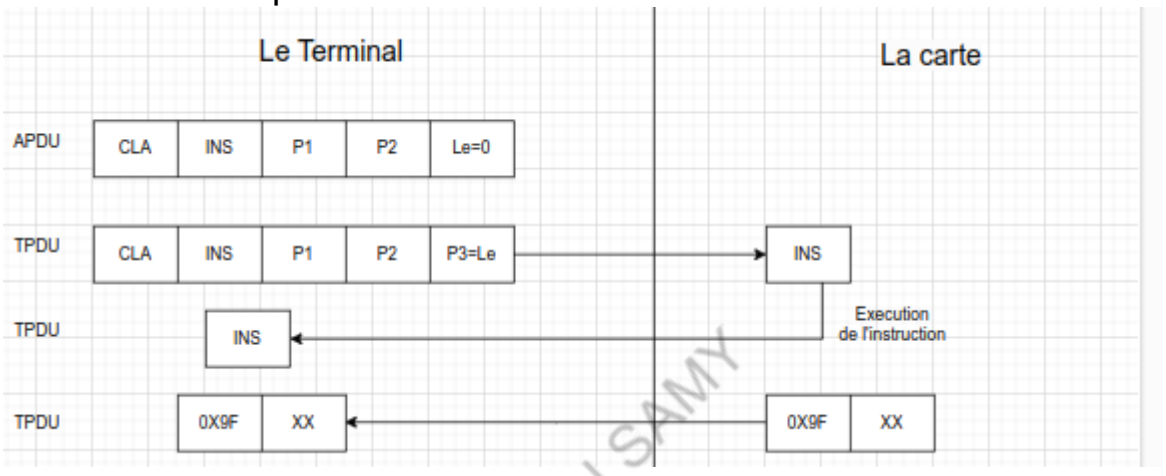

- **Commande avec réception de données utiles:**

  Dans une commande contenant le champ Le, le terminal demande à la carte de renvoyer un certain nombre d'octets (Le). En passant par la couche transport, le paramètre P3 au niveau TPDU est remplacé par Le. La carte exécute alors l'instruction et renvoie le nombre d'octets demandés, suivi du code SW (SW1 et SW2), avec

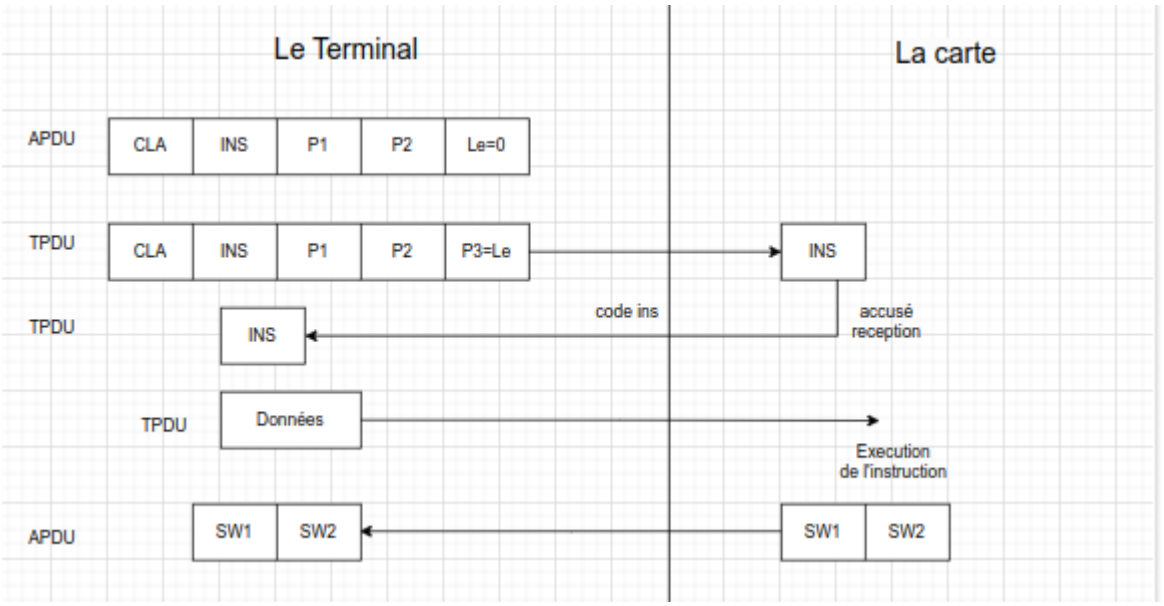
  0x9000 pour le succès ou un code d'erreur si nécessaire.



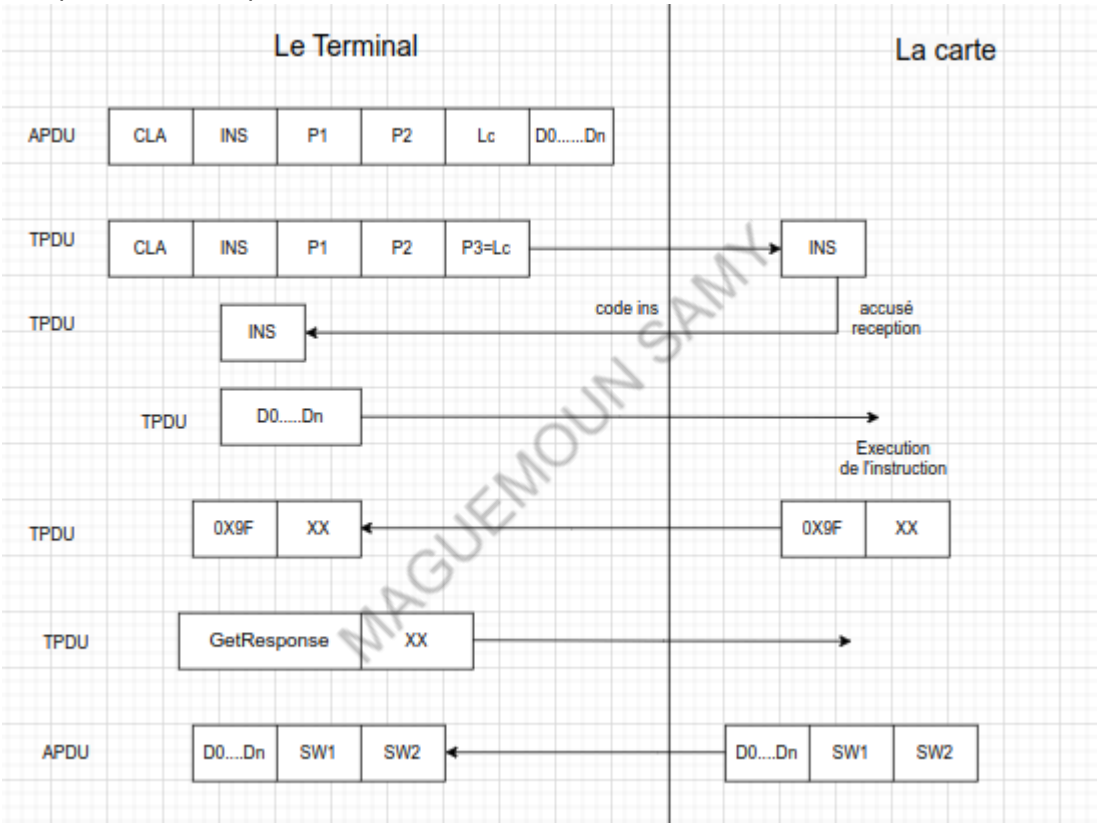
- **Commande avec invitation à lire des données depuis la carte le=0 le nombre d'octets attendu est inconnu:**
  Quand une commande APDU est envoyée avec Le=0, la carte reçoit et exécute l'instruction. Si elle a des données à transmettre, au lieu de renvoyer 0x9000 (succès), elle envoie 0x9FXX pour indiquer qu'elle peut fournir un certain nombre d'octets (XX). Le terminal décide alors s'il invite la carte à envoyer ces données, car la carte n'initie pas l'envoi elle-même.




