

Java CardTM et la sécurité.



Damien SAUVERON
sauveron@labri.fr

<http://www.labri.fr/~sauveron/>

Plan

1) La carte à puce

- Présentation
- Sa sécurité
- Les attaques

2) Java Card

- Présentation
- Sa sécurité
- Les attaques

3) Certification

- Présentation
- Problèmes

4) Le projet “Sécurité Java Card”

- Contexte
- JCatools

5) Le projet “Java Cards Grid”

6) Conclusions

La carte à puce

—

Présentation

Qu'est ce qu'une carte à puce ?

- ☞ *un morceau de plastique* de la taille d'une carte de crédit
- ☞ *un circuit électronique* capable de manipuler (stocker, calculer, etc) des informations

Historique

- En 1968, deux Allemands Jürgen DETHLOFF et Helmut GRÖTRUPP introduisent un circuit intégré dans une carte plastique
- Entre 1974 et 1978, le français Roland MORENO, *le père de la carte à puce* dépose 47 brevets dans 11 pays
- En 1983, apparition des premières cartes téléphoniques à mémoire
- En 1984, adoption par le G.I.E carte bancaire de la “ carte bleue ”
- Entre 1984 et 1987, normes ISO 7816 (carte à puce à contact)
- En 1997, apparition des premières Java Cards

Deux classements possibles

les cartes à mémoire



versus

les cartes à microprocesseur

les cartes à contact

versus



les cartes sans contact

versus

les cartes dual-interface

La carte à mémoire

- ☞ Premier modèle de cartes à puce
- ☞ Majorité des cartes vendues dans le monde en 1999

Elle possède :

- ☞ *une puce mémoire* de 1 à 4 Ko
- ☞ *une logique câblée non programmable*

Avantages :

- sa technologie simple
- son faible coût (1\$)

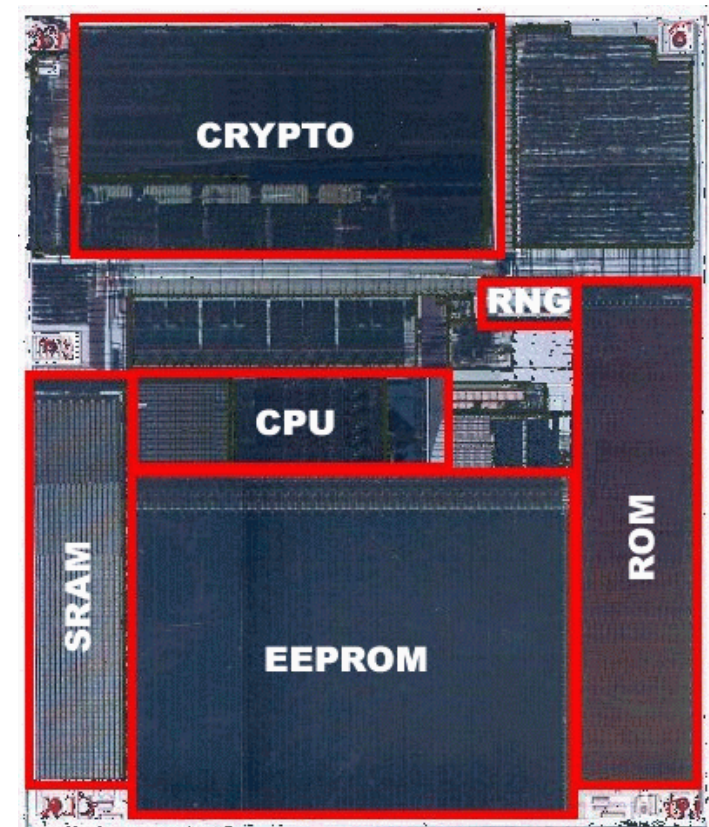
Inconvénients :

- sa dépendance vis-à-vis du lecteur de carte
- “assez” facile à dupliquer

La carte à microprocesseur

Taille de la puce : $25\text{mm}^2 * 200\mu\text{m}$

- ➡ **Microprocesseur** (CPU) : 8, 16 ou 32 bits (à architecture RISC ou pas)
- ➡ **ROM** : 16 à 64 Ko
- ➡ **EEPROM** : 16 à 64 Ko
- ➡ **RAM** : 1 à 2 Ko
- ➡ Coprocesseur cryptographique
- ➡ Générateur de nombres aléatoires (RNG)

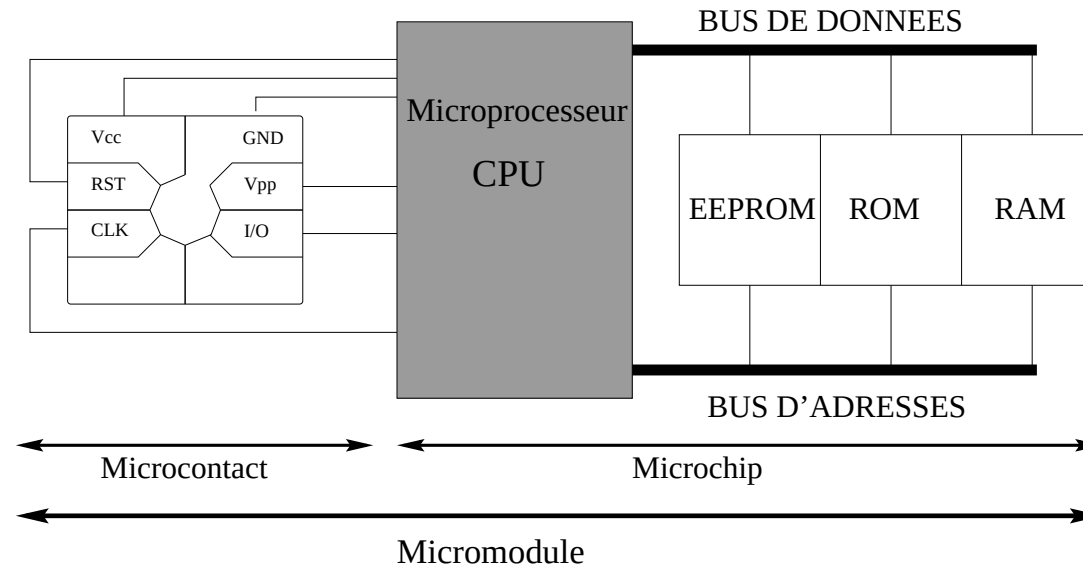


Avantage : le coût acceptable pour tant de sécurité (entre 1\$ et 20\$).

Les modes de communication

La carte à contact (1/2)

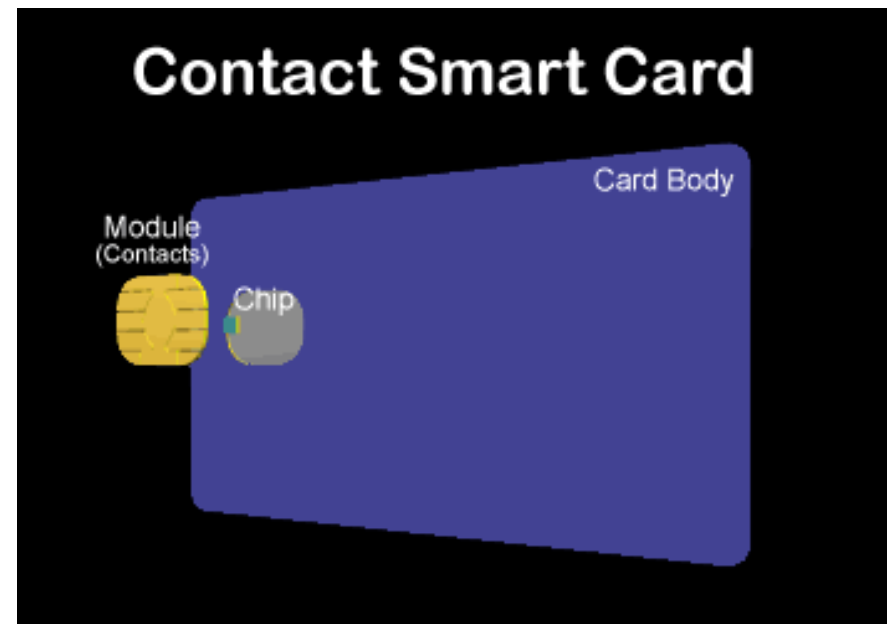
- ➡ Suit le standard *ISO 7816*
- ➡ *Communication série via huit contacts* $\Rightarrow$ insertion dans un lecteur de carte



La carte à contact (2/2)

Problèmes :

- l'insertion et le retrait sont des facteurs d'usure de la carte
- orientation de la carte dans le lecteur

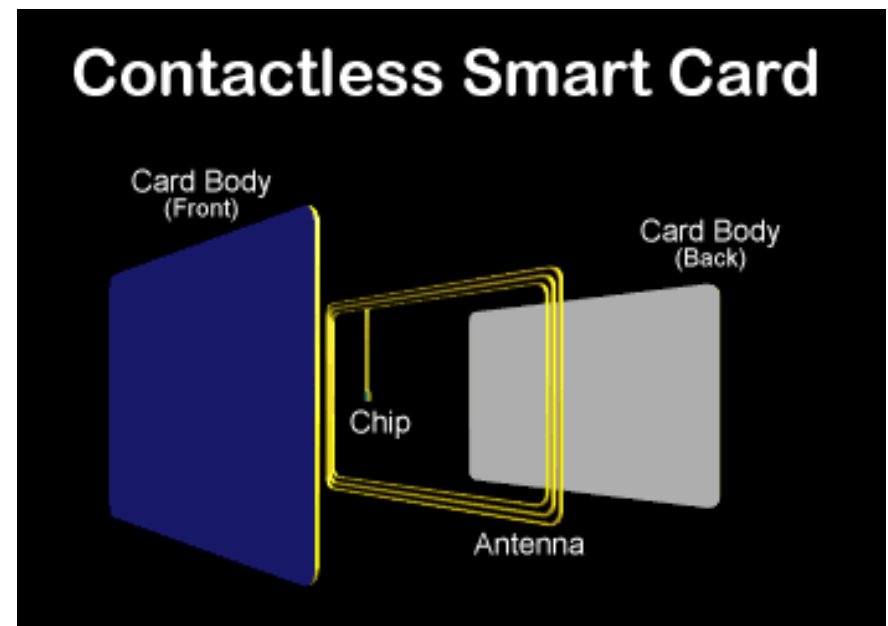


La carte sans contact

- ➡ *Communication via une antenne* dans la carte
- ➡ Récupère son énergie d'un couplage capacitif ou d'un couplage inductif
- ➡ Suit le standard *ISO 1443*

Problèmes :

- distance de communication limitée (environ 10 cm)
- temps de transaction est de l'ordre de 200 ms $\Rightarrow$ limite les données à échanger
- le coût élevé



La carte dual-interface

C'est une combinaison entre :

- ☞ la carte à contact
- ☞ et la carte sans contact

Ces deux possibilités de communication en font une carte “idéale”.

La carte à puce

—

Sa sécurité

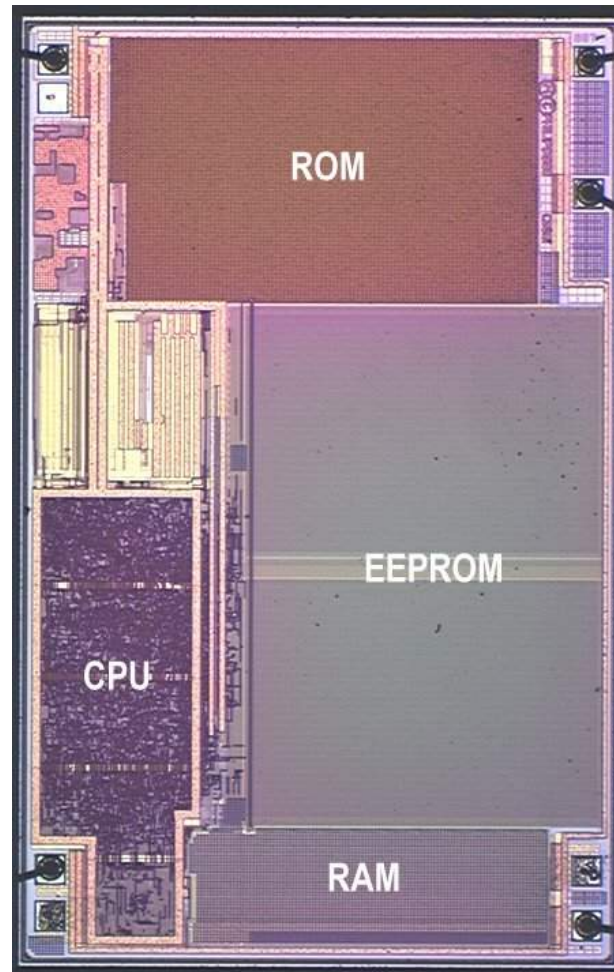
La sécurité physique

- ☞ techniques d'impression sophistiquées
 - différentes couches d'impression
 - hologramme
 - embossage
- ☞ procédés d'encollage
- ☞ matériaux plastiques
- ☞ design du micro-contact

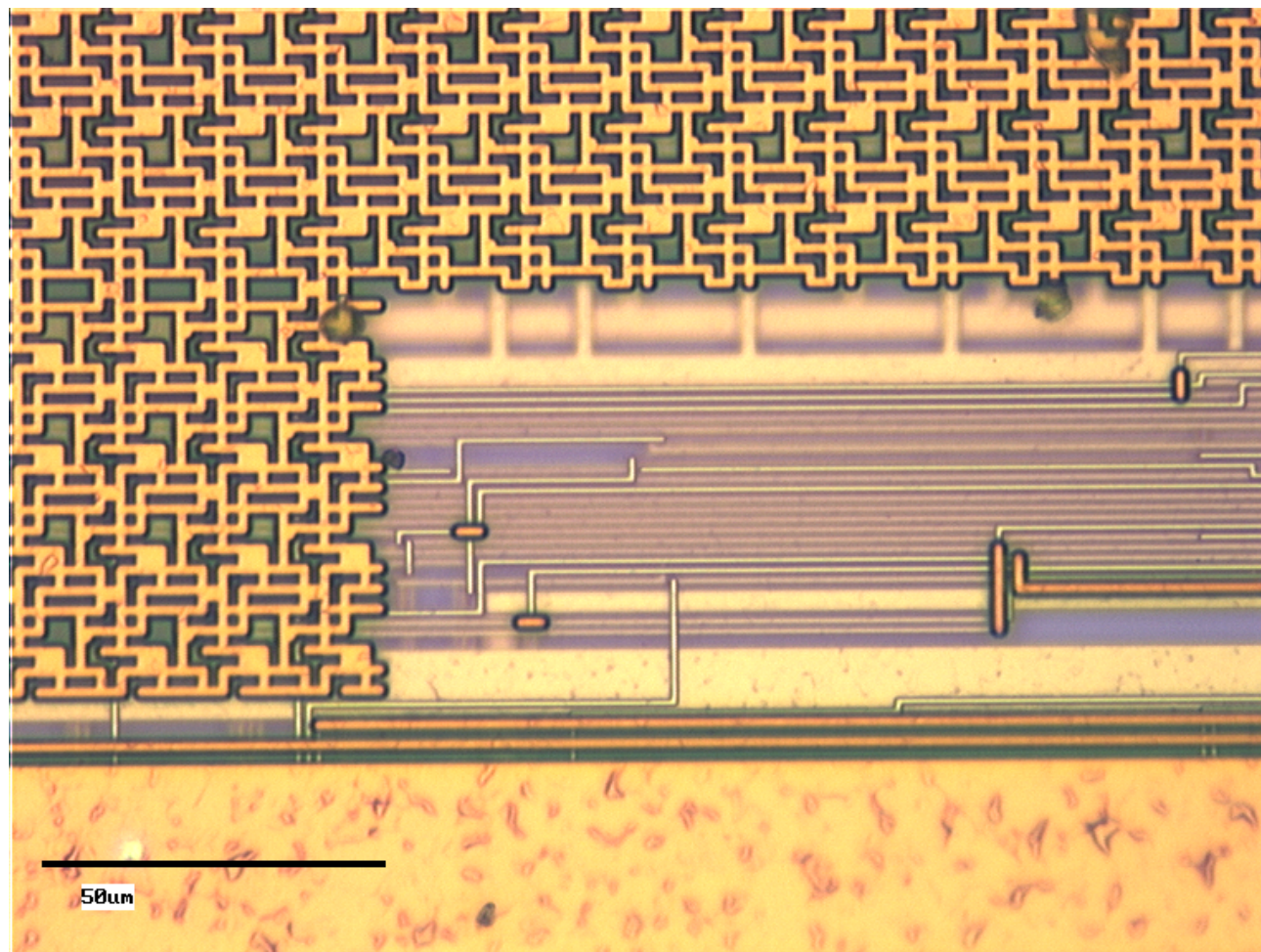
La sécurité hardware

- ➡ un numéro de série unique
- ➡ l'utilisation de mémoire de type PROM
- ➡ blindage physique du composant (grille, grille active, ...)
- ➡ des détecteurs de conditions anormales (température, lumière, ...)
- ➡ brouillage des informations dans le composant (mémoire, bus, ...)
- ➡ co-processeurs cryptographiques
- ➡ pompe de charge pour le lissage de la consommation