- **Commande avec envoie de données utiles:**
  Dans ce scénario, la commande APDU inclut des données utiles de taille Lc. Lors de son envoi, la couche transport ajoute le paramètre P3, qui prend la valeur de Lc. La commande est envoyée en deux étapes : d'abord les 5 premiers octets (partie TPDU), que la carte accuse réception, puis les données. La carte exécute ensuite l'instruction demandée et renvoie 0x9000 si tout s'est bien passé ou un code d'erreur en cas de problème.



- **Commande avec envoie et réception de données utiles:**
  Dans ce scénario complexe, une commande APDU avec des données est envoyée au terminal. Après ajout de P3 (remplaçant Lc), la partie TPDU est transmise à la carte pour exécuter l'instruction. Si la carte souhaite renvoyer des données, elle envoie le code 0x9FXX pour informer le terminal de la quantité de données disponibles (XX octets). Le terminal peut alors envoyer une commande GetResponse (0xC0) avec XX en paramètre, invitant la carte à transmettre les données, suivies du Status Word (SW1, SW2).




**Remarque:**
Lors de la réception d'une commande APDU dans le buffer de la carte, seule la partie TPDU arrive en premier. Pour récupérer les données associées, il est nécessaire d'exécuter la commande JavaCard `setIncomingAndReceive`, qui ajoute ces données dans le buffer. Bien que la commande APDU semble complète au niveau de l'application, elle est en réalité transmise en deux blocs via le TPDU : le premier bloc (TPDU) reçu immédiatement, et le second contenant les données, explicitement demandé avec `setIncomingAndReceive`.

## I.9.procedure d'utilisation et de fonctionnement d'une carte à puce:

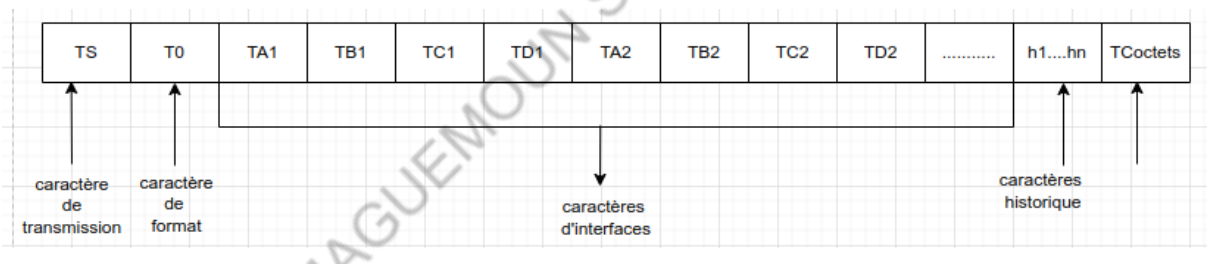
**1/connexion et activation des contacts par le lecteur:**

une fois la carte insérée dans le terminal ,les opérations suivantes sont déclenchés:

-Alimentation de la carte via son entrée VCC.
-Mise à haut niveau de l'entrée RST.
-Génération de l'horloge sur l'entrée CLK.
-mise en mode réception de la ligne I/O.
-Un reset est alors provoqué par le circuit d'interface de la carte.

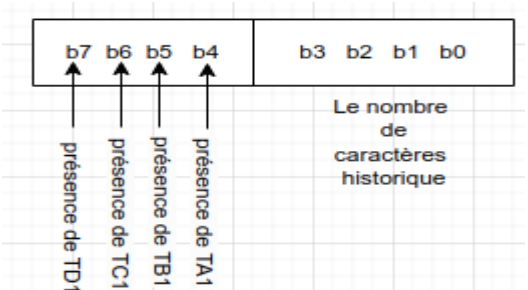
**2/reset de la carte:**

Dès que la carte est mise sous tension, elle envoie un message ATR (Answer to Reset) pour communiquer les paramètres essentiels à l'application cliente pour établir la connexion, comme le protocole de transport, le taux de transmission, et le numéro de série de la puce. L'ATR est une séquence d'octets incluant les paramètres obligatoires TS et TO.

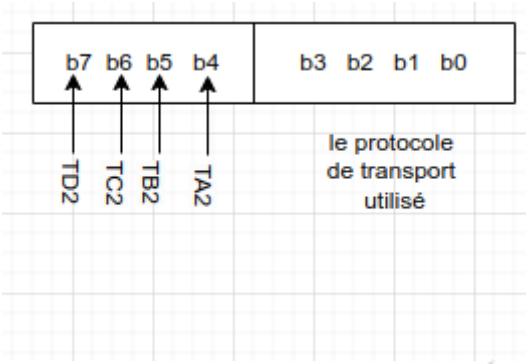





- **TS:** Le caractère de transmission peut être **0x3B** pour une logique directe, c'est-à-dire que le premier bit transmis est le bit de poids fort (bit 7) et le dernier est le bit de poids faible (bit 0). Avec **0x3F**, la logique est inverse : le premier bit transmis est le bit de poids faible (bit 0) et le dernier est le bit de poids fort (bit 7).
- **T0:**Le caractère de format dans l'ATR a deux parties : le quartet de poids fort (b7 à b4) indique quels caractères d'interface sont inclus dans l'ATR :
  b4=1 pour TA1, b5=1 pour TB1, b6=1 pour TC1, et b7=1 pour TD1.
  Le quartet de poids faible (b3 à b0) spécifie le nombre de caractères historiques dans l'ATR, pouvant aller jusqu'à 15.



- **TA1 :** Définit la vitesse de transmission après l’ATR et la fréquence d’horloge maximale.
- **TB1 :** Paramètre pour la programmation de la mémoire EEPROM, indiquant la tension requise pour l'écriture.
- **TC1 :** Temps de garde entre caractères.
- **TD1 :**




  Indique la présence de caractères d'interface supplémentaires (TA2, TB2, etc.) et le type de protocole de transport.
- **TA2, TB2, TC2, TD2 :** Informations supplémentaires sur les paramètres de transfert et protocoles.
- **Caractères historiques :** Fournissent des détails sur le constructeur, le type de carte et le numéro de puce.
- **TCK :** Vérifie l’intégrité des données lorsqu'il est présent dans l'ATR.

**3/Authentification mutuelle et communication:**

Après l'authentification, la carte et le terminal peuvent échanger des commandes et réponses APDU pour communiquer.

**4/Désactivation de l’interface de la carte:**

la désactivation aura lieu lorsque la transaction se termine,le terminal envoie  un message à retirer la carte avant le message de retrait , les opérations suivantes sont réalisés:

-mise au niveau bas du point RST

-mise au niveau bas du point CLK

-mise au niveau inactif de l’entrée I/O

-coupure de l’alimentation VCC

## I.10.Système d'exploitation carte à puce:

Le système d'exploitation de la carte à puce gère :

- L'allocation de mémoire (RAM, EEPROM).
- Les échanges avec l'extérieur via le gestionnaire d'ordres APDU.
- La sécurité (authentification, confidentialité) et les alertes d'attaques potentielles.
- Les anomalies de fréquence d'horloge et de tension d'alimentation.

## I.11.Les systèmes de fichiers:

Les systèmes de fichiers des cartes à puce permettent :

- La création/suppression de fichiers et la gestion des droits de lecture/écriture.
- Une arborescence définie par la norme ISO 7816 avec trois types de fichiers :
    1. **EF (Elementary File)** : Contient les données utiles, au bas de l'arborescence.
    2. **DF (Directory File)** : Répertoires ou sous-répertoires contenant des EF, sans données utiles.
    3. **MF (Master File)** : Répertoire principal unique contenant tous les autres fichiers et répertoires.



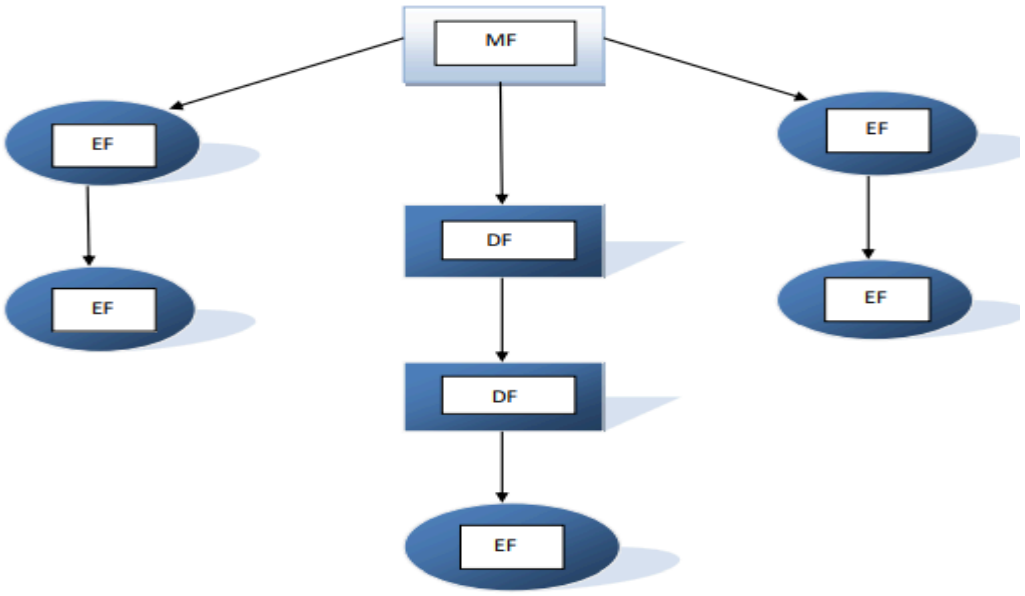


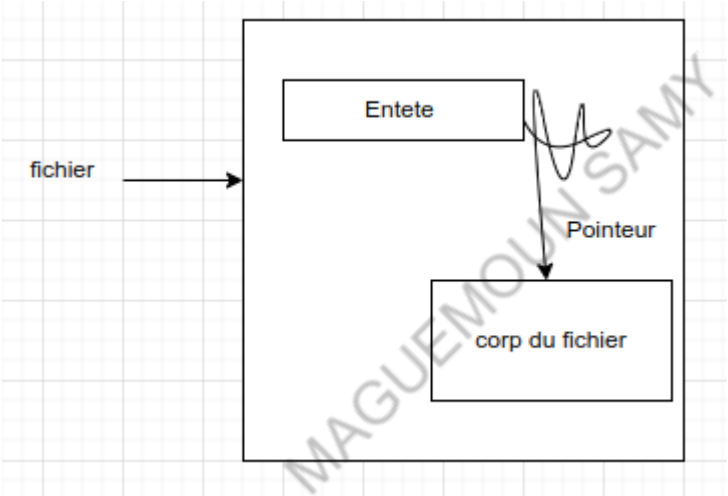

**Figure 9** : structure de système de fichier ISO 7816-4.

### I.11.1.Nommage et identification des fichiers:

Les fichiers (EF, DF) des cartes à puce sont nommés pour y accéder, mais avec des contraintes strictes en raison de la faible mémoire :

- Chaque EF ou DF est référencé par un **FID** (code sur 2 octets).
- Un EF peut aussi être référencé par un chemin constitué de la concaténation des FID depuis le MF.
- Selon la norme :
  - FID du **MF** : `3F00` (utilisé).
  - FID `FFFF` (non utilisé).