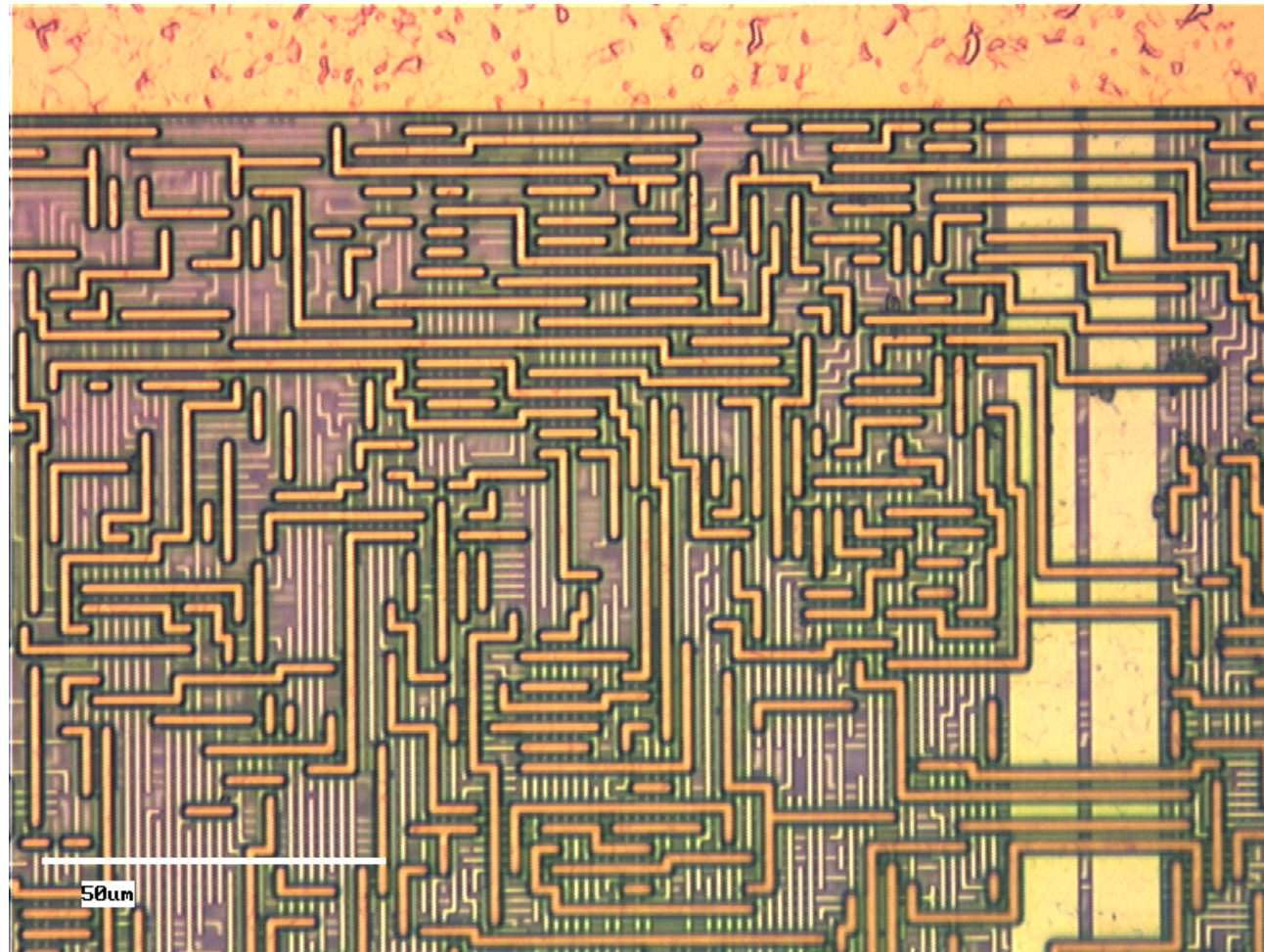
Une puce



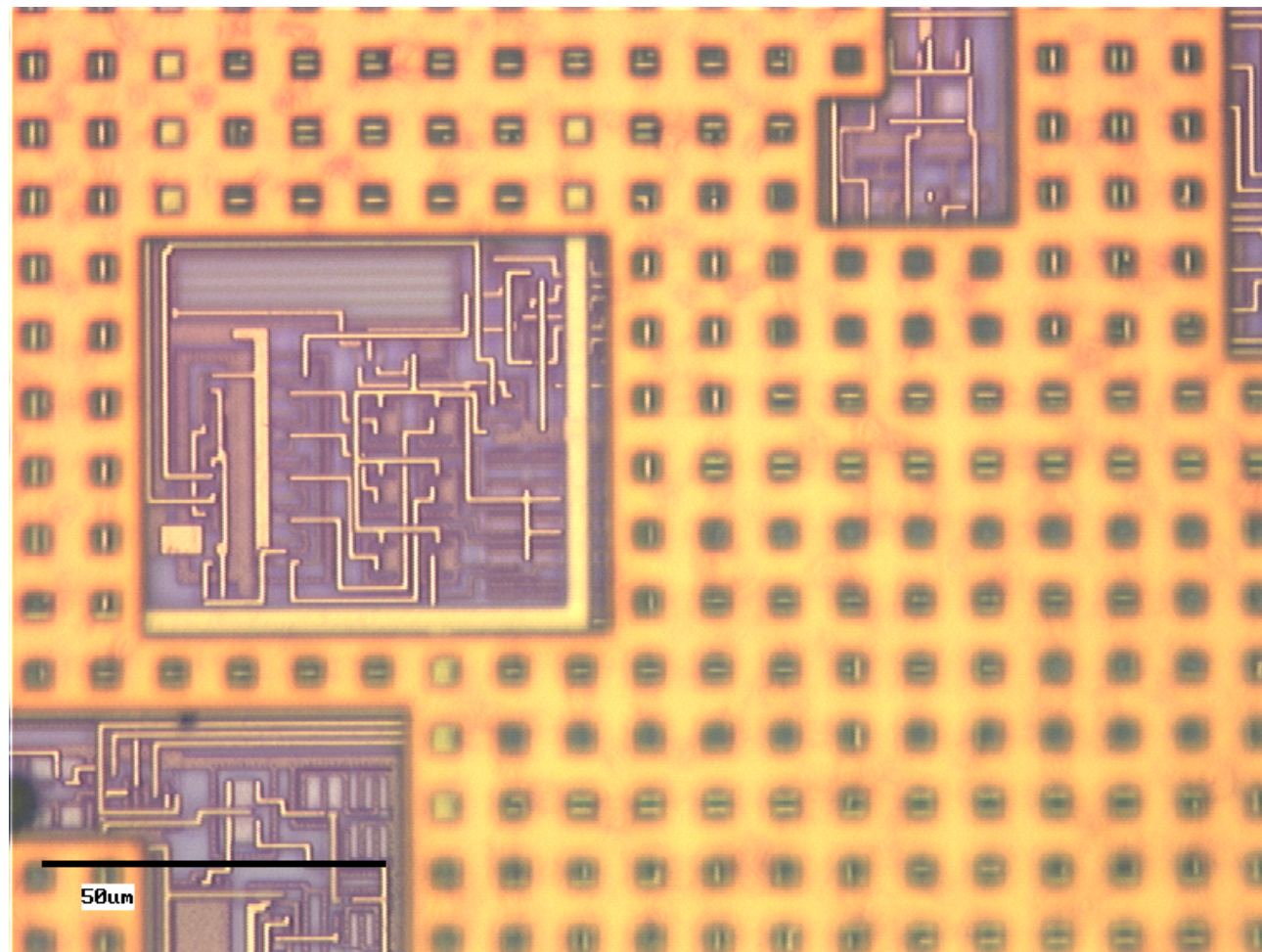
La grille de protection



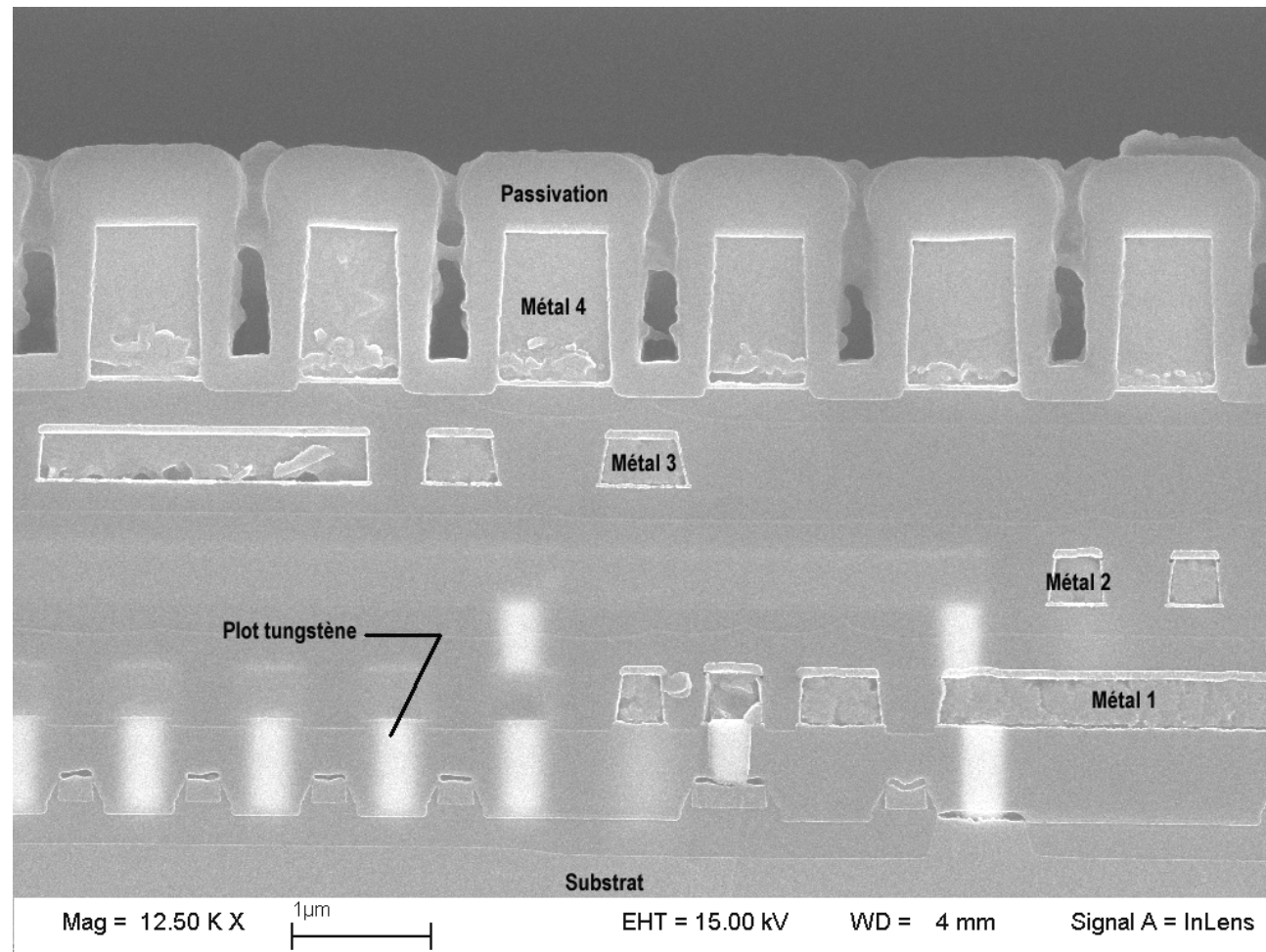
Le CPU



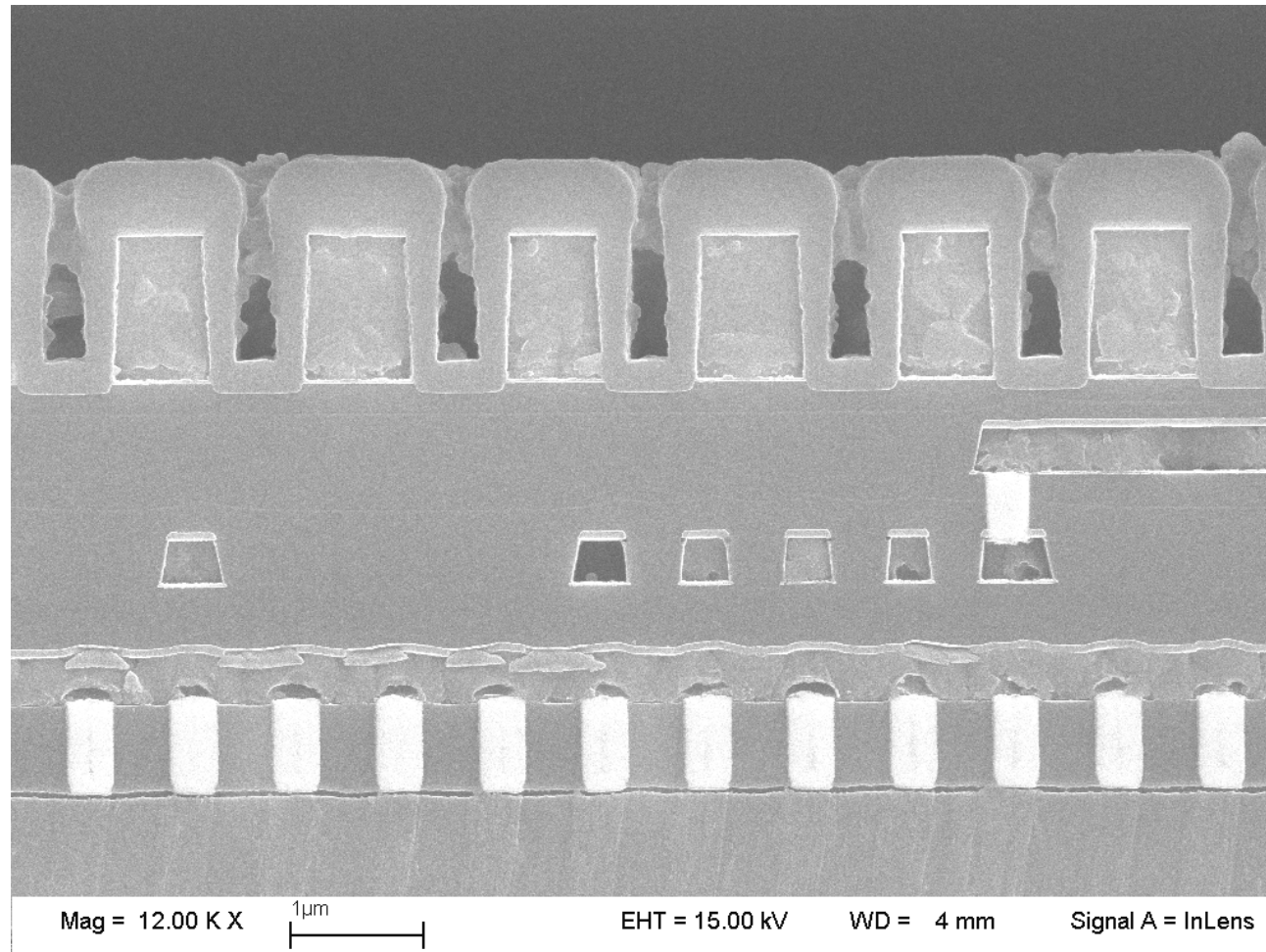
Une partie analogique



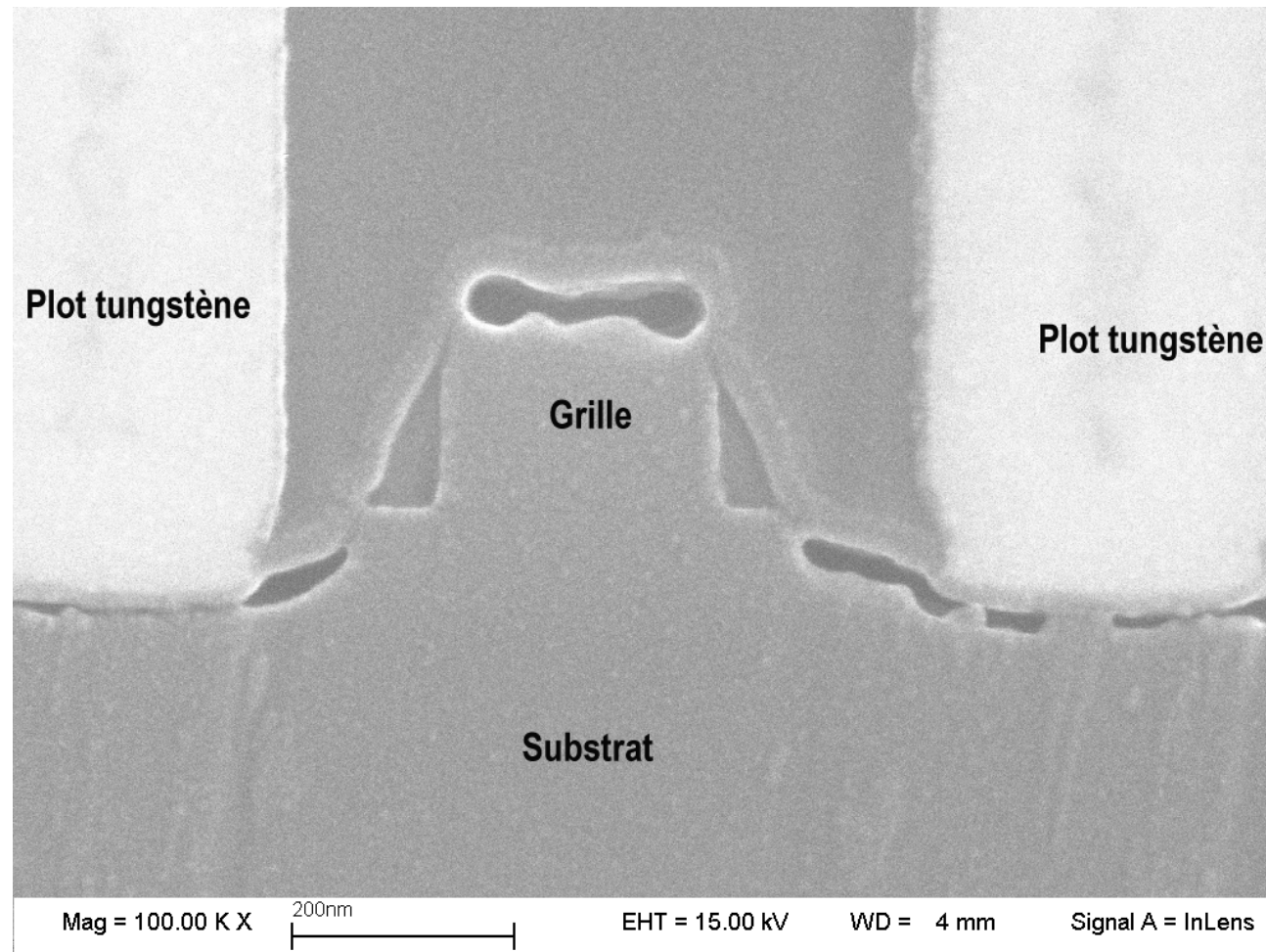
Une coupe de la puce



Une coupe de la puce



Une coupe d'un transistor



La sécurité software

- ☞ contrôles d'accès aux données
- ☞ maintien de l'intégrité des données
- ☞ entrées/sorties sécurisées
- ☞ migration du code

La sécurité de l'env. de production

- ☞ physiquement sécurisé
- ☞ régulièrement audité

Pour résumer : la triple alliance électronique, informatique et cryptographie assure un très haut degré de sécurité

La carte à puce

—

Les attaques

Non invasives

- ➡ modification des conditions opérationnelles (V_{cc} , F)
- ➡ modification de la température
- ➡ modification des rayonnements lumineux (UV, rayon X, lumière blanche, IR, ...)
- ➡ injection de fautes (glitches, rayonnements lumineux)
(injection de fautes sur la JVM :
<http://www.cs.princeton.edu/~sudhakar/papers/>)
- ➡ attaques sur les canaux cachés

Les canaux cachés

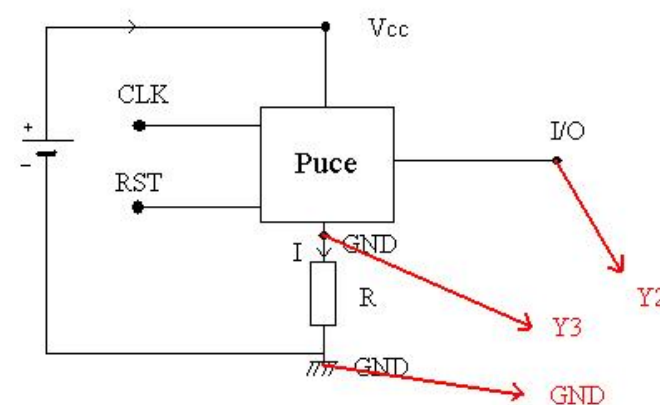
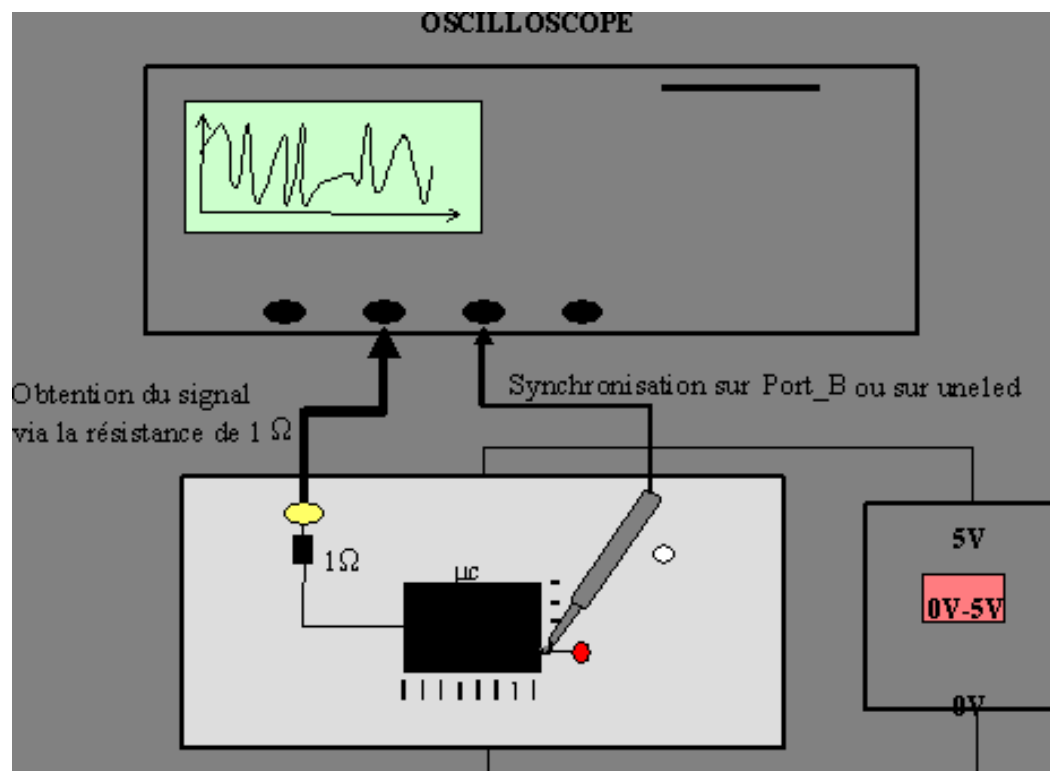
- ☞ le temps d'exécution
 - ⇒ nombre de cycle d'une instruction ou d'un algorithme.
- ☞ la consommation de courant
 - ⇒ Les modifications rapides de la tension et de l'intensité du courant au sein du même composant sont à la base des émissions du circuit car ils conduisent des courants RF à l'intérieur et à l'extérieur du chip.
- ☞ les émissions électromagnétiques
 - ⇒ Les courants RF entraînent un rayonnement électromagnétique.