### I.11.2./Structure des fichiers:



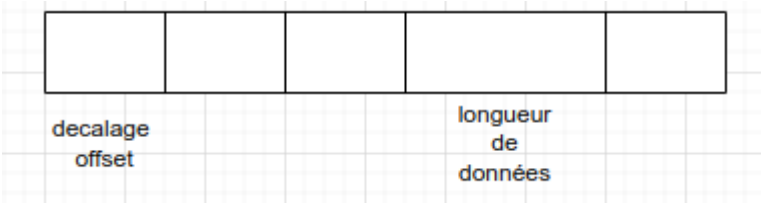

La structure des fichiers dans une carte à puce comporte deux parties :

1. **Zone d'en-tête** : contient le **FID** et les informations sur les autorisations d'accès (lecture/écriture).
2. **Corps du fichier** : contient les données utiles.

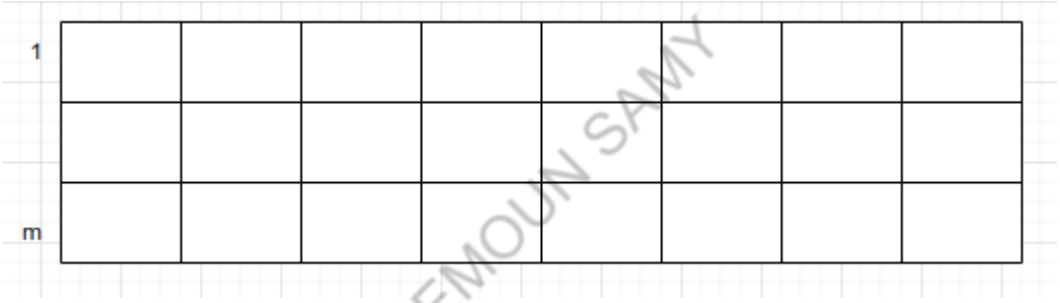
Les deux parties sont liées par un pointeur généré par le système d'exploitation de la carte, ce qui renforce la sécurité : une mauvaise manipulation des données n'affecte ni le fichier ni ses autorisations.

## I.11.3./Les types de structures de fichiers dans une carte à puce :

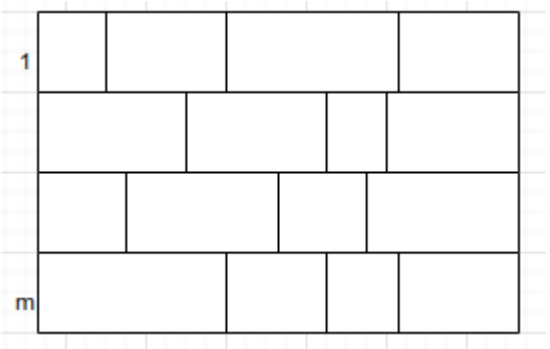
1. **Fichier à structure transparente** : série d'octets accessibles avec deux paramètres :
    - **Décalage (offset)** : position du premier octet.
    - **Longueur** : nombre d'octets de l'enregistrement.




2. **Fichier à structure linéaire fixe** : les enregistrements ont une taille fixe, accessibles par leur **numéro**.




3. **Fichier à structure linéaire variable** : similaire à la structure linéaire fixe, mais avec des enregistrements de **taille variable**.



**Résumé complet : Introduction à la carte à puce**

---

### I.1 Définition

- Une carte à puce est un circuit électronique compact doté d'un microprocesseur et d'une mémoire, utilisée pour stocker et traiter des données.
- Applications principales : cartes bancaires, cartes SIM, cartes de santé.
- **Caractéristiques principales** :
    - **Autonomie** : Faible consommation d'énergie.
    - **Fiabilité** : Résultats corrects et rapides.
    - **Sécurité** : Protection contre les intrusions malveillantes.
    - **Miniaturisation** : Composants compacts, mémoire limitée.
    - **Puissance de calcul** : Suffisante pour éviter une consommation excessive.



---

### I.2 Réalisations industrielles majeures

- **Cartes bancaires françaises (GIE-CB)** : 49 millions de cartes en circulation en 2004 pour 305 milliards d'euros de transactions.
- **Téléphonie mobile (1992-2005)** : 5 millions de cartes SIM vendues.
- **Porte-monnaie électronique (1996-2005)** : 100 millions de cartes distribuées dans 18 pays.
- **Carte SESAM-Vitale** : 45 millions de cartes en circulation en 2004.

---

### I.3 Normes ISO 7816

La norme ISO 7816 couvre les aspects physiques, électriques et fonctionnels des cartes à contact :

- **ISO 7816-1** : Caractéristiques physiques (dimensions, contraintes).
- **ISO 7816-2** : Position des contacts.

- **ISO 7816-3** : Fréquence (1-5 MHz) et vitesse de communication.
- **ISO 7816-4 à 15** : Gestion des fichiers, sécurité, protocoles et cryptographie.

---

## I.4 Types de cartes à puce

#### I.4.1 Cartes à mémoire

- **Simple** : Lecture sans protection, écriture limitée (non réinscriptible), utilisée pour des tâches simples comme les cartes téléphoniques.
- **Logique câblée** : Préprogrammée pour des fonctions spécifiques, gestion des compteurs, rigide et non évolutive.

#### I.4.2 Cartes à microprocesseur

- Les plus avancées, équipées de CPU, cryptographie, et mémoire.
- **Mémoires** :
    - **ROM** : Mémoire persistante, contient le système d'exploitation.
    - **RAM** : Mémoire volatile pour le stockage temporaire.
    - **EEPROM** : Mémoire réinscriptible pour des données évolutives.
- **Processeurs** :
    - **Central (CPU)** : 8 à 32 bits, architecture RISC.
    - **Coprocesseur** : Spécialisé en cryptographie.



#### I.4.3 Cartes à logique câblée

- Mémoire intégrée dans le silicium, utilisée pour des calculs figés (ex. cartes téléphoniques).

#### I.4.4 Cartes à contact

- Nécessitent un lecteur, conformes à ISO 7816, mais sujettes à l'usure.

#### I.4.5 Cartes sans contact

- Communiquent via une antenne pour des transactions rapides (10 cm max), mais coûteuses.

#### I.4.6 Cartes combi

- Combinaison des cartes à contact et sans contact.