Le matériel

- ☞ un oscilloscope numérique,
- ☞ un lecteur de carte à puce,
- ☞ un pc équipé de cartes d'acquisition et de logiciels mathématique pour le traitement des données,
- ☞ une sonde CEM si on veut étudier les émissions électromagnétiques.



Le montage



La “timing attack”

Cette attaque consiste à mesurer le temps d’exécution d’un algorithme.

⇒ Révèle des informations sur les opérations et/ou les opérandes.

Nécessite souvent :

- ☞ un grand nombre d’exécution à messages choisis,
- ☞ un traitement statistique des résultats obtenus.

La “timing attack” sur le PIN

```
for (b = 0; i >= 7; i++)
    if (pinCarte[i] != pinPresente[i])
        return false;
return true;
```

- ➡ Présenter les n valeurs de `pinPresente[0]` (256 valeurs).
- ➡ Mesurer la durée d'exécution de la commande τ pour les n valeurs.
- ➡ Calculer $\tau[n_0]$ le maximum des τ
 - $\tau[n_0] = \max(\tau[n]), n = 0, \dots, 255$
 - n_0 est la solution pour `pinPresente[0]`
- ➡ Itérer sur tous les `pinPresente[i]`

Nombre d'essais : $8 * 256 = 2048$ (contre 256^8 en force brute)

La consommation de courant

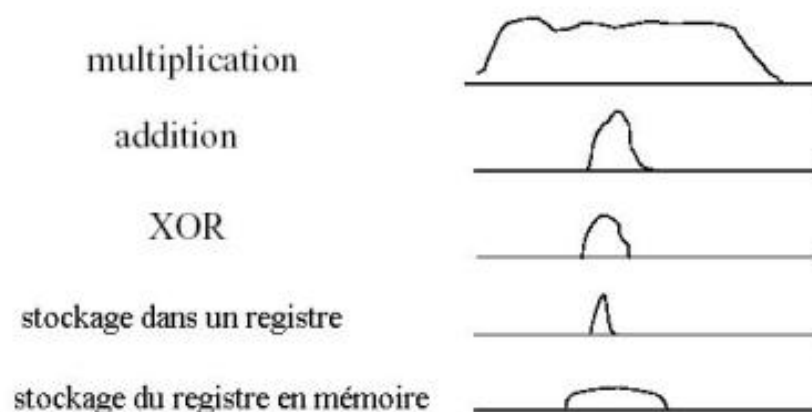
Elle est surtout utilisée dans le domaine de la cryptographie.

Il existe différentes attaques :

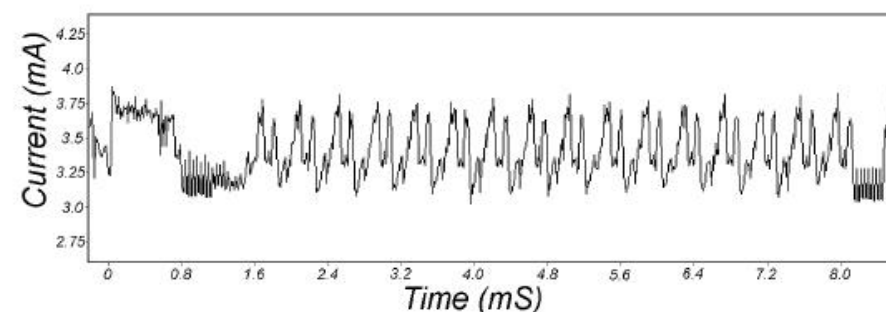
- ➡ la SPA (Simple Power Analysis)
- ➡ la DPA (Differential Power Analysis)
- ➡ la HODPA (High Order Differential Power Analysis)

La SPA (Simple Power Analysis)

Principe : Des instructions différentes ont une trace différente.



Consommation en courant d'un DES. On peut voir la permutation initiale, suivie des 16 tours.



La SPA sur la signature RSA

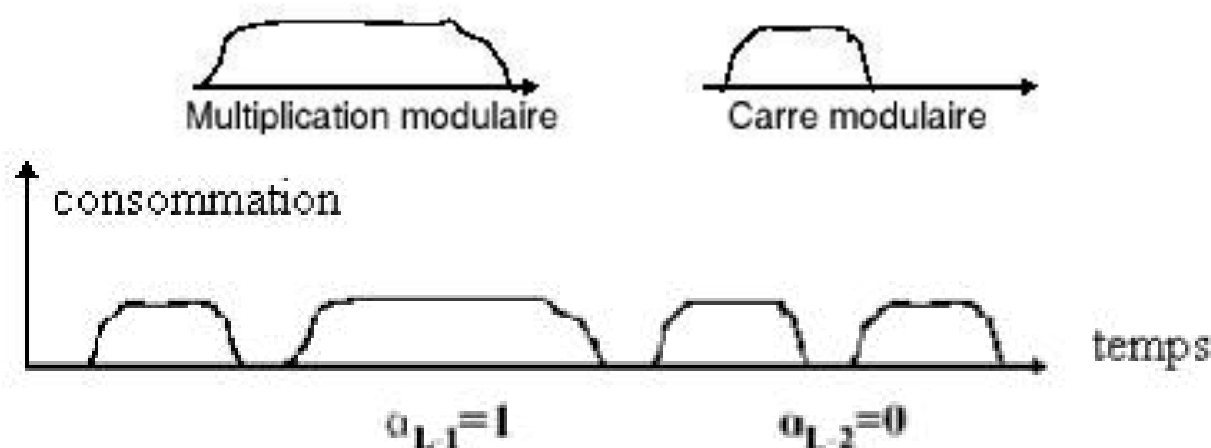
```
s = 1;
for (i = L - 1; i >= 0; i--) {
    s = s*s mod n;
    if (a[i] == 1)
        s = s*y mod n;
}
```

Signature RSA : $y^a \bmod n$

y est le message à signer,

n est public,

a , l'exposant peut être considéré comme la clé secrète.



La DPA (Differential Power Analysis)

Principe : La consommation dépend des opérations effectuées (les instructions) mais aussi des opérandes.

Elle utilise des **fonctions statistiques** adaptées à l'algorithme visé qui font ressortir des **corrélations entre un bit** intermédiaire a (ne dépendant que d'un fragment K_r de r bits de la clé et du message d'entrée M) **et la consommation de courant**.

La HODPA (High Order DPA)

La HODPA ou DPA d'ordre n est aussi basée sur une étude statistique de la consommation de courant de la carte.

Différence : elle utilise des corrélations entre la consommation de courant et n variables intermédiaires ne dépendant que d'un fragment de la clé et du message d'entrée.

Elle est beaucoup plus difficile à réaliser, mais beaucoup plus puissante.

Les émissions EM

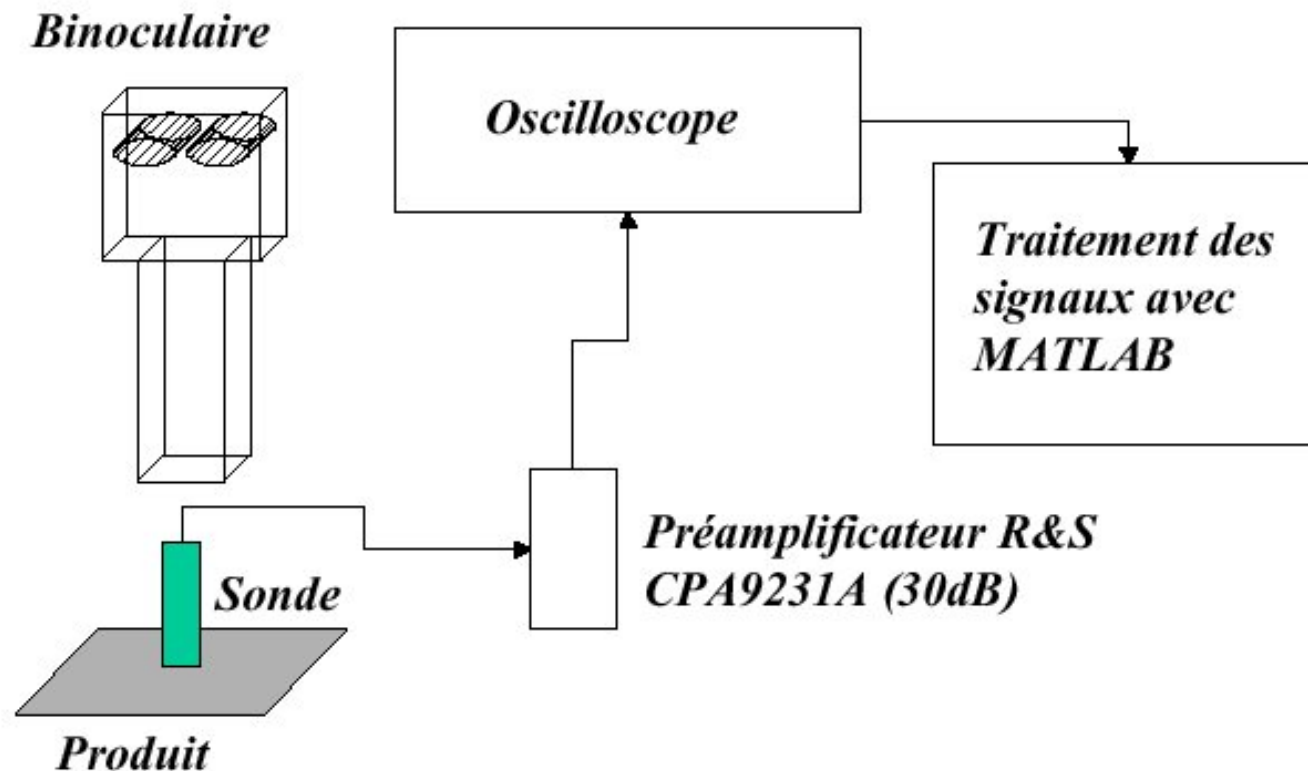
Principe : Les courants qui circulent dans la puce induisent des champs électromagnétiques qui sont susceptibles de donner le même type d'information que le courant.

Différence : L'information est plus locale. On peut déplacer la micro-sonde électromagnétique au dessus de la zone qui nous donnera le plus d'informations (exemple : co-processeur cryptographique).

Avantage : Insensible aux contre-mesures physiques tels que l'ajout de bruit en sortie ou le lissage de la consommation globale de courant.

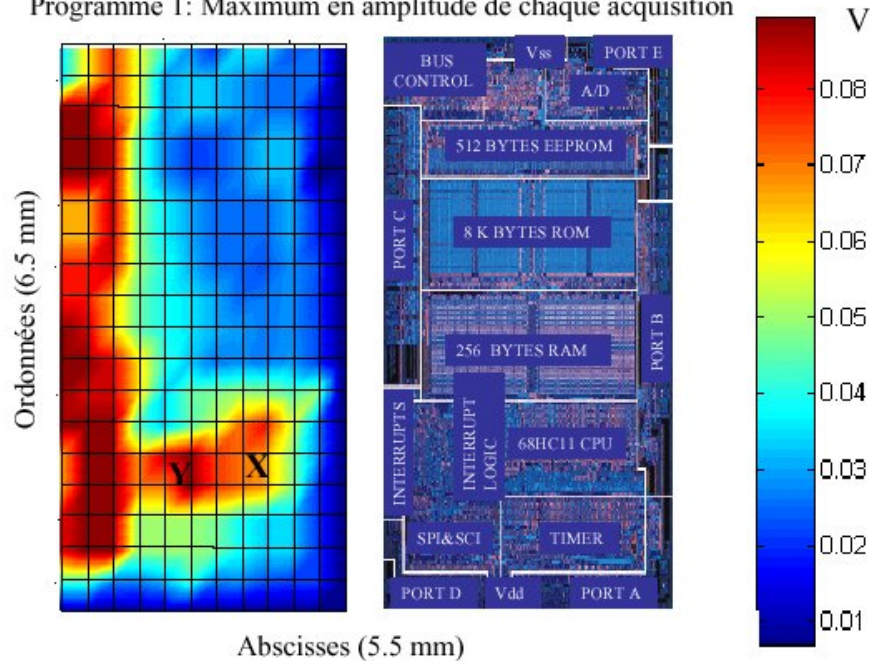
Inconvénient : La reproductibilité des mesures est difficile.

Le dispositif de cartographie EM

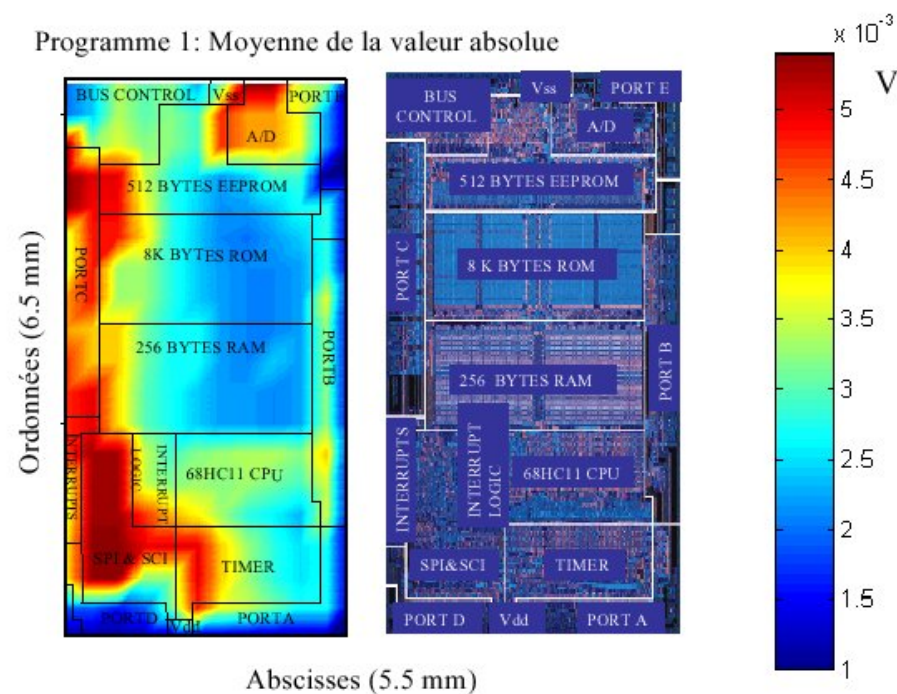


Cartographie pour l'algo 1

Programme 1: Maximum en amplitude de chaque acquisition

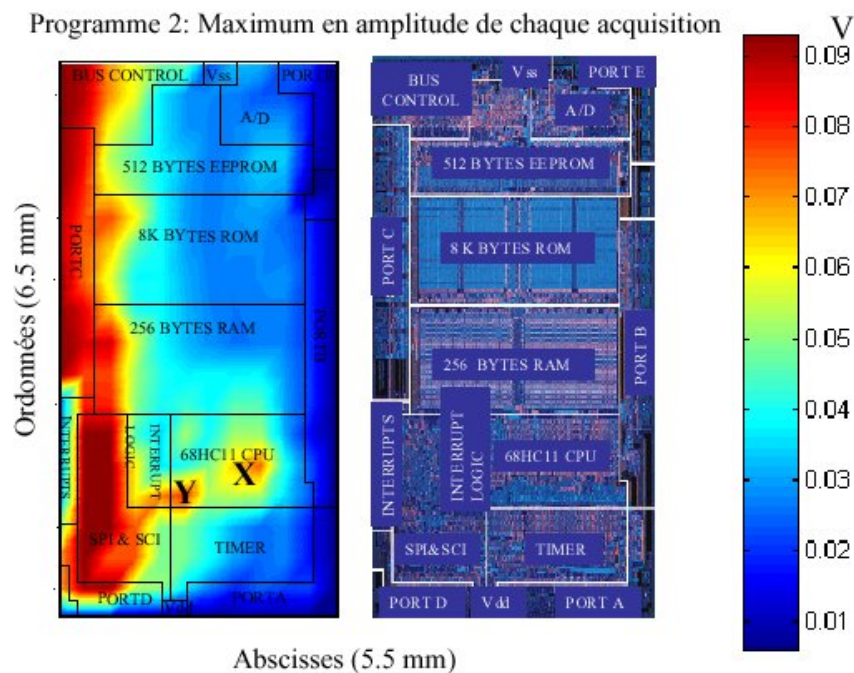


Programme 1: Moyenne de la valeur absolue

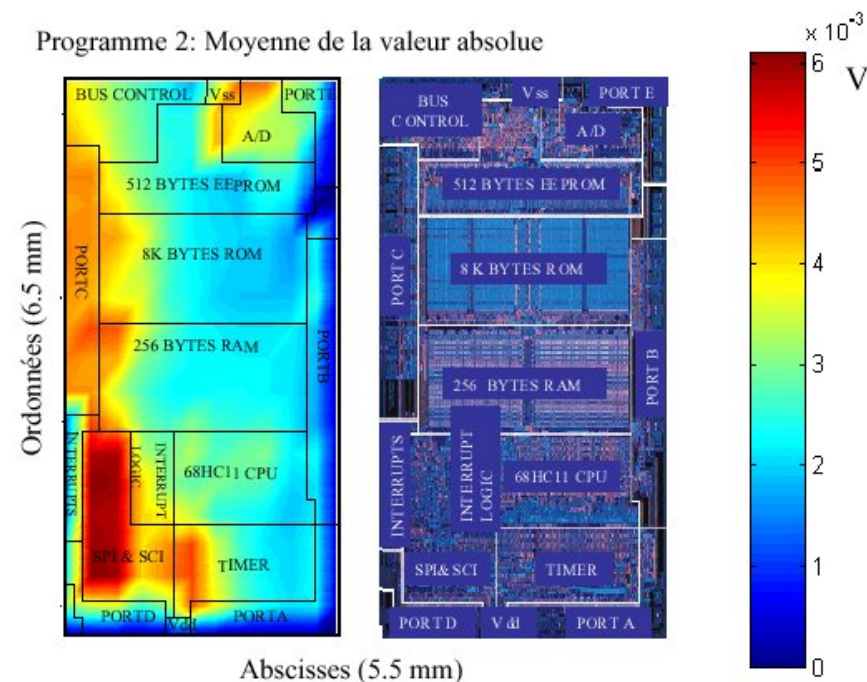


Cartographie pour l'algo 2

Programme 2: Maximum en amplitude de chaque acquisition



Programme 2: Moyenne de la valeur absolue



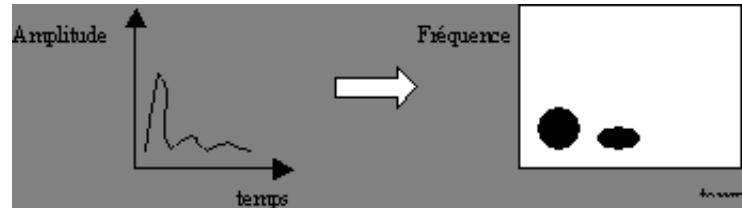
Attaques EM

- ➡ la SEMA (Simple EM Analysis)
- ➡ la DEMA (Differential EM Analysis)
 - ⇒ nécessite moins d'acquisitions

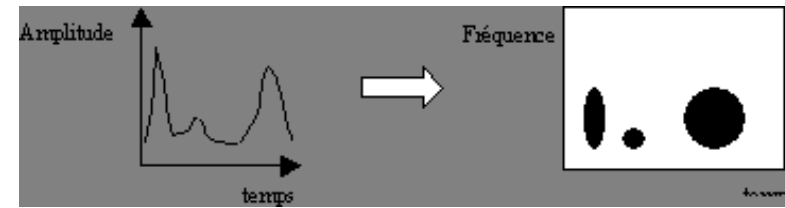
La rétro-conception logicielle

méthode du “dictionnaire” :

Une instruction :



Une autre instruction :



La séquence des deux :



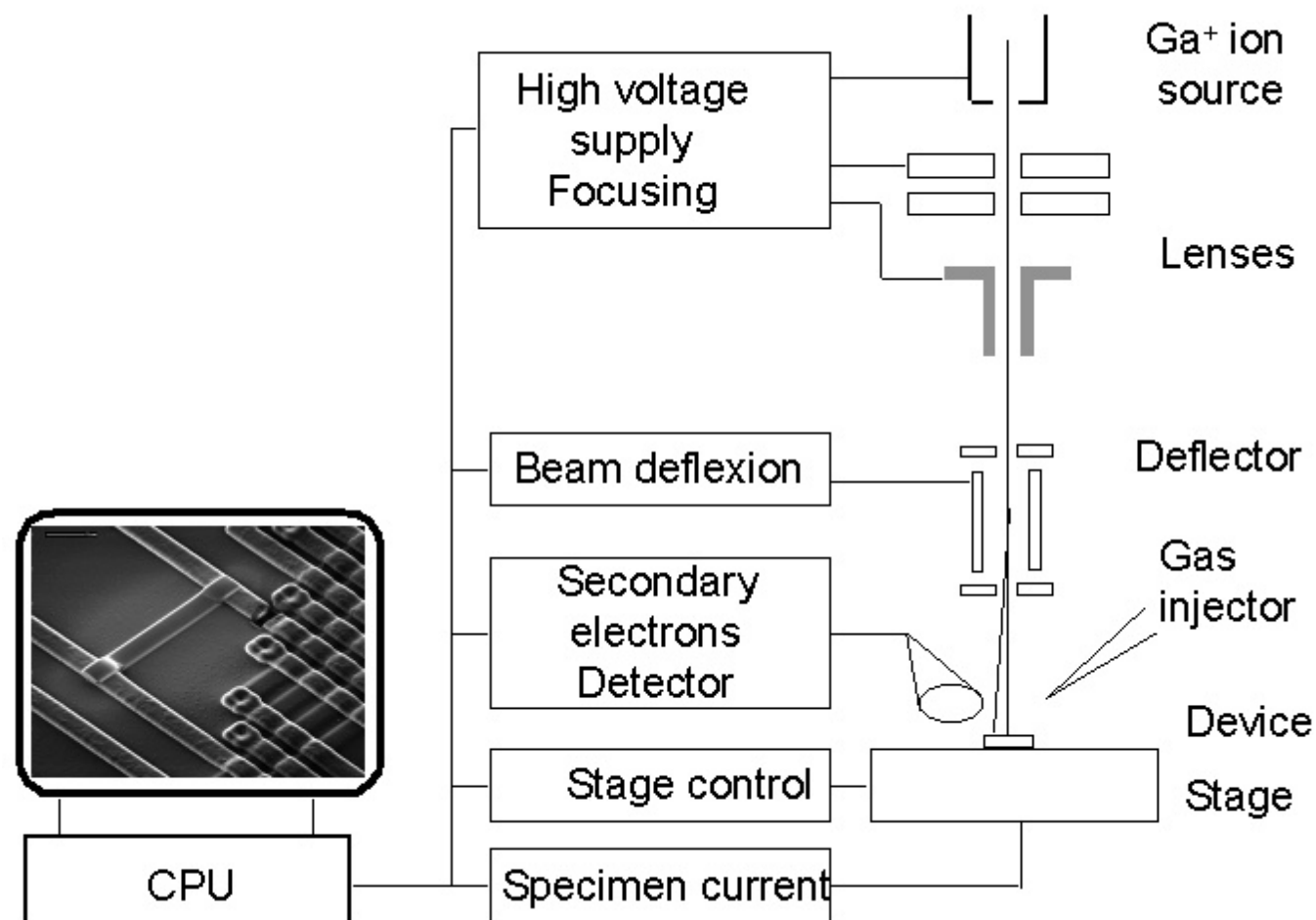
Utilisation possible de la consommation en courant ou des émissions EM.

Statut : recherches en cours.

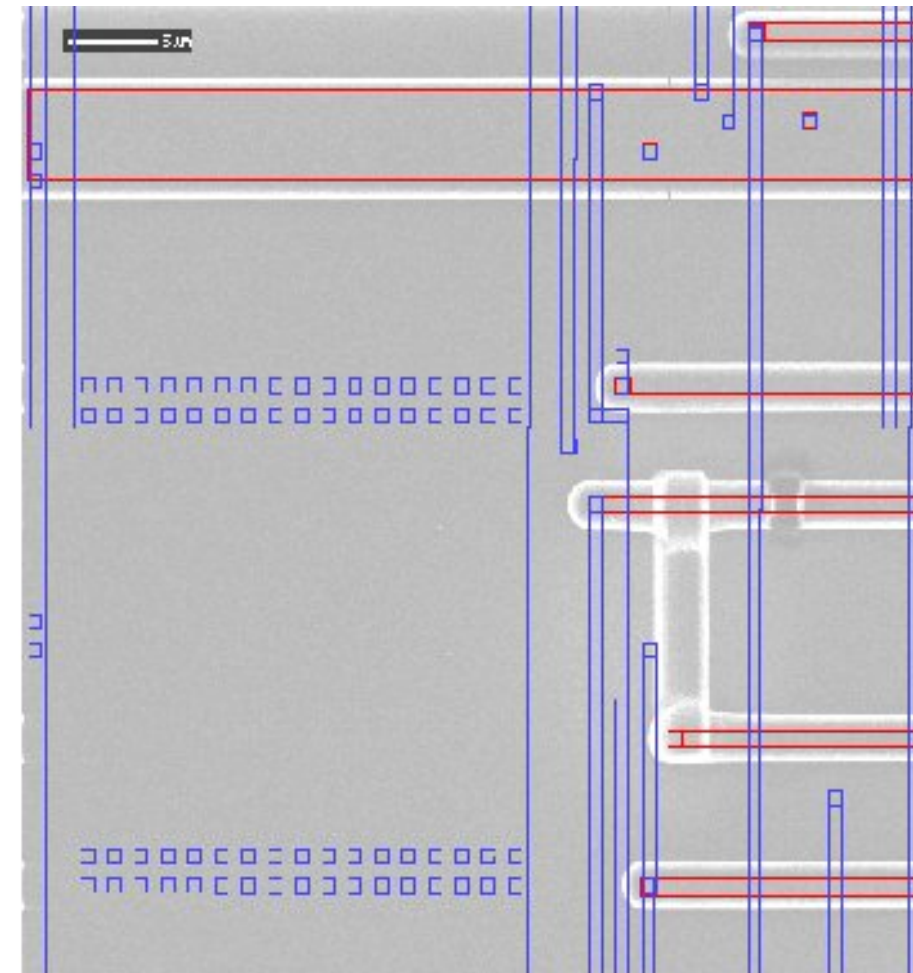
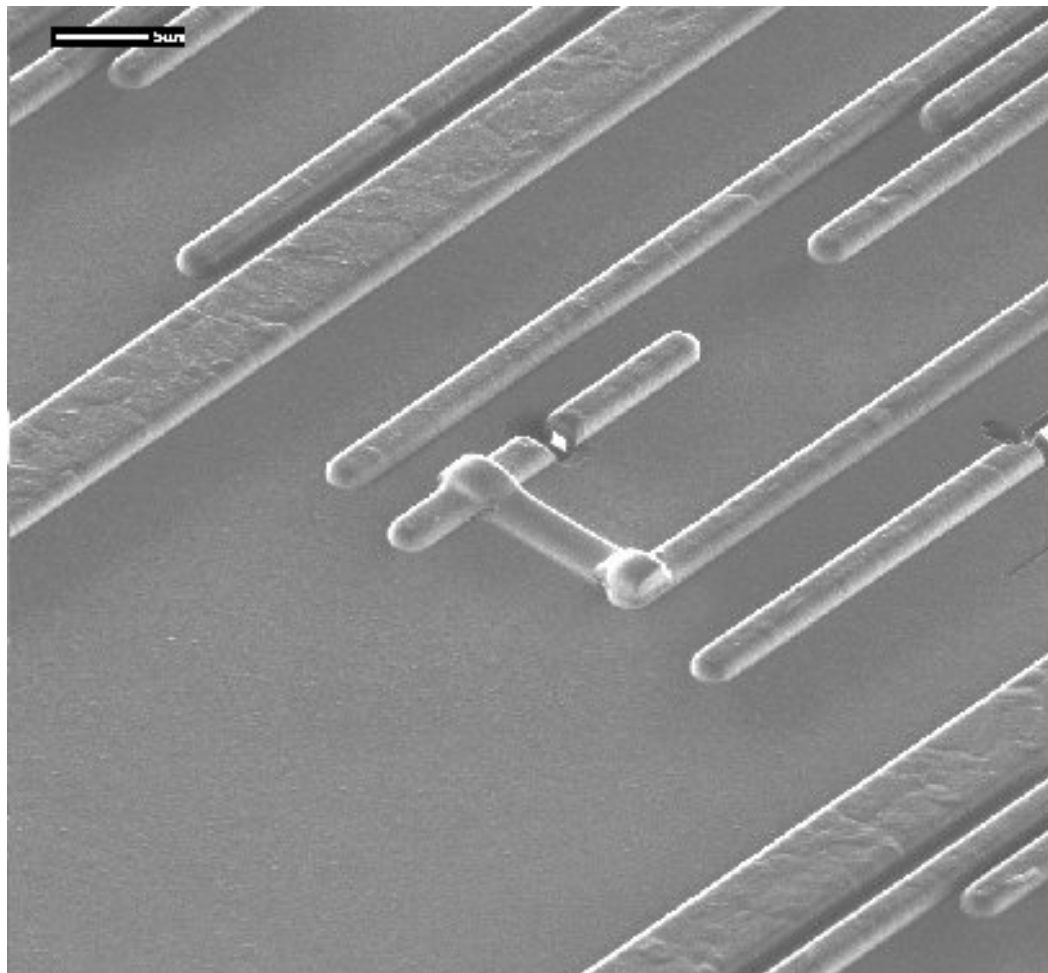
Invasives

- ☞ micro-probing
- ☞ modification de circuit (FIB)
- ☞ rétro-conception du circuit

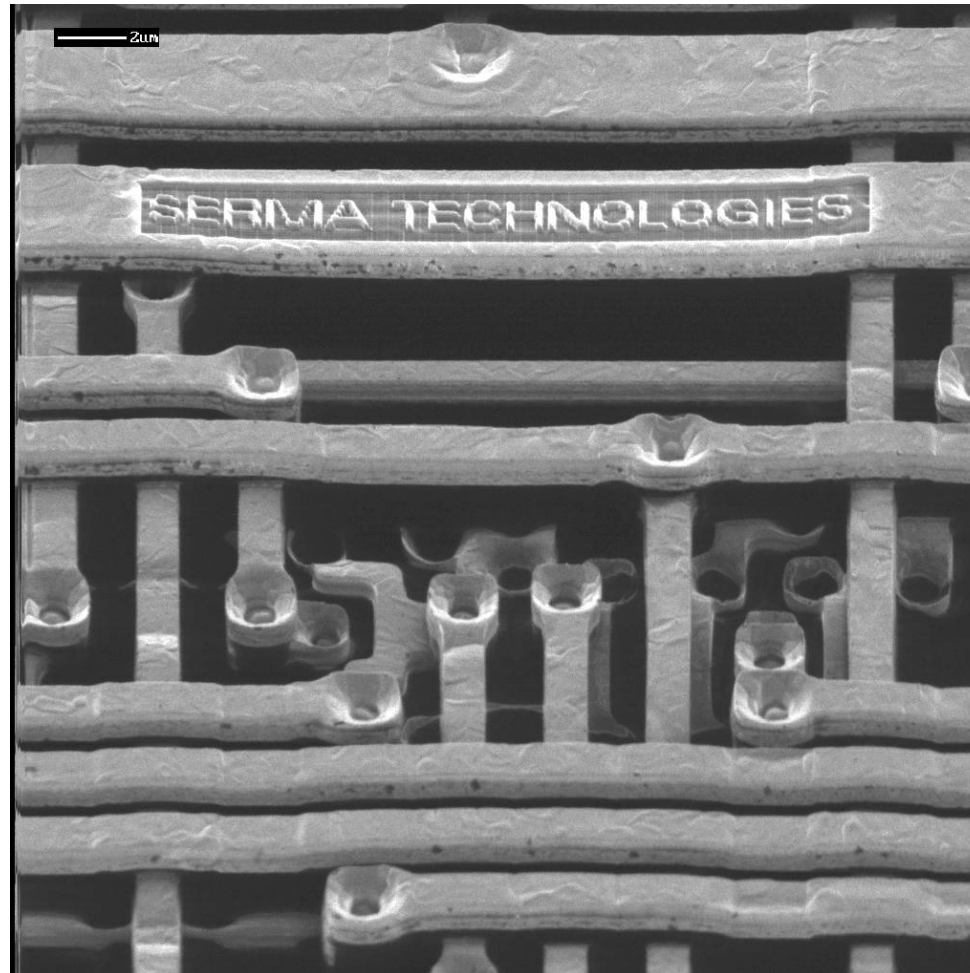
Le FIB



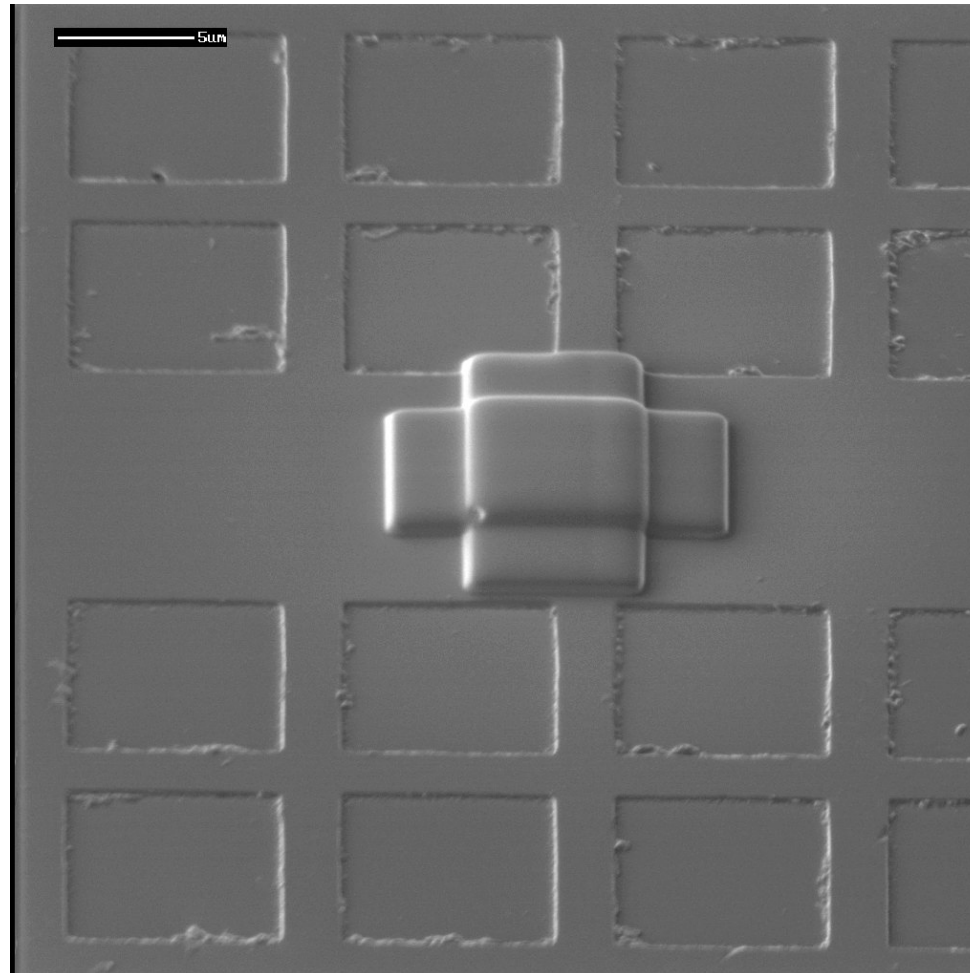
Modifications de circuits (1/2)



Modifications de circuits (2/2)



Micro-probing



Coût

Très cher !