---

### I.5 Domaines d'utilisation

Télécommunications, banque, santé, porte-monnaie électronique, transport, contrôle d'accès, identification en ligne.

---

### I.6 Caractéristiques physiques

- **Structure** : Support plastique, module de contact, puce intégrée.
- **Points de contact** :
  - **VCC** : Alimentation (3-5 V).
  - **RST** : Réinitialisation (warm et cold reset).
  - **CLK** : Signal d'horloge.
  - **GND** : Masse (0 V).
  - **I/O** : Communication en half duplex.
  - **VPP** : Obsolète, utilisé pour les anciens modèles.
  - **RFU** : Réservé pour une utilisation future.



---

### I.7 Cycle de vie d'une carte

1. **Fabrication** : Inscription des fonctionnalités de base (ROM, JVM).
2. **Initialisation** : Données spécifiques et sécurité en EEPROM.
3. **Personnalisation** : Configuration utilisateur (PIN, certificats, clés).
4. **Utilisation** : Exécution des commandes APDU.
5. **Mort** : Invalidation, saturation, perte ou vol.

---

### I.8 Communication carte/lecteur

#### I.8.1 Protocole APDU

- **Commande APDU (C-APDU)** :
  - **CLA** : Classe de commande.

    - **INS** : Instruction à exécuter.
    - **P1, P2** : Paramètres de commande.
    - **Lc** : Nombre d'octets à envoyer.
    - **Le** : Nombre d'octets attendus en réponse.
    - **Données** : Informations envoyées à la carte.
- **Réponse APDU (R-APDU)** :
    - **Données** : Résultat de la commande.
    - **SW1, SW2** : Status Word indiquant le résultat :
        - **0x90 0x00** : Succès.
        - **0x6E 0x00** : Erreur CLA.
        - **0x6D 0x00** : Erreur INS.
        - **0x98 0x04** : Mauvais PIN.

---

#### Scénarios d'échange APDU

1. **Commandes sans données** : Renvoi d'un Status Word (SW).
2. **Commandes avec réception de données** : Le terminal demande des données via Le.
3. **Commandes avec `Le=0`** : La carte signale qu'elle peut envoyer des données (0x9FXX).
4. **Commandes avec envoi de données** : La commande contient Lc pour des données.
5. **Commandes avec envoi et réception de données** : Combinaison des deux.



---

### I.9 Procédure d'utilisation

#### Connexion des contacts

- Activation de VCC, RST, CLK, et I/O.

#### Reset de la carte

- Envoi d'un ATR contenant :
    - **TS** : Logique de transmission (directe ou inverse).
    - **T0** : Format (présence de paramètres TA1, TB1, etc.).

    - **TA1, TB1, TC1, TD1** : Informations sur vitesse, tension et protocoles.
    - **Caractères historiques** : Infos constructeur.
    - **TCK** : Vérification d'intégrité.
    - **TA2, TB2, TC2, TD2** : Paramètres supplémentaires pour protocole et sécurité.

#### Authentification et communication

- Échange de commandes et réponses APDU après authentification mutuelle.

#### Désactivation

- Coupure des signaux RST, CLK, I/O, et alimentation VCC.

---

### I.10 Système d'exploitation

Gère :

- La mémoire (RAM, EEPROM).
- Les échanges APDU.
- La sécurité (authentification, alertes d'attaques).
- Les anomalies (fréquence, tension).



---

### I.11 Systèmes de fichiers

#### Types de fichiers

1. **MF (Master File)** : Répertoire principal unique.
2. **DF (Directory File)** : Répertoires contenant des EF.
3. **EF (Elementary File)** : Contiennent les données utiles.

#### Structure des fichiers

- **En-tête** : FID et autorisations d'accès.
- **Corps** : Données utiles.

#### Nommage et identification

- Chaque fichier a un **FID** (2 octets).
- FID du MF : **3F00**, FFFF non utilisé.

**Types de structures**

1. **Transparente** : Accès via décalage et longueur.
2. **Linéaire fixe** : Enregistrements de taille fixe accessibles par numéro.
3. **Linéaire variable** : Enregistrements de taille variable.

---

# Chapitre 2 :Technologie javacard

Les cartes JavaCard permettent d'exécuter des programmes en langage Java, contrairement aux cartes traditionnelles. Pour surmonter les limitations de mémoire, seule une partie du langage Java est supportée, et la machine virtuelle est divisée en deux : une partie s'exécutant dans la carte et l'autre en dehors.

## II.1.L'architecture générale d'une Java Card:

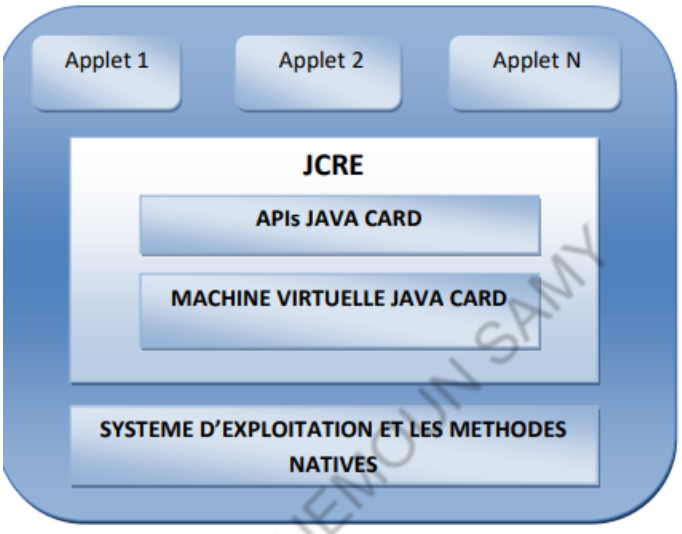



**Figure 1** : structure de la plateforme Java Card

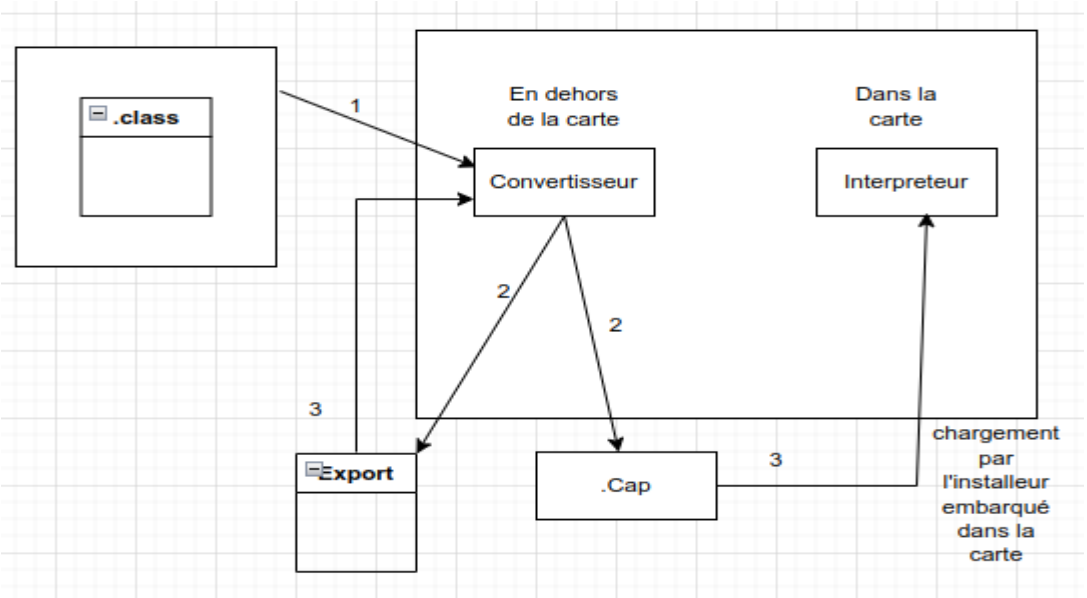


### II.1.1.OS natif:

Le système d'exploitation (SE) de la carte gère l'accès aux ressources (RAM, EEPROM, coprocesseur cryptographique) et est écrit en assembleur, stocké en ROM, donc inaltérable.

### II.1.2.JCVM(Java Card Virtual Machine):

La JCVM, similaire à la JVM, est adaptée aux contraintes de mémoire en étant divisée en deux parties : une externe contenant le convertisseur et une interne embarquée avec l'interpréteur.



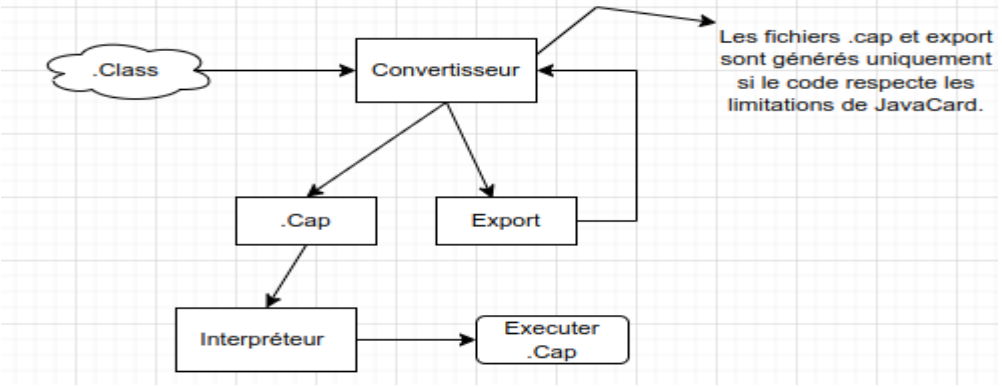

La JCVM, comme la JVM, fournit toutes ses fonctions. Lorsqu'un fichier **.class** est généré à partir du code source **.java**, il doit être converti pour la JCVM.



### II.1.2.1.Le convertisseur:

Il produit :

- **.cap** : Fichier binaire exécutable chargé dans la carte pour être interprété.
- **Export** : Fichiers utilisés par le convertisseur pour la vérification et l'édition des liens, mais non chargés dans la carte.



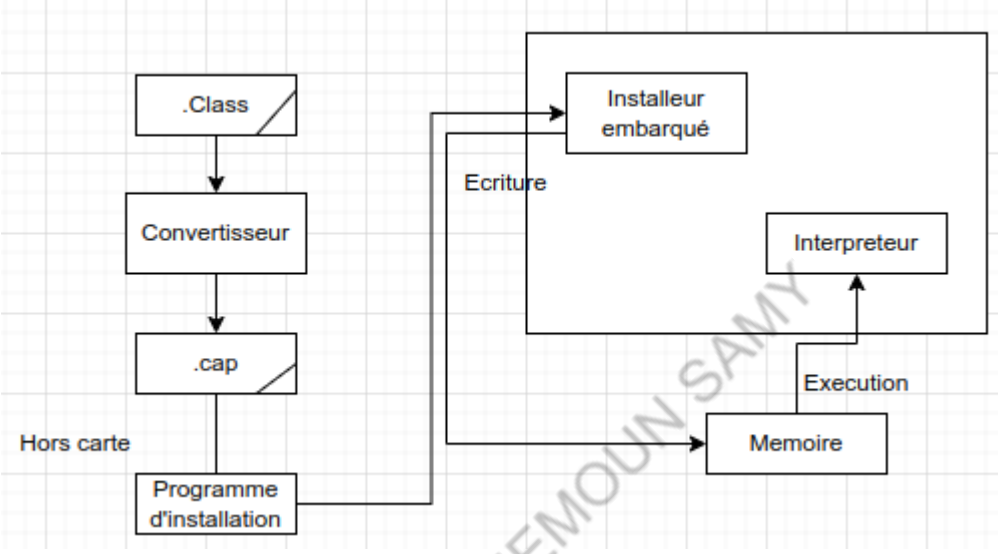

### II.1.2.2.L'interpréteur:

embarqué dans la carte, exécute les fichiers **.cap** et permet l'indépendance des applets vis-à-vis du matériel. Il gère également le

partage de données entre applets : les données communes sont accessibles à toutes, tandis que les données privées restent protégées.