Java Card

—

Présentation

Qu'est ce que Java Card ?

Technologie permettant de faire fonctionner des applications écrites en langage Java pour :

- ☞ carte à puce
- ☞ autres périphériques à mémoire limitée

La technologie Java Card définit une plate-forme sécurisée pour cartes à puce, portable et multi-application qui incorpore beaucoup des avantages du langage Java.

Historique

- En Novembre 1996, un groupe d'ingénieurs de Schlumberger cherche à simplifier la programmation des cartes à puce tout en préservant la sécurité.
⇒ la spécification Java Card 1.0
- En Février 1997, Bull et Gemplus se joignent à Schlumberger pour cofonder le “ *Java Card Forum* ”.
- En Novembre 1997, Sun présente les spécifications Java Card 2.0.

Historique

➤ En Mars 1999 sort la version 2.1 des spécifications Java Card. Elles consistent en trois spécifications :

- The Java Card 2.1 API Specification.
- The Java Card 2.1 Runtime Environment Specification.
- The Java Card 2.1 Virtual Machine Specification.

Contribution la plus significative :

- Définition explicite de la machine virtuelle de la Java Card.
- Le format de chargement des applets.

Historique

- En Mai 2000, sort une petite correction $\implies$ version 2.1.1
- En Octobre 2000, plus de 40 entreprises ont acquis la licence d'exploitation de la technologie Java Card.
- En juin 2002, spécifications Java Card 2.2. Introduction de RMI, des canaux logiques, ...
- En octobre 2003, spécifications Java Card 2.2.1.

Les avantages de la technologie Java Card

La facilité de développement des applications grâce :

- ➡ à la programmation orientée objet offerte par Java
- ➡ à l'utilisation des environnements de développement existants pour Java.
- ➡ à une plate-forme ouverte qui définit des APIs et un environnement d'exécution standard.
- ➡ à l'encapsulation de la complexité fondamentale du système des cartes à puce.

Les avantages de la technologie Java Card

L'indépendance au hardware réalisée grâce au langage Java

⇒ “ Write Once, Run Anywhere ”

La capacité de stockage et de gestion de multiples applications.

⇒ possibilité de mises à jour des applications de la Java Card sans avoir besoin de changer de cartes.

La *compatibilité* avec les standards existants sur les cartes à puce.

Son architecture

Problème : contraintes mémoires

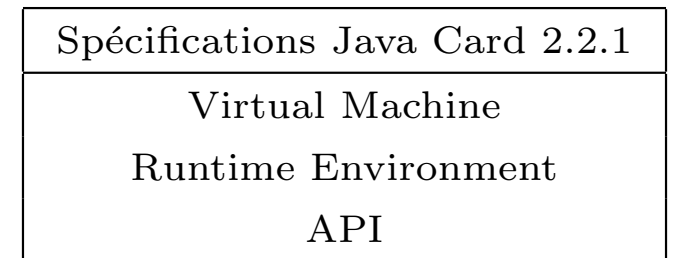
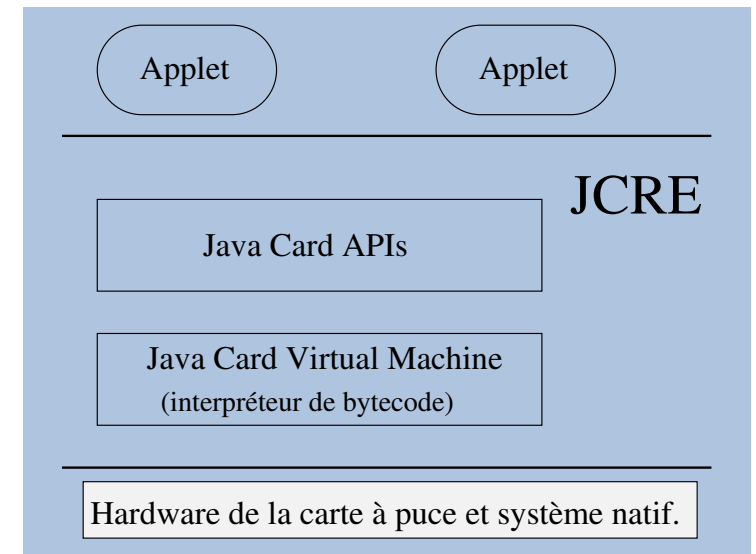
Solutions :

- un sous-ensemble des caractéristiques du langage Java
- découper la machine virtuelle Java en deux parties

Problème : pas de vérificateur embarqué

Solution :

- fournir des mécanismes sécuritaires avec l'environnement d'exécution



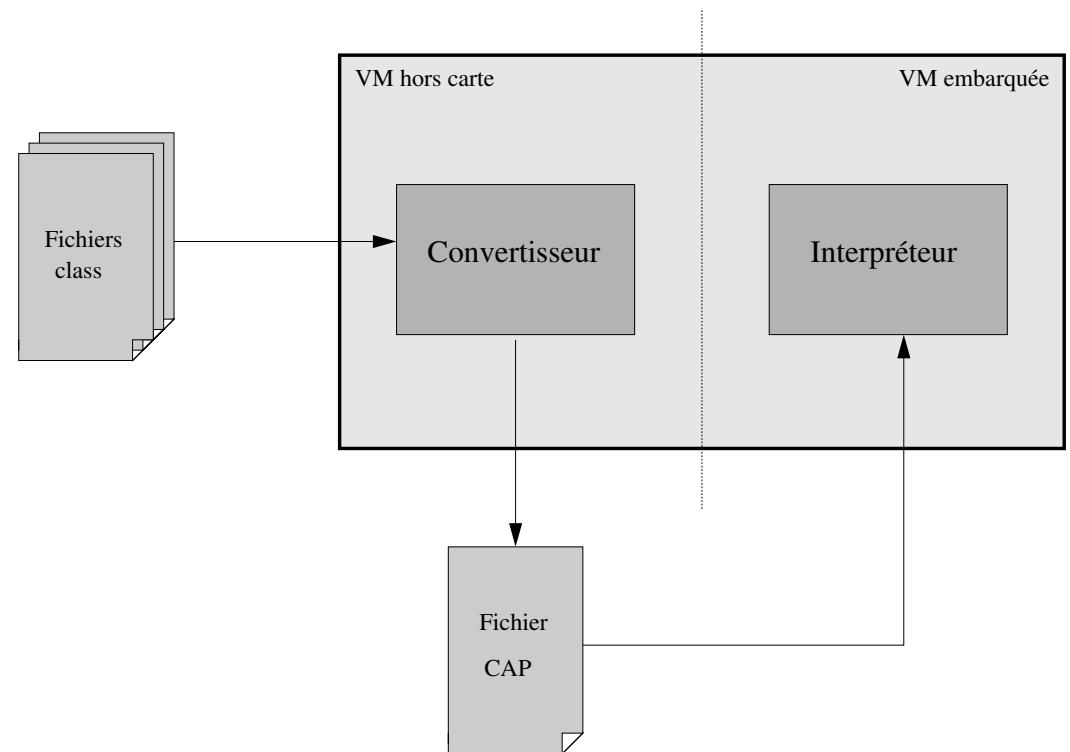
Le langage

Caractéristiques Java supportées	Caractéristiques Java non supportées
▷ Type simple de donnée de petite taille : boolean, byte, short ▷ Tableaux à une dimension ▷ Paquetages Java, classes, interfaces et exceptions ▷ Caractéristiques orientées objets : héritage, méthodes virtuelles, surcharge et création dynamique d'objets, contrôle d'accès ▷ Le mot clé int et le support des entiers sur 32 bits sont optionnels	▷ Type simple de donnée de grosse taille : long, double, float ▷ Tableaux à plusieurs dimensions ▷ Caractères et chaînes ▷ Chargement dynamique des classes ▷ Security Manager ▷ Ramasse-miettes et finalisation ▷ Threads ▷ Sérialisation d'objet ▷ Clonage d'objet

La machine virtuelle Java Card : JCVM

Les deux parties implémentent toutes les fonctions d'une machine virtuelle.

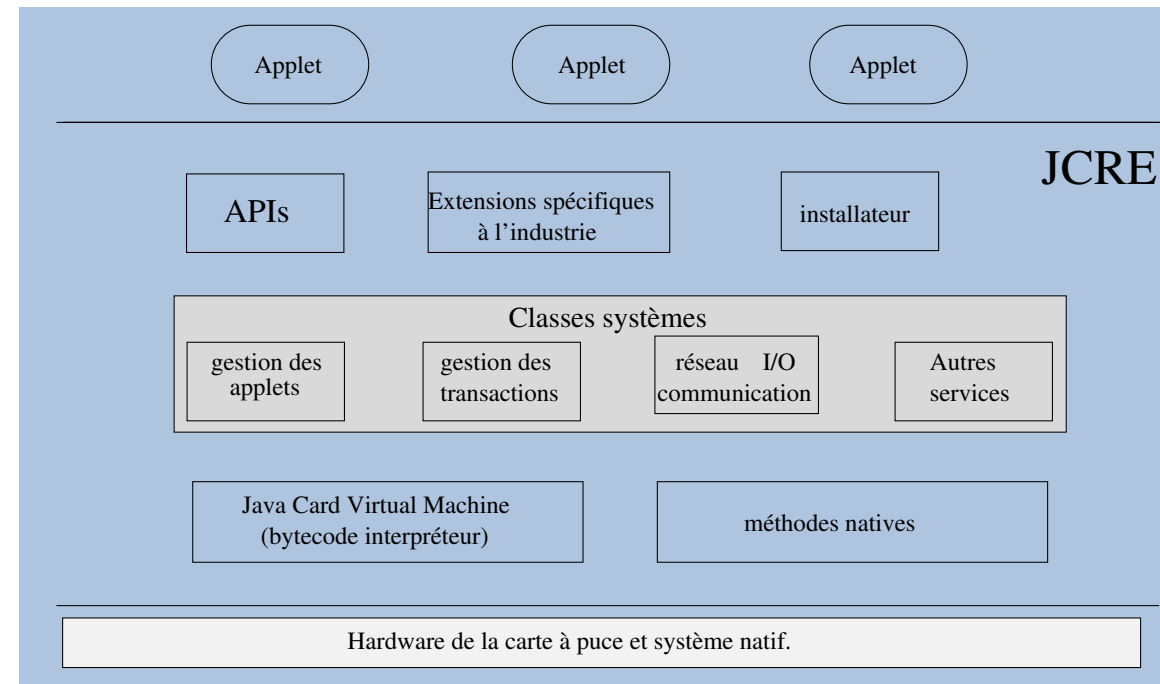
A cause du découpage de la JCVM, *la plate-forme est distribuée dans le temps et dans l'espace.*



L'environnement d'exécution Java Card

Responsable :

- ➡ de la gestion des ressources de la carte
- ➡ de la communication réseau
- ➡ de l'exécution des applets
- ➡ du système de la carte
- ➡ de la sécurité des applets



L'environnement d'exécution Java Card

Les caractéristiques du JCRE

- ➡ *Objets persistants* existant au travers des sessions avec le lecteur.
- ➡ *Objets temporaires* dont les données ne persistent pas au travers des sessions avec le lecteur.
- ➡ Chaque opération d'écriture de la JCVM est *atomique*.
- ➡ *Une transaction* est un bloc d'opérations atomiques.
- ➡ Le “ *firewall* ” isole les applets à l'intérieur de leur espace (contexte).
- ➡ Pour mettre en commun des données, il existe des *mécanismes sécurisés de partage*.

Les APIs Java Card

Ensemble de paquetages optimisés pour la programmation des cartes à puce en accord avec le modèle ISO 7816.

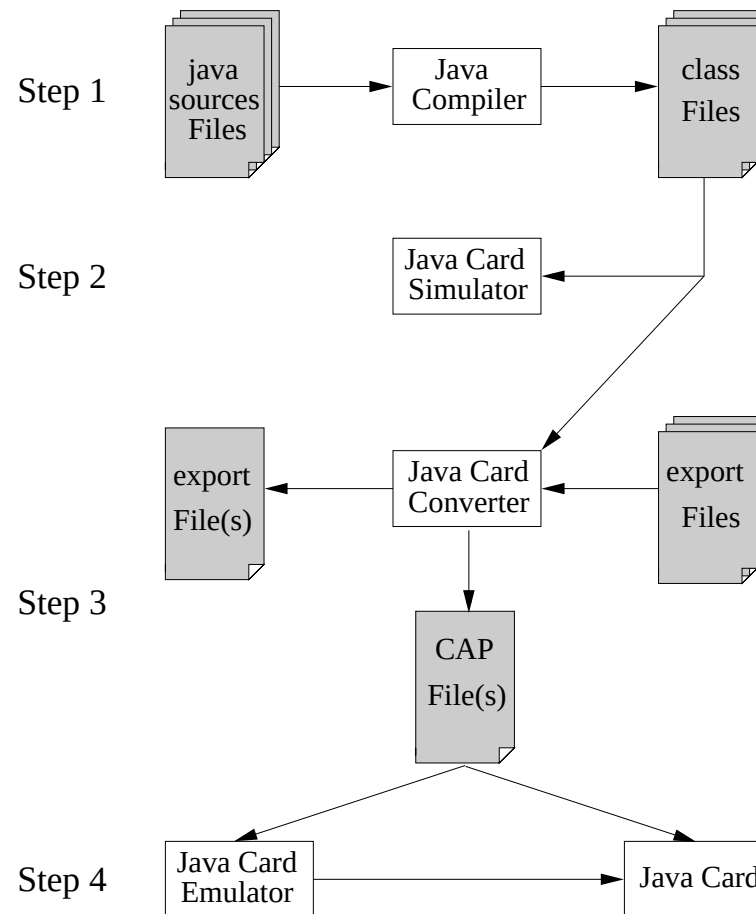
`java.lang` : un sous ensemble strict de son équivalent sur la plate-forme Java

`javacard.framework` : classes et interfaces pour le noyau fonctionnel des applets Java Card

`javacard.security` : modèle pour les fonctions cryptographiques supportées sur la plate-forme Java Card

`javacardx.crypto` : un paquetage d'extension

Cycle de développement



Java Card

—

Sa sécurité

La sécurité

- ➡ plusieurs niveaux de contrôle d'accès aux méthodes et aux variables (`public`, `protected`, `private`);
- ➡ un langage fortement typé;
- ➡ l'impossibilité de construire des pointeurs;
- ➡ un “ firewall ”.

Convertisseur Java Card

Réalise les tâches que la machine virtuelle Java sur une station de travail doit réaliser au chargement des classes :

- ➡ *Vérifie* que le chargement des *images des classes Java* sont bien formées.
- ➡ *Contrôle des violations* du langage Java Card.
- ➡ *Réalise des initialisations* de variables `static`.

Interpréteur Java Card

Fournit le *support d'exécution* du modèle du langage Java qui *autorise une indépendance au hardware* du code de l'applet.

Réalise les tâches suivantes :

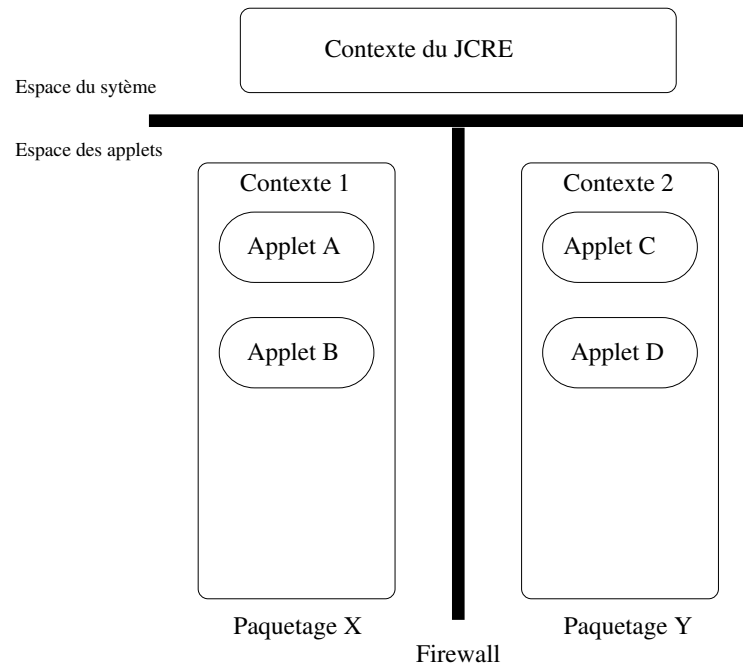
- ➡ *Exécute les instructions du bytecode* et ultimement les exécutions des applets.
- ➡ *Contrôle les allocations de mémoire et les créations d'objets.*
- ➡ Joue un rôle crucial pour s'assurer de la *sécurité* lors *de l'exécution*.

JCRE

👉 “ *Firewall* ” des applets et les mécanismes de partage

Isole les applets à l’intérieur de leur espace (contexte).

Si les applets ont besoin d’accéder à des données ou à des services du JCRE, il existe des *mécanismes sécurisés de partage*.



Java Card

—

Les attaques

Les attaques

☞ Les attaques externes

⇒ attaques identiques à celles des cartes classiques.

☞ Les attaques internes :

- détection de services ;
- scan d'applets / usurpation d'identité (spoofing) ;
- identification physique de service ou de bytecodes.

Identification physique (1/2)

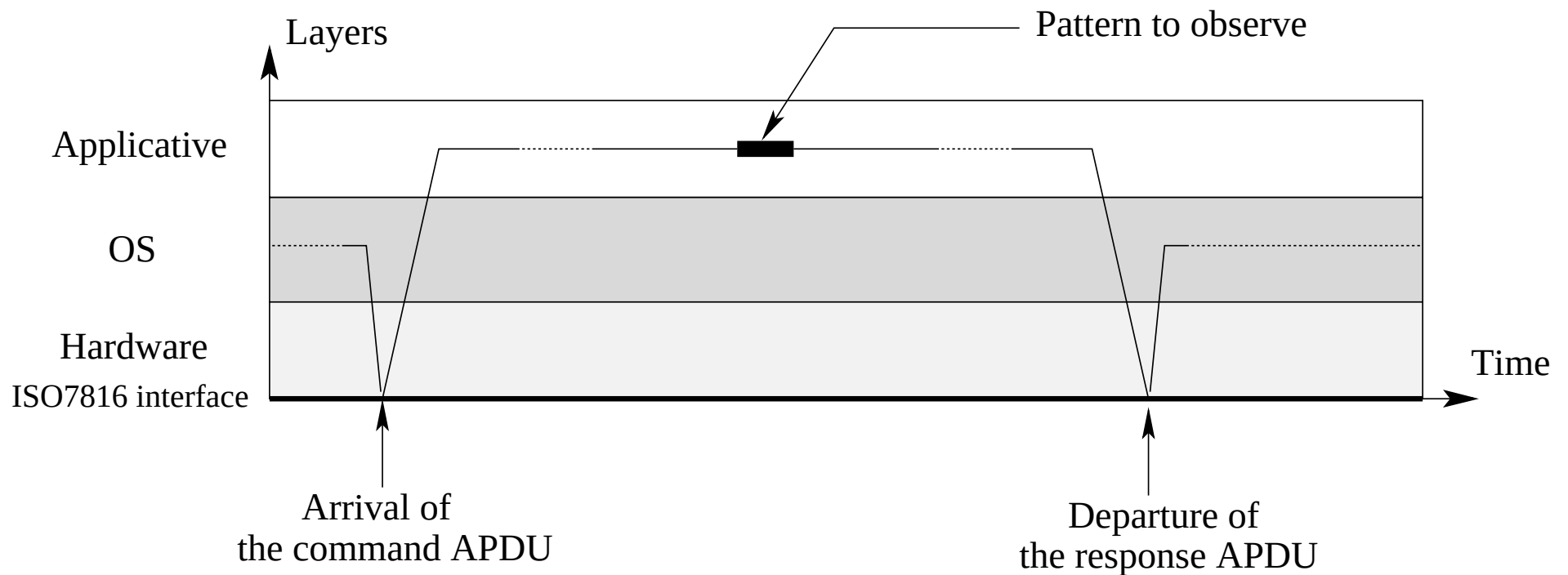


FIG. 1 – Trace d'une exécution normale.