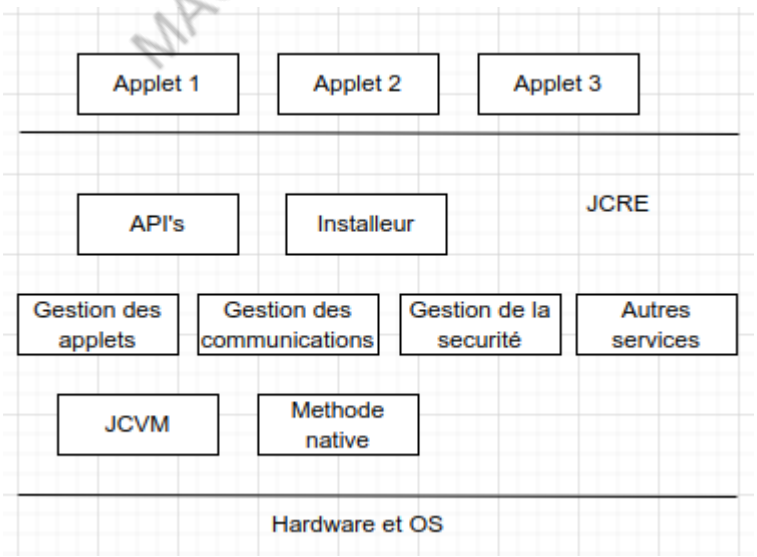
### II.1.2.3.L'installeur:

L'interpréteur n'est pas chargé de charger les fichiers **.cap**, il exécute uniquement leur contenu. Le **mécanisme d'installation**, appelé **installeur**, se trouve dans la carte et coopère avec un programme externe qui lui transmet le fichier **.cap**. L'installeur écrit le fichier binaire dans la mémoire de la carte, où il sera exécuté par l'interpréteur.






## II.1.3.L'environnement JavaCard (JCRE) :




JCRE:Regroupe les composants du système JavaCard dans la carte, gérant les applets, communications, transactions et sécurité. Il inclut la JCVM, l'installateur, les API JavaCard et les classes système du JCRE.

## II.1.4.Les API's JAVACARD:

Ensemble de classes spécialisées pour la programmation des cartes à puce selon la norme ISO 7816.

- **Packages principaux** : `java.lang`, `javacard.security`, `javacard.framework`.
- **Package d'extension** : `javacardx.crypto`.

**Solutions pour la contrainte mémoire** :

- Découper la machine virtuelle Java en deux parties (dans la carte et en dehors).
- Supporter uniquement un sous-ensemble du langage Java.

**Fonctionnalités supportées** :

- Types simples : `boolean`, `byte`, `short`.
- Tableaux à une dimension.
- Exceptions, héritage, et création dynamique d'objets.



**Fonctionnalités non supportées** :

- Types de grande taille : `double`, `float`, `long`.
- Tableaux à deux dimensions, chaînes de caractères, chargement dynamique des classes.

**Applet JavaCard** :
Programme écrit avec un sous-ensemble de Java pour les cartes à puce.
**Étapes de création** :

**1/Ajouter l'API JavaCard** : Ensemble de classes spécialisées pour programmer des cartes à puce conformes à la norme ISO 7816.

- **`javacard.framework`** : Package principal contenant :
  - **Classe `Applet`** : Superclasse de toutes les applets résidentes.
  - **Classe `APDU`** : Gère l'encapsulation des commandes APDU.
  - **Classe `ISOException`** : Gère les exceptions.

- **Classe `ISO7816`** : Définit les mots d'état pour signaler les erreurs courantes.

**2/Déclaration des constantes et des attributs utilisés**.

**3/méthodes obligatoires des applets JavaCard:**

**Méthode `install()`** :

- Appelée lors de l'installation de l'applet sur la carte.
- Crée une instance de l'applet et l'enregistre auprès du JCRE via `register()`.
- Une fois installée, l'applet reste inactive jusqu'à sa sélection.

**Méthode `process()`** :

- Méthode principale de l'applet pour traiter les commandes APDU (C-APDU).
- Interprète et exécute les instructions spécifiées dans les C-APDU.
- Renvoie le résultat dans une R-APDU après l'exécution.



**Méthode `select()`** :

- Active une applet lorsque le JCRE reçoit une commande **Select APDU**.
- Retourne :
  - `false` : Sélection échouée, le JCRE envoie un message d'erreur.
  - `true` : Sélection réussie, les C-APDU sont transmises à la méthode `process()`.

**Méthode `deselect()`** :

- Désélectionne l'applet courante pour permettre au JCRE d'en sélectionner une autre.

**Méthodes pour traiter une C-APDU :**

1. `getBuffer()` :
    - Accède au tampon (buffer) contenant la commande C-APDU sous forme d'un tableau d'octets.
    - Stocké dans la RAM.
2. `setIncomingAndReceive()` :
    - Reçoit les données entrantes si la C-APDU contient des données utiles.
3. `setOutgoingAndSend()` :
    - Exécute la C-APDU et envoie la réponse R-APDU contenant le résultat de l'exécution.

**4/Status word(SW1 et SW2):**

Le **Status Word (SW1 et SW2)** est généré automatiquement par le JCRE :

- **0x90 00** : Indique que tout s'est bien passé.
- En cas d'erreur : Renvoie un code correspondant à l'erreur.

**Résumé complet : Chapitre 2 - Technologie JavaCard**

---

**Introduction**

Les cartes JavaCard exécutent des programmes écrits dans un sous-ensemble du langage Java. Pour gérer les contraintes de mémoire, la machine virtuelle est divisée en deux parties : une interne (embarquée) et une externe.

---

## II.1 Architecture générale d'une JavaCard

### II.1.1 OS natif

- Gère l'accès aux ressources (RAM, EEPROM, coprocesseur cryptographique).
- Écrit en assembleur et stocké en ROM, il est inaltérable.

### II.1.2 JCVM (Java Card Virtual Machine)

- Machine virtuelle adaptée aux cartes à puce, divisée en deux parties :
  - **Externe** : Inclut le convertisseur, qui produit des fichiers compatibles avec la JCVM.
  - **Interne** : Embarquée dans la carte, avec l'interpréteur qui exécute les fichiers `.cap`.



**Composants de la JCVM** :

1. **Le convertisseur** :
   - Transforme le code Java en fichiers adaptés à la JCVM.
   - Produit :
     - **Fichier .cap** : Contient le bytecode optimisé pour la carte.
     - **Fichier export** : Vérifie et lie les applets, mais n'est pas chargé dans la carte.
   - Ces fichiers sont générés uniquement si le code respecte les limitations de JavaCard.
2. **L'interpréteur** :

    - Situé dans la carte, il exécute les fichiers `.cap`.
    - Gère l'indépendance des applets vis-à-vis du matériel et le partage des données (communes ou privées).
3. **L'installeur** :
    - Présent dans la carte, il coopère avec un programme externe pour recevoir les fichiers `.cap`.
    - Écrit les fichiers `.cap` dans la mémoire de la carte, où ils seront exécutés par l'interpréteur.

---

#### II.1.3 JCRE (Java Card Runtime Environment)

- Environnement regroupant tous les composants JavaCard dans la carte.
- **Responsabilités** : Gestion des applets, communications, transactions, sécurité.
- **Contient** : JCVM, installeur, API JavaCard, et classes système du JCRE.



---

#### II.1.4 API JavaCard

- Ensemble de classes pour programmer les cartes selon la norme ISO 7816.
- **Packages principaux** :
    - `java.lang`, `javacard.security`, `javacard.framework`.
    - Extension : `javacardx.crypto`.
- **Solutions aux contraintes de mémoire** :
    - Découpage de la machine virtuelle (interne et externe).
    - Support limité à un sous-ensemble du langage Java.

---

### Fonctionnalités supportées et non supportées

#### Supportées

- Types simples : `boolean`, `byte`, `short`.
- Tableaux à une dimension.
- Exceptions, héritage, création dynamique d'objets.

#### Non supportées

- Types de grande taille : `double`, `float`, `long`.
- Tableaux à deux dimensions.
- Chaînes de caractères et chargement dynamique des classes.

---

### Applet JavaCard

Programme écrit dans un sous-ensemble de Java pour les cartes à puce.



#### Étapes de création d'une applet

1. **Ajouter l'API JavaCard** : Contient des classes clés, notamment :
    - **`Applet`** : Superclasse de toutes les applets.
    - **`APDU`** : Gère les commandes APDU.
    - **`ISOException`** : Gère les exceptions.
    - **`ISO7816`** : Définit les codes d'état pour signaler les erreurs.
2. **Déclarer les constantes et attributs utilisés.**
3. **Implémenter les méthodes obligatoires** :
    - **`install()`** : Installe l'applet, crée une instance et l'enregistre auprès du JCRE.
    - **`process()`** : Traite les commandes APDU et renvoie les résultats dans une R-APDU.
    - **`select()`** : Active une applet. Retourne :
        - `false` si la sélection échoue (erreur).
        - `true` si la sélection réussit (les C-APDU sont traitées).
    - **`deselect()`** : Désélectionne l'applet courante.

#### Méthodes pour traiter une C-APDU

- `getBuffer()` : Accède au tampon contenant la commande APDU.
- `setIncomingAndReceive()` : Reçoit les données si la commande contient un champ de données.
- `setOutgoingAndSend()` : Exécute la commande et envoie une réponse APDU.

---

### Status Word (SW1 et SW2)

- **0x90 00** : Succès.
- En cas d'erreur, renvoie un code correspondant (ex. : mauvais PIN ou carte bloquée).

**Résumé : Chapitre 3 - Sécurité des systèmes embarqués**

---

**Introduction à la sécurité des systèmes embarqués**

La sécurité des systèmes embarqués repose sur trois aspects principaux :

1. **Sécurité matérielle** : Protection contre les attaques physiques.
2. **Sécurité logicielle** : Défense contre les logiciels malveillants.
3. **Protection des communications** : Garantir la confidentialité, l'intégrité et la disponibilité des données échangées.

**Principes fondamentaux** :

- **Confidentialité** : Empêcher l'accès non autorisé aux données.
- **Intégrité** : Préserver les données contre les modifications ou suppressions non autorisées.
- **Disponibilité** : Assurer l'accessibilité du système malgré les attaques de surcharge (ex. : DDoS).
- **Authentification** : Identifier les entités et retracer leurs actions ("qui fait quoi à quel moment").



**III.1 Causes de vulnérabilité des systèmes embarqués**

1. **Complexité de conception** :
    - Composants sécurisés individuellement, mais leur combinaison peut introduire des vulnérabilités.
    - Risques liés à l'origine diversifiée des composants (multiples pays).
2. **Environnement non fiable** :
    - Exposition à des environnements à risque.
3. **Exécution de logiciels malveillants**.

**III.2 Types d'attaques**

**III.2.1 Attaques logicielles**

- Exploitent des failles dans les applications embarquées pour corrompre ou détourner leur fonctionnement.

#### III.2.2 Attaques physiques

##### 1. Invasives

- **Caractéristiques** :
    - Nécessitent des coûts élevés et des connaissances approfondies de l'architecture de la carte.
    - Reposent sur des technologies de rétro-ingénierie comme la reconstitution du circuit via FIB (Focused Ion Beam).
- **Objectif** : espionner ou altérer les communications via des sondes,modifier les données avec un laser.

##### 2. Non invasives

- **Caractéristiques** :
- Peu coûteuses, non destructrices et discrètes.
- Invisibles pour le détenteur de la carte.
    - Peu coûteuses, non destructrices et discrètes.
    - Invisibles pour le détenteur de la carte.
- **Exemples** : Espionner ou modifier les informations circulant dans la carte.



##### Exemple d'attaque : Injection de faute

- **Principe** : Perturber le fonctionnement d'une partie du système pour contourner les protections.
- **Outils** :
    - **Piques d'alimentation** : Perturbent les déclencheurs de calcul en manipulant la tension d'entrée.
    - **Attaques optiques** : Utilisent une lumière spécifique pour inverser le contenu d'une mémoire.
    - **Champs électromagnétiques** : Manipulent des bits en mémoire pour corrompre les données.

### III.3 Méthode de débit avec injection de faute

**Situation normale :**

```
private void debit (APDU apdu) {
    ifeq (0x99) {  // Condition pour valider le PIN
        effectuer l'opération de débit;
    } else {
ISOException.throwit(SW_PIN_VERIFICATION_REQUIRED);
    }
}
```

**Après injection de faute :**

- Le champ électromagnétique modifie l'instruction `ifeq` (0x99) en `NOP` (0x00).
- Cela rend la condition invalide, permettant l'exécution directe de l'opération de débit sans validation du PIN.

**La méthode devient après attaque :**

```
private void debit (APDU apdu) {
    effectuer l'opération de débit;

ISOException.throwit(SW_PIN_VERIFICATION_REQUIRED);
// Levée d'exception trop tard.}
```

- **Conséquences** : L'utilisateur malveillant effectue le débit, et l'exception n'est levée qu'après l'opération, rendant l'attaque efficace.



### Conclusion

- Les attaques logicielles ciblent les vulnérabilités des applications pour les corrompre.
- Les attaques physiques exploitent les faiblesses matérielles ou perturbent le fonctionnement pour contourner les protections.

- L’injection de faute est un exemple concret de manipulation d’un système embarqué, permettant un contournement des mécanismes de sécurité.