Identification physique (2/2)

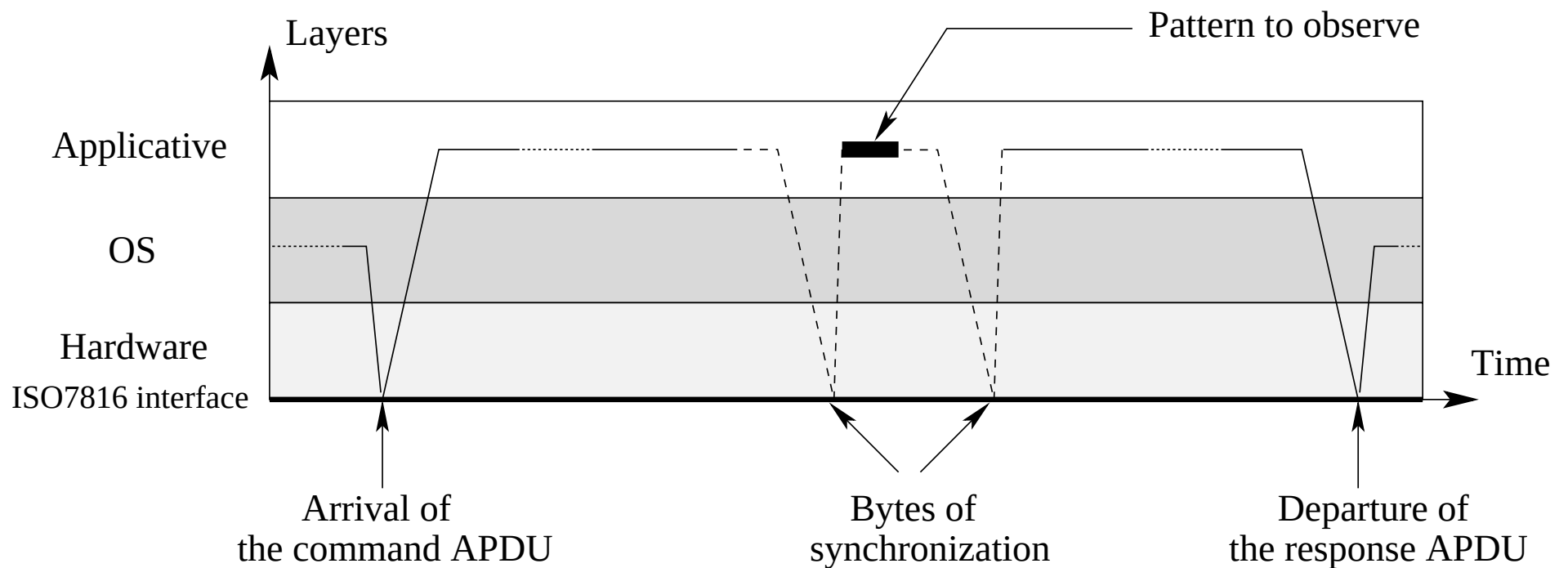


FIG. 2 – Trace d'une exécution glitchée.

Si absence de typage sur la pile

```

    sconst_0
: loop
    dup
    sconst_0 // On essaye de récupérer un octet mais
    baload   // on pourrait envisager d'autres traitements.
    invoke do_something
    sconst_1
    sadd
    goto loop

```

```

void do_something(byte value@adress) {
// take an operand stack and save it somewhere
}

```

etc.

La Certification

=

une assurance de sécurité

La certification

Objectifs :

- ➡ Garantir le niveau de sécurité d'un produit TI (confidentialité, intégrité, disponibilité).
- ➡ Bénéficier des accords de reconnaissance mutuelle.

Avantages pour :

- les utilisateurs : comparer sur une base objectif ;
- les industriels : prouver leur compétence et d'étendre leurs marchés ;
- les autorités d'homologation : s'assurer que les objectifs de sécurité sont satisfaits.

Le schéma français

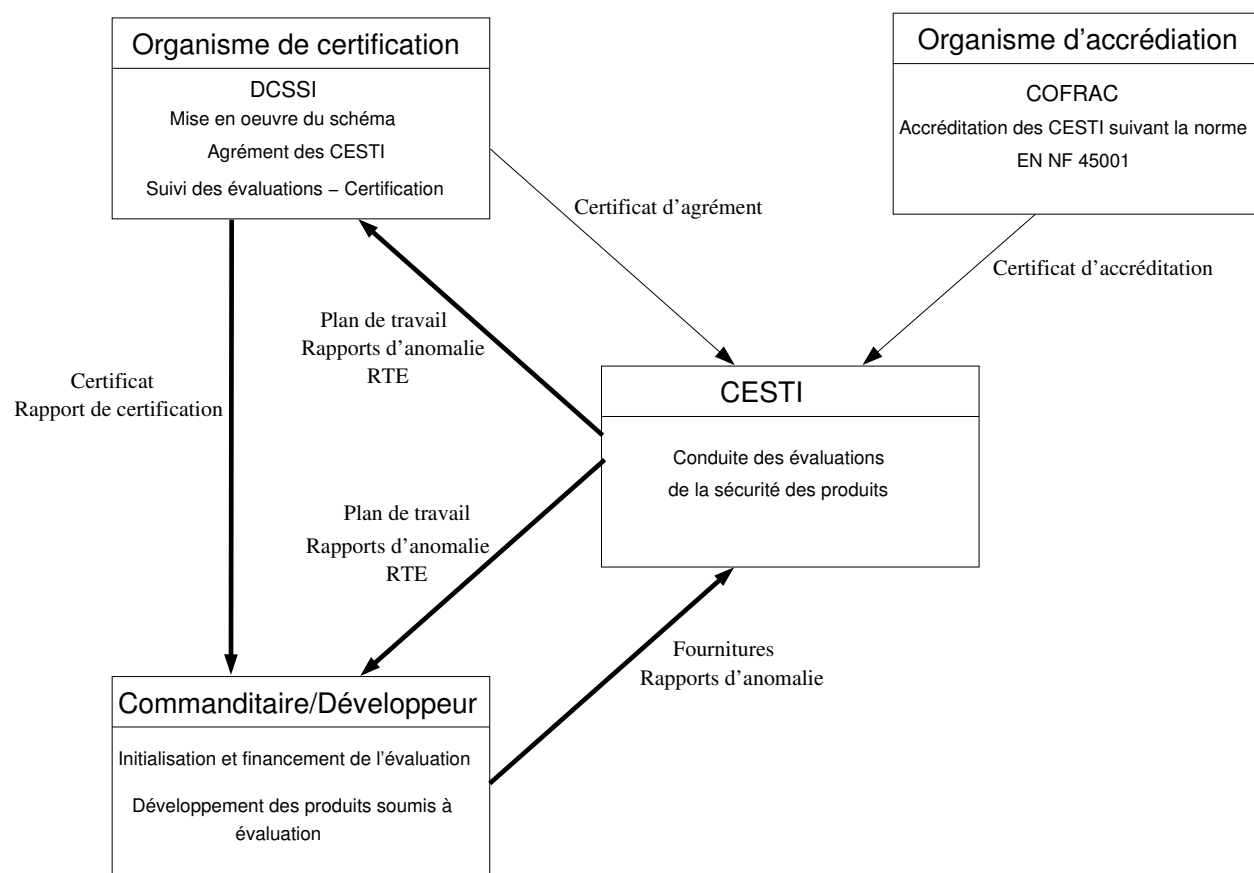


FIG. 3 – Rôles des différents intervenants et échanges d'information

La méthode Critères Communs

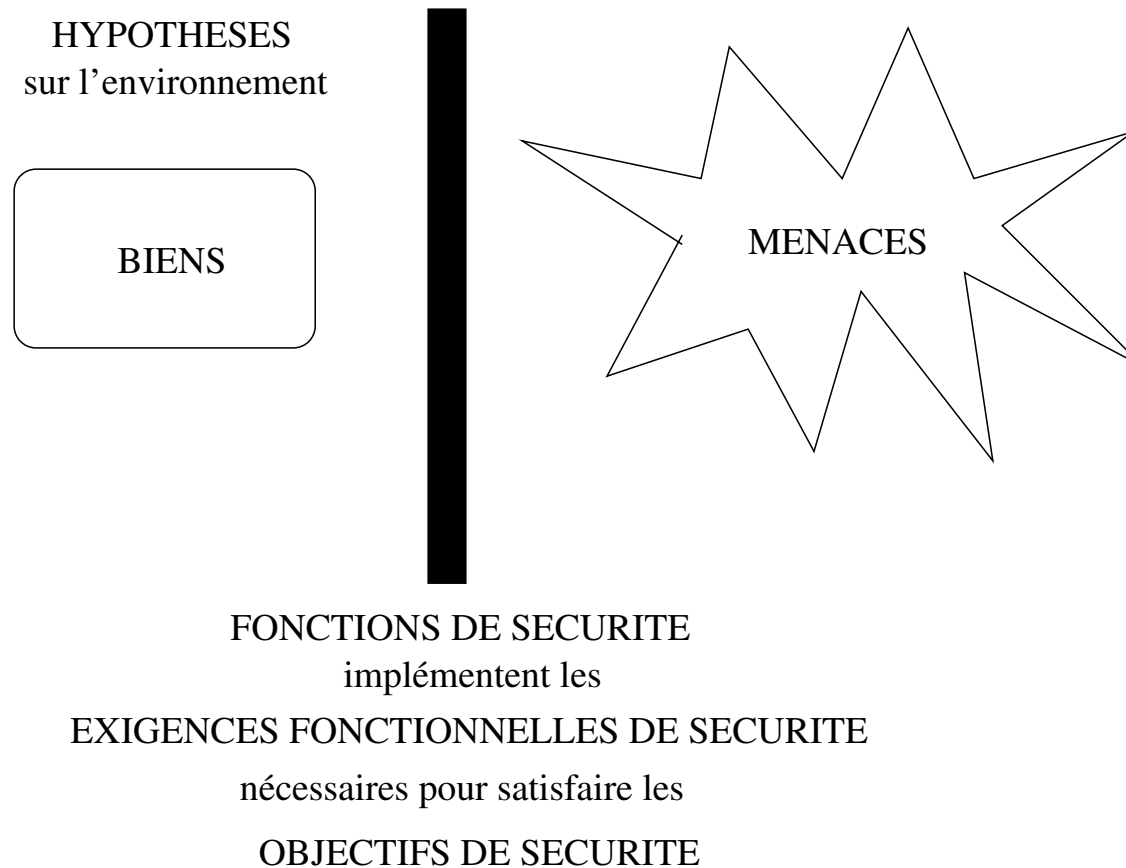


FIG. 4 – La sécurité dans les Critères Communs

L'échelle d'assurance

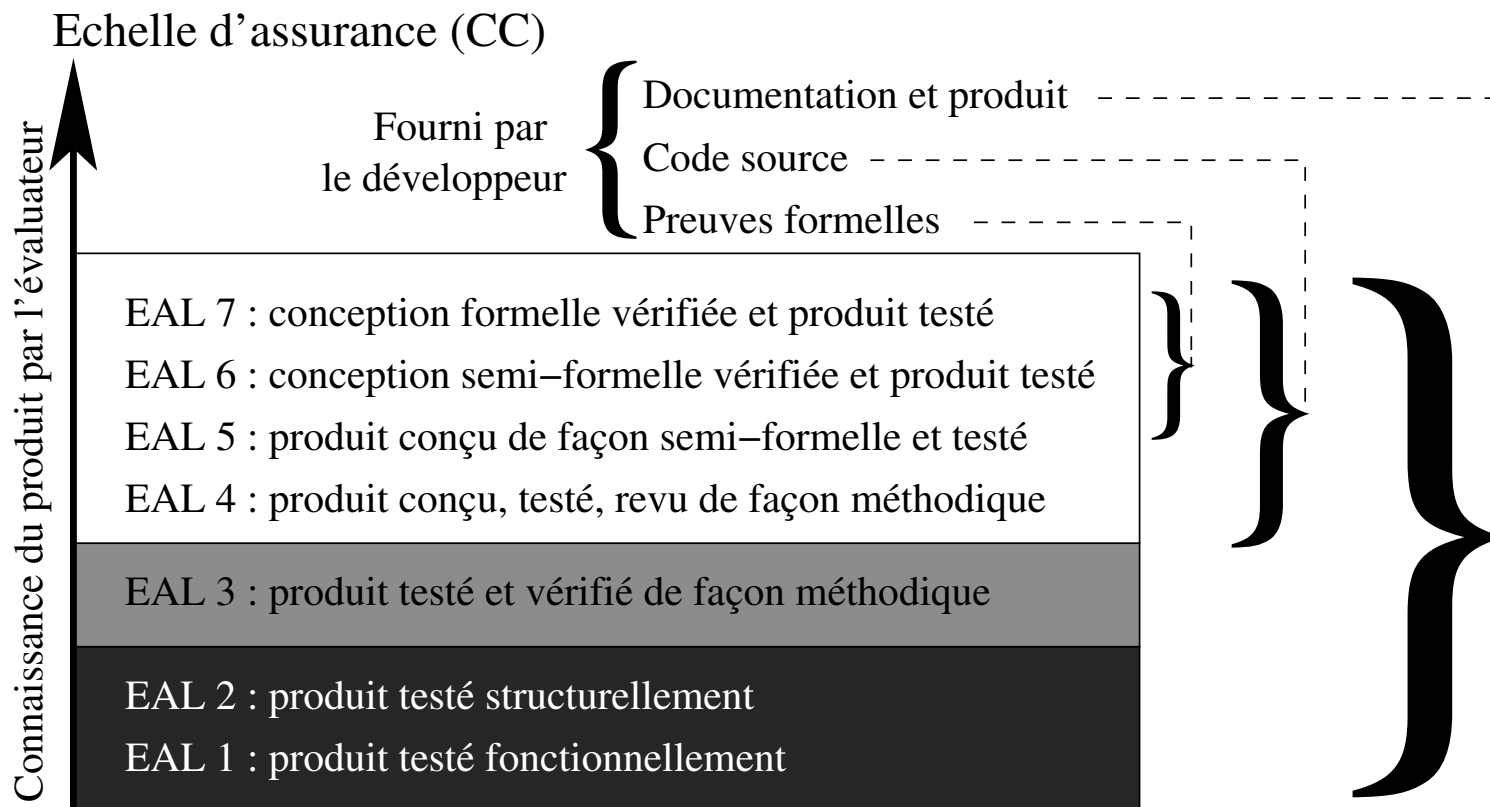
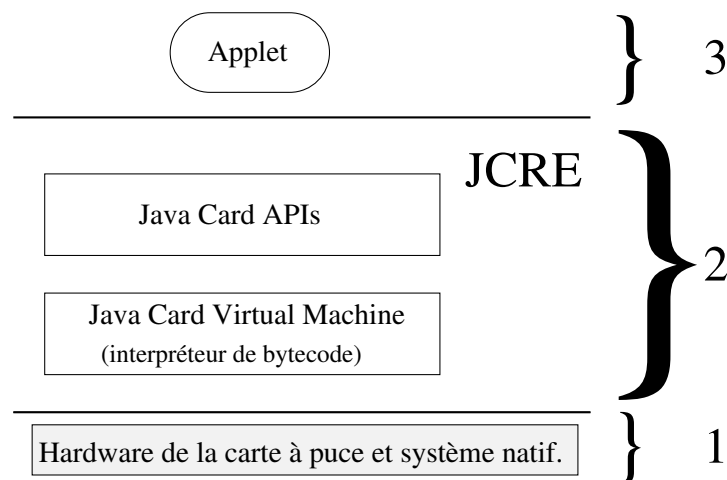


FIG. 5 – L'échelle d'assurance des Critères Communs

Périmètre d'évaluation ...

Défini dans la cible de sécurité dans la description de la cible d'évaluation.



Cible à évaluer	Périmètre		
	1	2	3
Puce	X		
JCRE	X	X	
VM & APIs	Certifié	X	
Applet	X	X	X
		Certifié	X
			X

Nouveaux problèmes apportés par les applets Java Card :

- ➡ pas de définition précise du périmètre d'évaluation ;
- ➡ pas de méthodologie d'évaluation d'une applet ;
- ➡ la multi-application.

... et les problèmes de la multi-application

Avantages :

- possibilité de réduire les coûts de développement et d'évaluation.

Problèmes :

- la peur des industriels et des utilisateurs (verrou psychologique).

Solution :

- ☞ utilisation du standard Open Platform pour maintenir un niveau de sécurité supplémentaire afin de rassurer les industriels et le schéma ;
- ☞ utilisation de Java Card comme plate-forme mono-applicative afin de ne pas inquiéter les consommateurs.

Le projet “Sécurité Java Card”

Le contexte

Besoins de SERMA Technologies :

- ➡ posséder de bonnes connaissances des technologies Java et Java Card ;
- ➡ développer une méthodologie d'évaluation :
 - pour les applets,
 - pour les plate-formes ;
- ➡ développer des batteries de tests d'attaques ;
- ➡ rester proche de la réalité (attaques hardware/software) ;
- ➡ améliorer la qualité de ses évaluations par l'utilisation de preuves formelles.

Objectifs :

- soumettre la méthodologie à DCSSI et renforcer le schéma ;
- confirmer l'agrément du CESTI pour évaluer la technologie Java Card.

Le projet Sécurité Java Card

JCatools (1/2)

Réponses aux besoins de SERMA Technologies :

Buts	JCatools			
	Emulator	View	Converter	Autres
Connaissance détaillée de la technologie Java Card	X	X		
Définition d'une méthodologie d'évaluation de plate-forme	X			
Création d'une batterie de tests d'attaques	X	X	X	X
Expérimentation de tests d'attaques software & hardware	X			

Développements réalisés en *Java*.

JCatools (2/2)

JCat Emulator :

Destiné à tester et attaquer des implémentations d'applets.

JCat View :

Outil de visualisation des formats de fichiers CAP de différents constructeurs.

JCat Converter :

Outil de conversion d'un format de fichier CAP d'un constructeur vers celui d'un autre.

Autres JCatools : par exemple, l'outil d'aide à la modification de bytecode
 $\implies$ faciliter la création de batterie de tests et d'attaques.

JCat Emulator (1/2)

Implémentation des spécifications JC 2.1.1 :

- ☞ VM et APIs ;
- ☞ JCRE (firewall et transaction).

Objectifs :

- tester et déboguer des applets ;
- détecter des problèmes dans leur comportement.

Fonctionnalités :

- exécution pas à pas ;
- visualisation des mémoires, des objets, buffer de transaction, pile de frame (en cours), etc.

JCat Emulator (2/2)

Vers une méthodologie d'évaluation d'une plate-forme :

Liste des points obscurs dans les spécifications qui sont importants pour la sécurité.

Exemple : transaction annulée dans un `install()` après l'appel à `register()`.

Futures améliorations :

- instrumentations de la machine $\implies$ statistiques ;
- améliorer la simulation d'attaques physiques ;
- intégrer l'implémentation des APIs crypto ;
- compatibilité JC 2.2 (RMI, multi-channels) et Open Platform.

JCAT

File Options Help

Java Card 0

Loaded Package

0xa0	0x00	0x00	0x00	0x62	0x00	0x01
0x00	0x00	0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	0x00	0x00	0x01
0xa0	0x00	0x00	0x00	0x62	0x01	0x01
0x00	0x11	0x22	0x33	0x44		

Frames

Transaction

In progress : ●
index : 0

0x0	0xd	0x0	0x8	0x0	0x0	0x2
0x7	0xfd	0xfd	0x0	0x1d	0x0	0x0
0xa	0xa	0xa	0xa	0xa	0x5	0xfe
0x0	0x1a	0x0	0x6	0xff	0xff	0x2
0x7	0xfd	0x0	0x8	0x0	0x0	0x2
0x7	0xfd	0x0	0x1b	0x0	0x0	0x0
0x0	0x2	0x7	0xfd	0x0	0x1b	0x0
0x2	0x0	0x0	0x2	0x7	0xfd	0x0
0x1b	0x0	0x4	0x0	0x0	0x2	0x7
0xfd	0x0	0x1b	0x0	0x6	0x0	0x0
0x2	0x7	0xfd	0x0	0x1b	0x0	0x8
0x0	0x0	0x2	0x7	0xfd	0x0	0x1a
0x0	0x0	0x0	0x0	0x2	0x7	0xfd
0x0	0x1c	0x0	0x0	0x0	0x1	0x6
0xfd	0x0	0x1c	0x0	0x1	0x0	0x1
0x6	0xfd	0x0	0x1c	0x0	0x2	0x0
0x1	0x6	0xfd	0x0	0x1c	0x0	0x3
0x0	0x1	0x6	0xfd	0x0	0x1c	0x0
0x4	0x0	0x1	0x6	0xfd	0x0	0x1a
0x0	0x2	0x0	0x0	0x2	0x7	0xfd
0x0	0x1d	0x0	0x0	0x0	0x1	0x6
0xfd	0x0	0x1d	0x0	0x1	0x0	0x1
0x6	0xfd	0x0	0x1d	0x0	0x2	0x0
0x1	0x6	0xfd	0x0	0x1d	0x0	0x3

eeeprom

- ram ref: 0x1
- fields = 0xe 0x1a 0x0 0x0 0x0
- fields = 0xf 0x1 0x0
- 0x00 0x21 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
- fields =
- fields =
- fields =
- fields =

ram

- ram
 - 0xff 0xff 0xff 0xff 0x00 0x00 0x00 0x00 0x00 0x00 0:

```

/* 1618, 0x3aa */ sload_1      , Operand Stack : [ 0xc 0x0 ]
/* 1619, 0x5ab */ sload_2      , Operand Stack : [ 0xc 0x0 0x102 ]
/* 1620, 0x5ac */ sload_3      , Operand Stack : [ 0xc 0x0 0x102 0x0 ]
/* 1621, 0x5ad */ invokestatic , 0x0, null, Operand Stack : [ ]
/* 1622, 0x00f */ invokestatic , 0x0, null, Operand Stack : [ ]
/* 1623, 0x018 */ impdep1      , [0x2], array [ ram ref: 0x1 ]
/* 1624, 0x012 */ return       , Operand Stack : [ ]
/* 1625, 0x5b0 */ sload_1      , Operand Stack : [ 0x0 ]
/* 1626, 0x5b1 */ sload_2      , Operand Stack : [ 0x0 0x102 ]
/* 1627, 0x5b2 */ sadd         , Operand Stack : [ 0x102 ]
/* 1628, 0x5b3 */ sreturn      , Operand Stack : [ 0x102 ]
/* 1629, 0x233 */ pop          , Operand Stack : [ ]
/* 1630, 0x234 */ getfield_a_this, Operand Stack : [ 0xc ]
/* 1631, 0x236 */ sload_1      , Operand Stack : [ 0xc 0x9000 ]
/* 1632, 0x237 */ invokestatic , 0x0, null, Operand Stack : [ ]
/* 1633, 0x01b */ invokestatic , 0x0, null, Operand Stack : [ ]
/* 1634, 0x0a0 */ impdep1      , [0x14]

- out going bytes:
- SW                : 0x90 0x00
- apdu received    : 0xffffffff ...
  
```


JCAT

File Options Help

431 return

Java Card 0 Java Card 1 Java Card 2 Java Card 3

Loaded Package

0xa0	0x00	0x00	0x00	0x62	0x00	0x01
0x00	0x00	0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	0x00	0x00	0x01
0xa0	0x00	0x00	0x00	0x62	0x01	0x01
0x00	0x11	0x22	0x33	0x44		

Frames

Depth: 0

Owner: 0x0, null

PCReg: 0x682

Stack:

Local: 0x0 0x0 0x0

Transaction

In progress : ●

index: 0

0x0	0x1	0x0	0x1d	0x0	0x1	0x6
0xfd	0xfd					

eeeprom

- fields = 0x0
- fields = 0x0
- fields = 0x0 0x0
- fields = 0x0
- fields = 0x0
- fields = 0x0
- fields = 0x0
- fields = 0xc
- ram ref: 0x1
- fields = 0x0 0x0 0x0 0x0 0x0
- fields = 0xf 0x0 0x0
- 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

reference 0xc

ram

- ram
- 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x

L'appui sur le bouton provoque un arrachage (TEARING) et donc la carte rep redémarre au debut

```

418, 0x030 /* return , Operand Stack : [ ]
/* 417, 0x7e7 /* getstatic_a , Operand Stack : [ 0xb ]
/* 418, 0x7ea /* invokevirtual , 0x0, null, Operand Stack : [ ]
/* 419, 0x1bf /* getfield_a_this, Operand Stack : [ 0xc ]
/* 420, 0x1c1 /* areturn , Operand Stack : [ 0xc ]
/* 421, 0x7ed /* putstatic_a , static fields = 0x0 0x4 0x0 0x5 0x0 0x6 0x0 0x3 0x0 0xd 0x0 0
/* 422, 0x7f0 /* getstatic_a , Operand Stack : [ 0xc ]
/* 423, 0x7f3 /* invokestatic , 0x0, null, Operand Stack : [ ]
/* 424, 0x033 /* invokestatic , 0x0, null, Operand Stack : [ ]
/* 425, 0x0bc /* impdep1 , [0x18]
/* 426, 0x036 /* return , Operand Stack : [ ]
/* 427, 0x7f6 /* sconst_1 , Operand Stack : [ 0x1 ]
/* 428, 0x7f7 /* putstatic_b , static fields = 0x0 0x4 0x0 0x5 0x0 0x6 0x0 0x3 0x0 0xd 0x0 0
/* 429, 0x7fa /* return , Operand Stack : [ ]
/* 430, 0x682 /* invokestatic , 0x0, null, Operand Stack : [ ]
/* 431, 0x701 /* invokestatic
TEARING TEARING TEARING TEARING TEARING TEARING TEARING
, 0x0, null
/* 431, 0x67a /* getstatic_b , Operand Stack : [ 0x1 ]
/* 432, 0x67d /* ifne , Operand Stack : [ ]
/* 433, 0x682 /* invokestatic

```

Le projet Sécurité Java Card

Vue des objets (1/3)

Object view

Properties **Values**

Reference 0x1

Owner

PackageAID : null

Applet Owner : 0x0

Attributes

Special attributes : NONE

Transient type : NOT_A_TRANSIENT_OBJECT

Type

Static Fields: byte [30]

Object view

Properties **Values**

[0]	0x0
[1]	0x4
[2]	0x0
[3]	0x0A
[4]	0x0
[5]	0x6
[6]	0x0
[7]	0x3
[8]	0x0
[9]	0xd
[10]	0x0
[11]	0x9
[12]	0x0

Le projet Sécurité Java Card

Vue des objets (2/3)

Object view

Properties **Values**

Reference 0x3

Owner

PackageAID : null

Applet Owner : 0x0

Attributes

Special attributes : TEMPORARY_ENTRY_POINT

Transient type : NOT_A_TRANSIENT_OBJECT

Type

flags	: 0
interface count	: 0
super classRef	: external class 0x2 of package 0x0
declared instance size	: 1
first reference token	: 255
reference count	: 0
public method table base	: 1
public method table count	: 2
package method table base	: 0
package method table count	: 0
public virtual method table	: { 0x259 0x25e }
package virtual method table	: { }
implemented interfaces	: { }

Object view

Properties **Values**

Reference 0x1f

Owner

PackageAID : null

Applet Owner : 0x0

Attributes

Special attributes : PERMANENT_ENTRY_POINT

Transient type : NOT_A_TRANSIENT_OBJECT

Type

flags	: 0
interface count	: 0
super classRef	: external class 0x0 of package 0x0
declared instance size	: 1
first reference token	: 0
reference count	: 1
public method table base	: 0
public method table count	: 5
package method table base	: 0
package method table count	: 0
public virtual method table	: { 0x129 0x193 0x159 0x116 0x176 }
package virtual method table	: { }
implemented interfaces	: { }

Le projet Sécurité Java Card

Vue des objets (3/3)

Object view

Reference 0xc

Owner

PackageAID : null

Applet Owner : 0x0

Attributes

Special attributes : GLOBAL_ARRAY

Transient type : CLEAR_ON_RESET

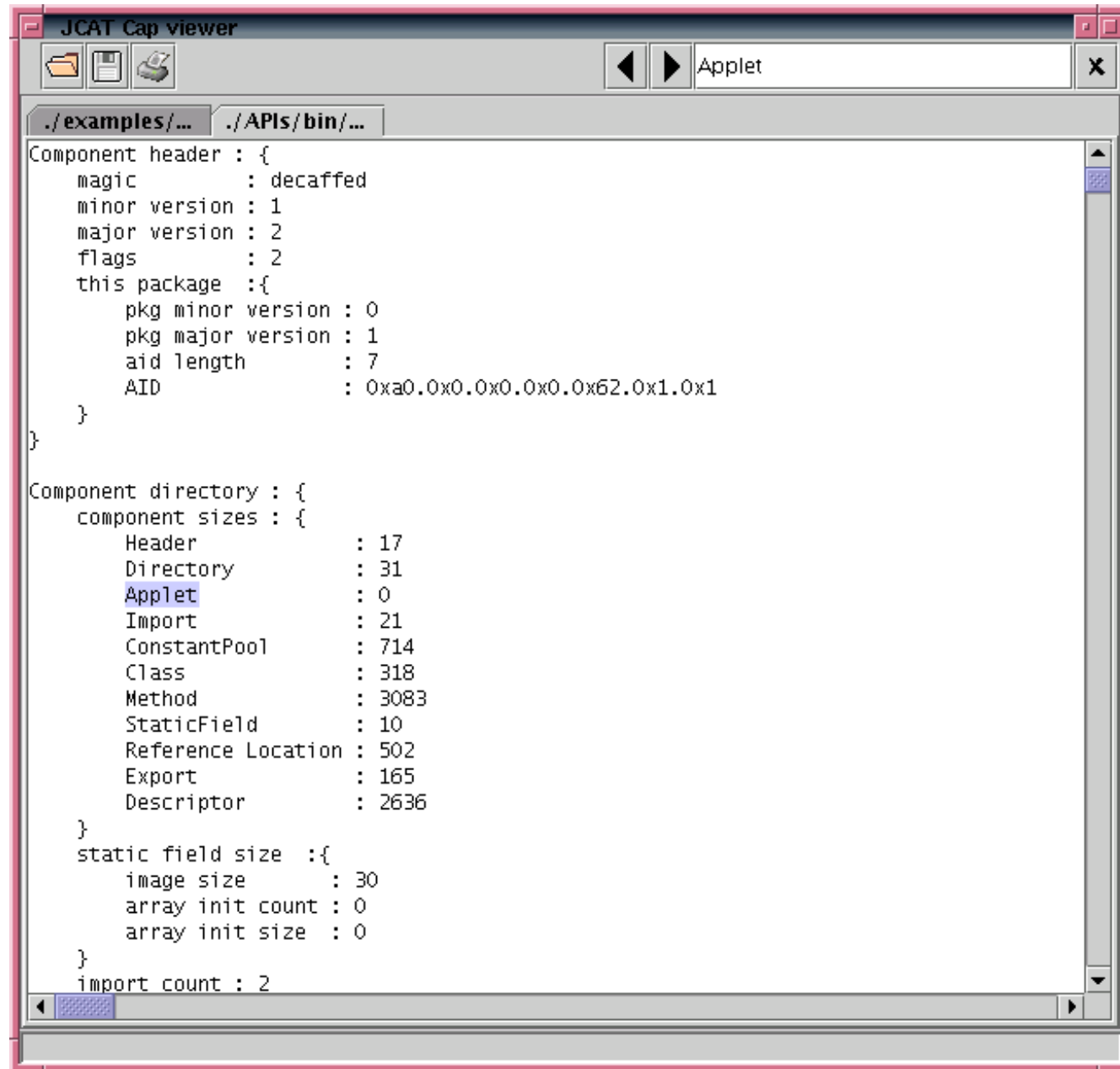
RAM

RAM reference: 0x1

Le projet Sécurité Java Card

Statistiques

Statistics	
byte code name	count
aload_0	168
return	144
invokestatic	126
sload_1	80
invokespecial	63
sload	62
sconst_0	61
sload_3	48
getstatic_a	48
impdep1	47
getfield_a_this	44
sadd	43
invokevirtual	43
aload_2	36
dup	32
sload_2	31
ifnonnull	29
sconst_1	27
sreturn	27
baload	25
arraylength	22
putfield_a	18
putfield_b	18
getstatic_b	17
aload_1	16



De la théorie à la pratique

Les travaux LaBRI/SERMA Technologies ont facilité :

Evaluation EAL4+ d'une plate-forme Java Card (première fois en France) :

- ☞ utilisation des résultats et des recommandations issus du développement de JCat Emulator ;
 - ☞ mise en place d'une première batterie de tests d'attaques logicielles et logicielles/matérielles.
- ⇒ Recommandations pour les développeurs d'applets sur cette plate-forme.

2 évaluations EAL4+ de produits comportant des applets Java Card

- ☞ batterie spécifique de tests d'attaques logicielles et logicielles/matérielles.

Le projet “Java Cards Grid”

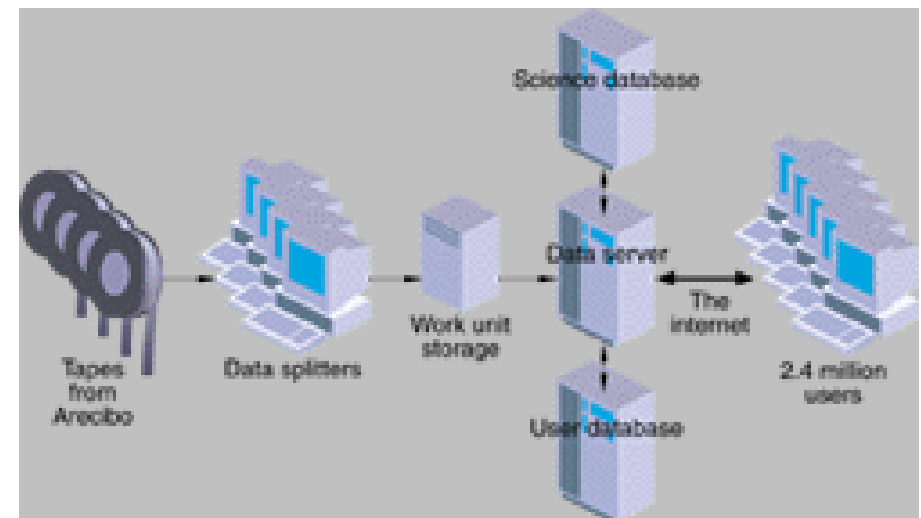
Quelques projets utilisant la grille



Décrypton

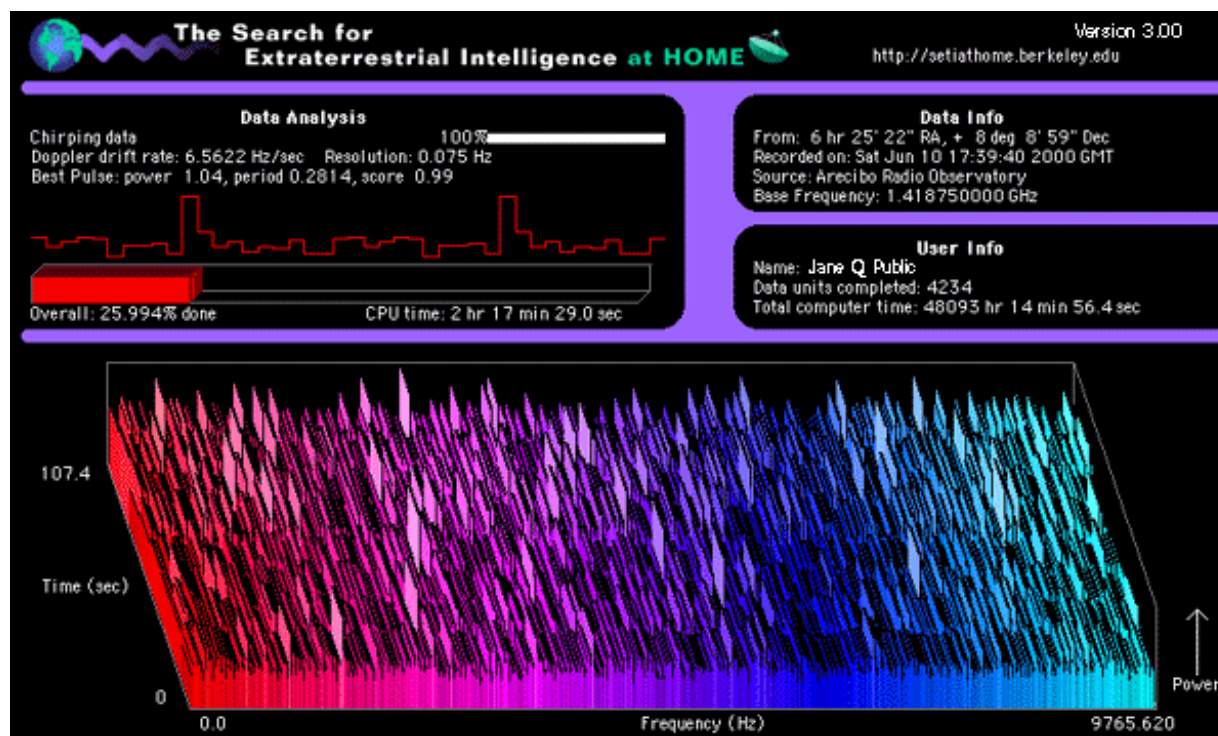
Seti@home

But : Recherche de signaux extra-terrestres.

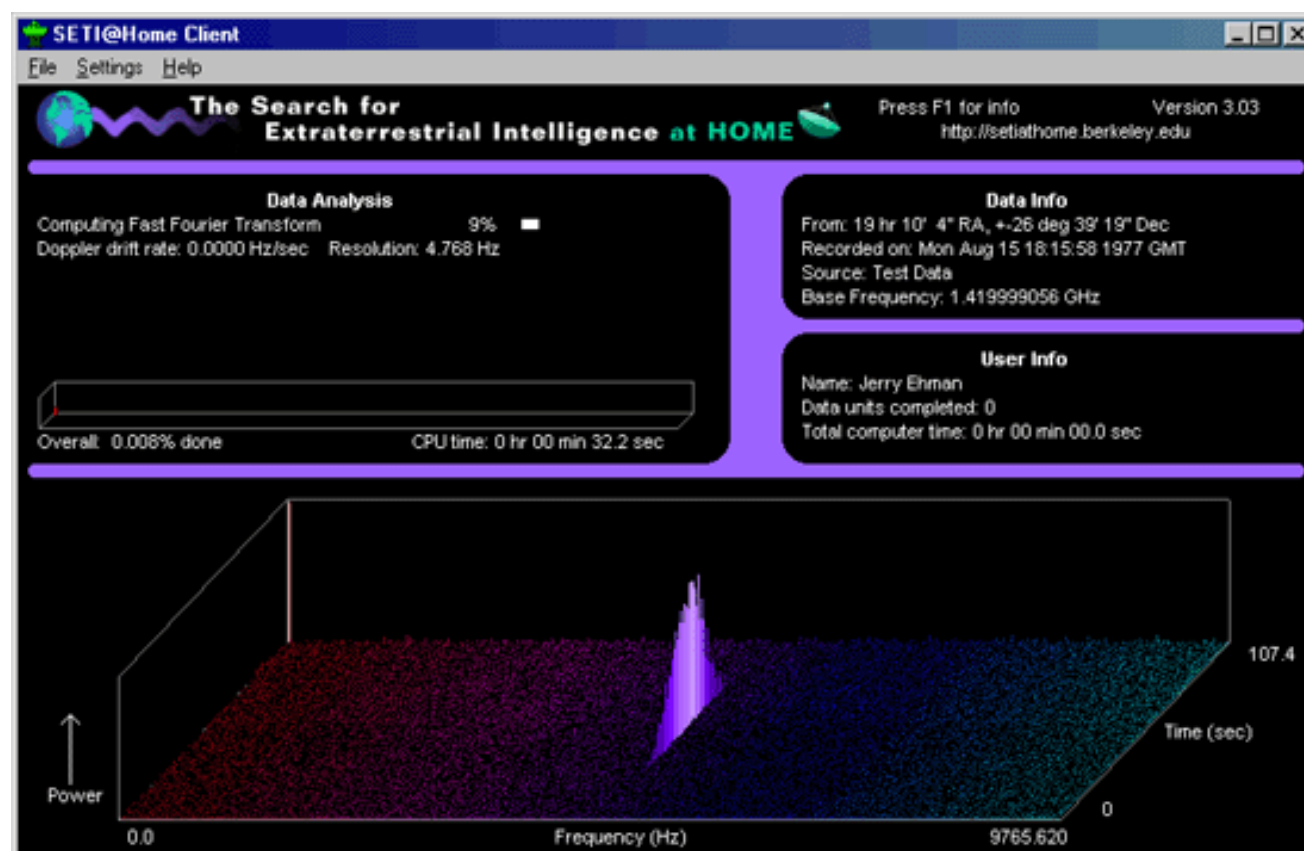


Les calculs

L'utilisateur télécharge un screensaver puis les données.



Les résultats



Nous ne
sommes PAS
SEULS dans
l'univers ?

SI! $\implies$ Il s'agit de résultats truqués! :-)

Les problèmes

- ☞ Les résultats ne sont pas sûrs.
 ⇒ ***solution*** : Faire calculer les mêmes données plusieurs fois par différents utilisateurs

- ☞ Code source non disponible.
 ⇒ éviter les faux résultats.

- ☞ Que fait le code sur la machine de l'utilisateur ?

Le projet “Java Cards Grid”

Objectifs :

- ➡ Garantir la sécurité du code et des calculs sur la grille.
- ➡ Proposer un framework logiciel exploitant du matériel sécurisé pour assurer la confiance des utilisateurs.

Avantages pour :

- les utilisateurs du *distributed computing* : une plus grande confiance dans le fournisseur des ressources de calculs ;
- les fournisseurs des ressources de calculs : accroître son marché avec des nouveaux clients comme le CEA, ...

Le projet “Java Cards Grid”

Java Card fournit la Pull Security

Pull Security : le fait d’être sûr que quand vous téléchargez un programme sur le réseau, il ne sera pas dangereux ni pour votre machine ni pour les autres code qui tournent dessus

☞ Sand box approach

Le projet “Java Cards Grid”

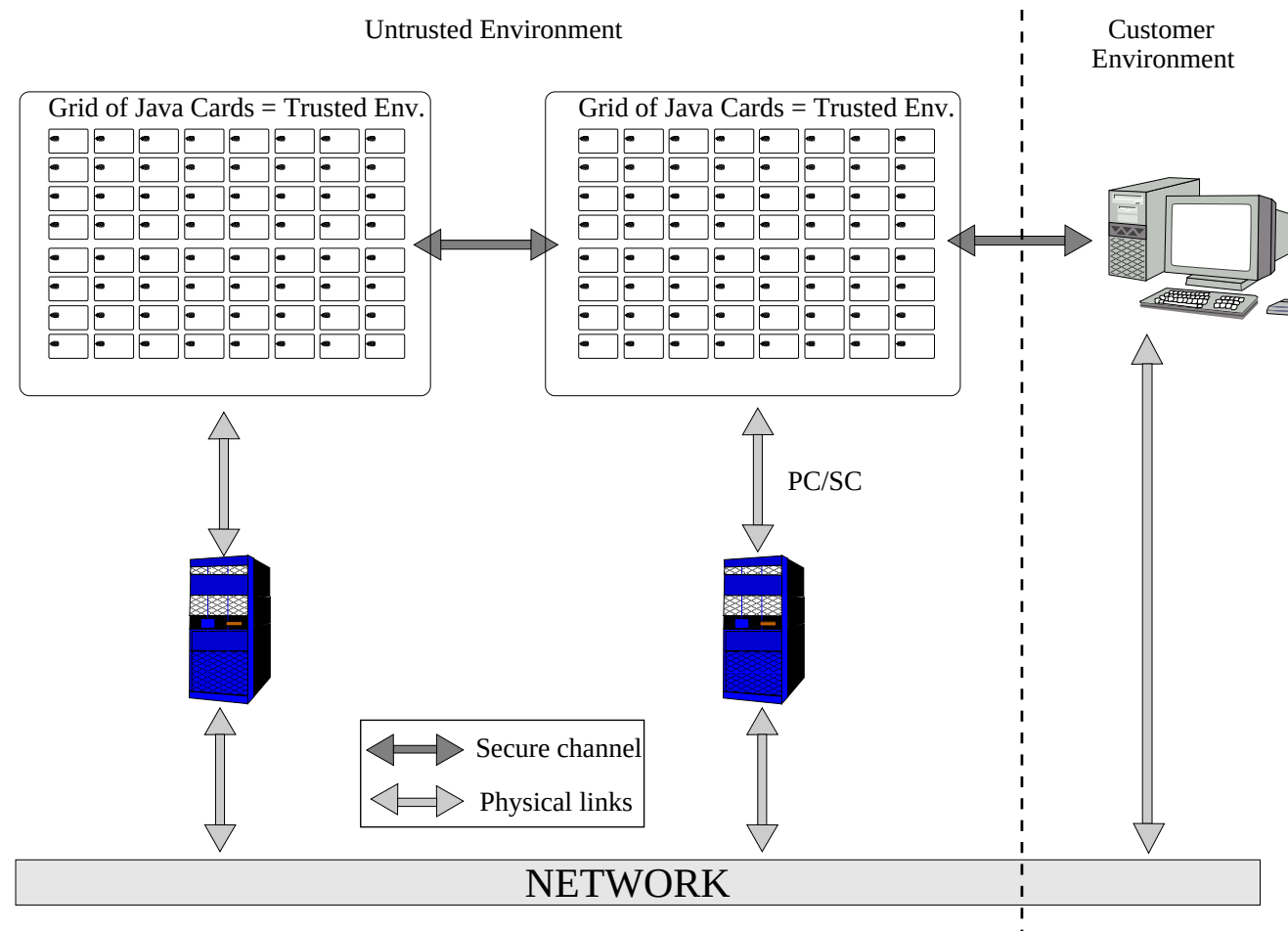
Java Card fournit la Push Security

Push Security : le fait d’être sûr que quand vous installez un programme sur une ressource de calcul du réseau, le code restera confidentiel et ne sera espionné par personne.

☞ JCRE Firewall

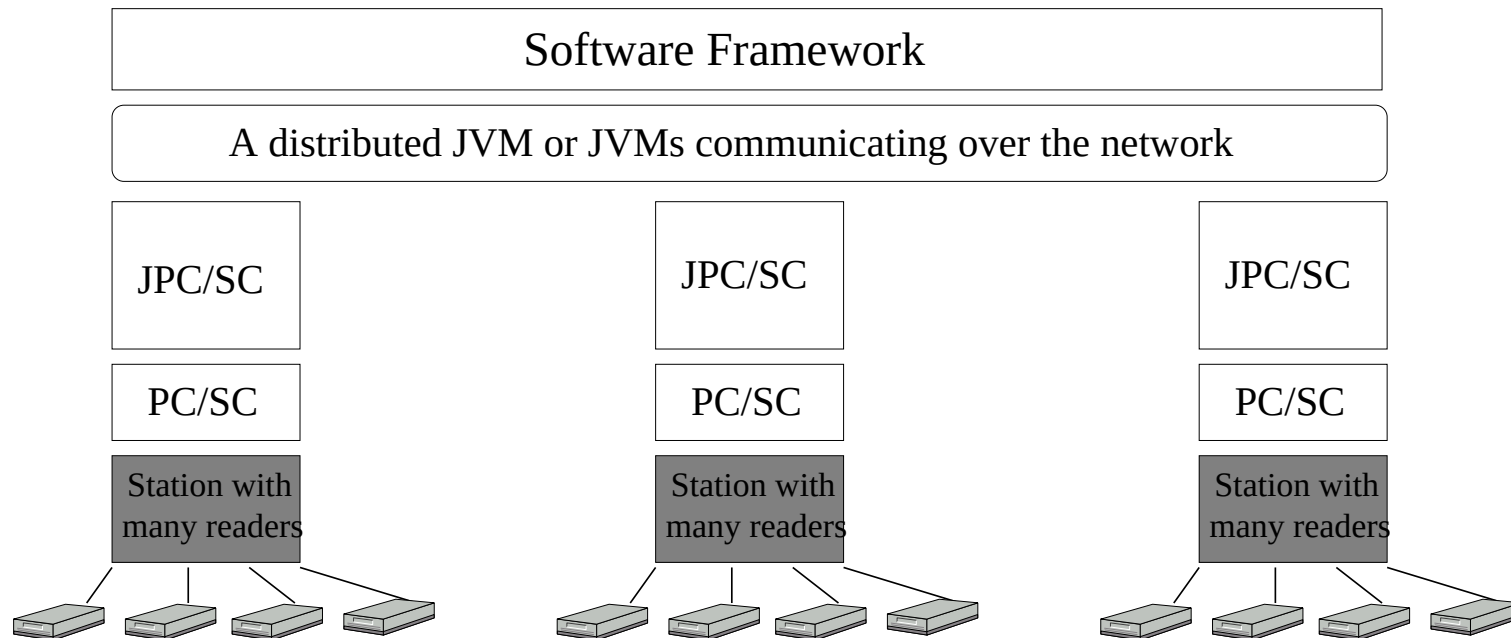
Le projet “Java Cards Grid”

Notre proposition



Le projet “Java Cards Grid”

La plate-forme logicielle



Le projet “Java Cards Grid”

PC/SC

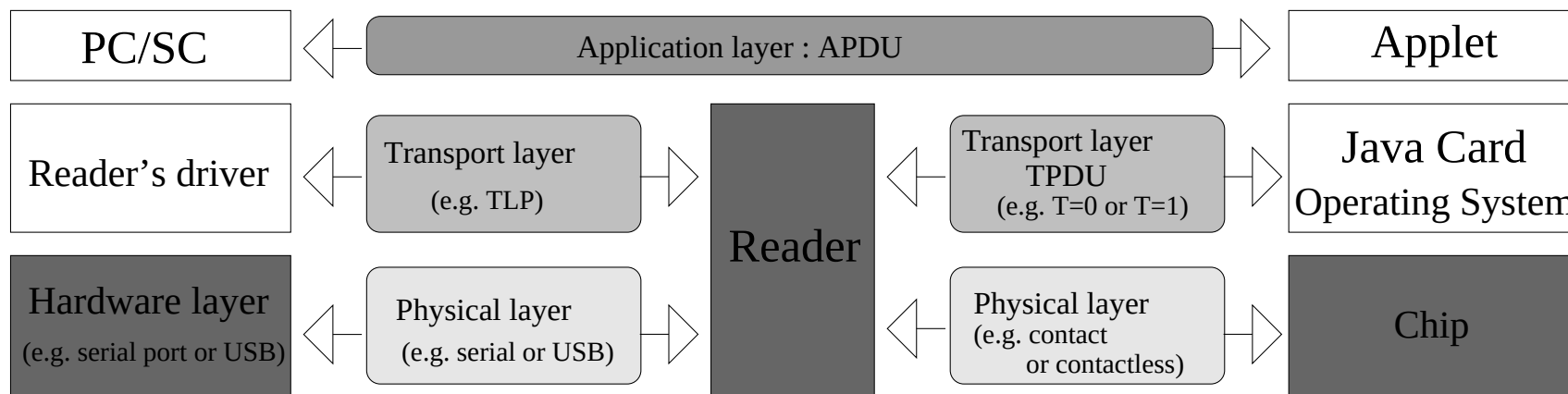


FIG. 6 – The PC/SC solution

PC/SC est un standard qui fournit :

- ☞ coté utilisateur une API de haut niveau pour dialoguer avec les lecteurs de carte à puce ;
- ☞ coté lecteur une interface à implémenter.

Le projet “Java Cards Grid”

La réalisation



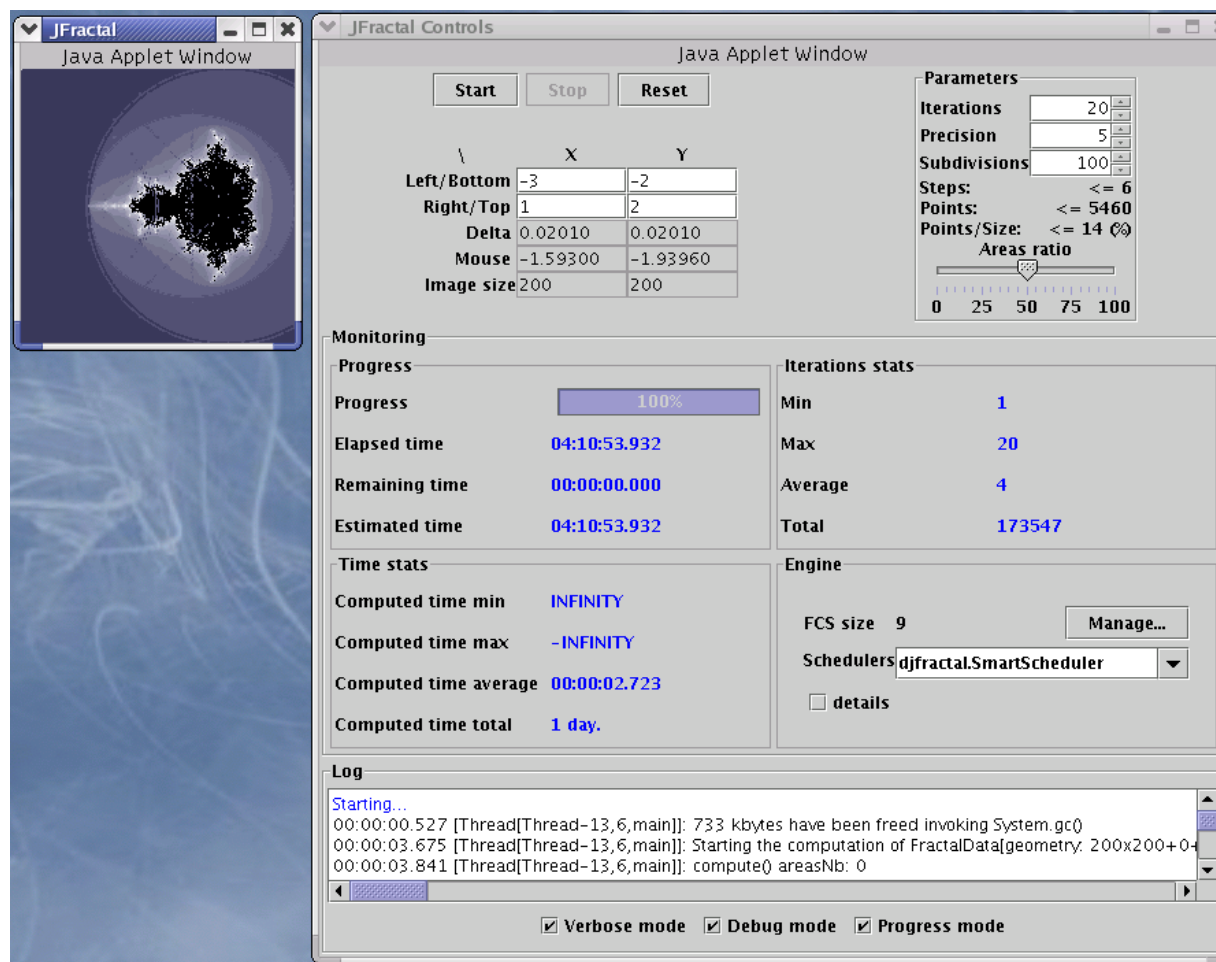
Le projet “Java Cards Grid”

Avantages de la plate-forme

- ➡ Peu coûteux : 1500 euros pour toute la plate-forme.
- ➡ Sécurité intégrée.
- ➡ Langage de programmation et runtime identiques sur toute la plate-forme : Java.
- ➡ Tests faciles de certaines caractéristiques.

Le projet “Java Cards Grid”

Première démonstration



Le projet “Java Cards Grid”

Quelques benchmarks

9 GemXpresso Pro R3 E32PK	3h22min
9 JCOP31bio	4h13min
9 GemXpresso Pro R3 64PK	6h16min
9 SmartC@fé Expert	6h29min

TAB. 1 – Benchmarks sans secure channel



Conclusions