

Java Card™

Damien Sauveron

damien.sauveron@unilim.fr

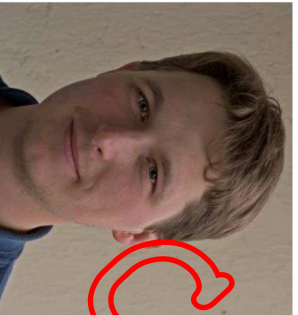
<http://damien.sauveron.fr/>

Intervenants dans le module

Maître de conférences Habilité à Diriger des Recherches

- Membre de l'équipe Cryptis et responsable du projet « Sécurité des Systèmes et Réseaux » (P.F. Bonnefoi, E. Conchon, K. Tamine)
- Thème de recherche : sécurité des systèmes embarqués et des réseaux mobiles

Pour me contacter :



Damien Sauveron
XLIM UMR 7252 CNRS -- Université de Limoges
123 avenue Albert Thomas
87060 Limoges Cedex, FRANCE

Email: Damien.Sauveron@unilim.fr
Web: <http://damien.sauveron.fr/>
Phone: +33 (0) 5 87 50 67 93

MCF à l'Université de Limoges depuis septembre 2006 – HDR (2014)

Visite postdoctorale au Smart Card Centre de l'Information Security Group du Royal Holloway University of London

Thèse au LABRI – Université Bordeaux 1 sur la Sécurité de la Technologie Java Card
Ingénieur R&D dans le CESTI de SERMA Technologies (Pessac)

Fondateur de WISTP (Workshop in Information Security Theory and Practice)
Président du WG11.2 Pervasive System Security de l'IFIP

Agenda

- La machine virtuelle Java
- La technologie Java Card
 - Introduction
 - Architecture
 - Mémoire
 - Les différentes versions
 - JCRE
 - JCVM
 - APIs

État de l'art des cartes avant Java Card

- La carte à puce est un véritable ordinateur utilisable comme serveur portable et sécurisé de données personnelles
- Elle est programmée comme un composant embarqué avec un code applicatif figé
- Elle est difficile à intégrer dans les systèmes d'informations
 - Car si l'application requiert de nouvelles fonctions carte il faut développer un nouveau masque (=> coûteux)
 - => Sinon il faut prévoir un masque qui accepte d'exécuter des programmes en EEPROM (rare et pas standardisé)
 - Car pas d'interface standard de communication avec les différents lecteurs (travaux en cours – problème complexe)
 - Communication via APDUs (très bas niveau)
- Problèmes à résoudre et/ou besoins à satisfaire
 - Permettre le développement de programmes pour la carte sans avoir besoin de graver un nouveau masque
 - Faciliter et accélérer les développements de codes dans la carte
 - Faire de la carte un environnement d'exécution de programmes ouvert (chargement dynamique de code)
 - Rendre plus souples et plus évolutives les applications carte
 - Faciliter l'intégration des cartes dans les applications
 - Faciliter et accélérer les développements d'applications clientes des cartes
- Besoins :
 - Diversification des sources cartes et des terminaux
 - Pérennité des développements
- Procédé de construction des applications
 - Permettre le développement «rapide» d'applications carte «évolué» par des «non spécialistes»
 - But : de nouvelles applications pour de nouveaux marchés !

Qu'est ce que Java Card ?

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

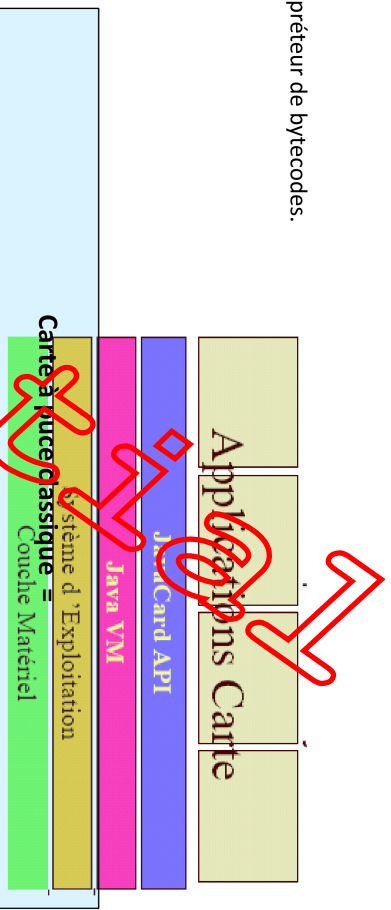


autres périphériques à mémoire limitée (par exemple iButton)



De manière intuitive :

Il s'agit d'une carte basée sur un interpréteur de bytecodes.



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

5

Historique 1/2

- En Novembre 1996, un groupe d'ingénieurs de Schlumberger cherche à simplifier la programmation des cartes à puces tout en préservant la sécurité (Cyberflex 1.0 au CNIT – Cartes'96).
==> la spécification Java Card 1.0
- En Février 1997, Sun et Gemplus se joignent à Schlumberger pour cofonder le « Java Card Forum ».
 - Consortium de fabricants :
 - Cartes (Gemplus, Schlumberger, etc)
 - Informatique : IBM, Sun
 - Matériel : DEC, Motorola
 - Utilisateurs : Banques (Citibank, etc.)
 - Buts :
 - promouvoir Java Card
 - Faire des choix technologiques communs
 - Contre pouvoir à Sun (officieusement)
 - Solutions :
 - Un comité technique
 - Un comité business
 - <http://www.javacardforum.org/>
- En Novembre 1997, Sun présente les spécifications Java Card 2.0.
- 1998 : Cyberflex 2.0, GemXpresso, ...

Historique 2/2

- 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.
- En Mai 2000, sort une petite correction ==> 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, de nouveaux algorithmes cryptographiques, ...
- En octobre 2003, spécifications Java Card 2.2.1.
- En novembre 2005, la version 2.2.2 est Public Review Draft
- En décembre 2005, les licenciés sont :
 - ARM, Aspects, Axalto, CCL/ITRI, Fujitsu, Gemplus, Giesecke & Devrient GmbH, GoldKey Technology, HismarTech, i'M Technologies Ltd., IBM, incard, KEYTechnology, LogisSmart Card, Microelectrónica Española, Oberthur Card Systems, ORGA Kartensysteme, SAGEM, Setmepa, Setec, Sharp, Smart Card Laboratory Inc., SSP Solutions, Inc., STMicroelectronics, Toppan Printing, Trusted Logic. + d'autres qui préfèrent rester anonymes.

JavaCard Forum Members

(as of 10th Jan 2005)



Giesecke & Devrient



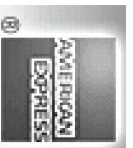
Datacard Group



ARM



The Smart Card Integrator



HITACHI



Axalto
A MasterCard company



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 sous-jacente du système des cartes à puce.

L'indépendance au hardware réalisée grâce au langage Java

==> « Write Once, Run Anywhere »

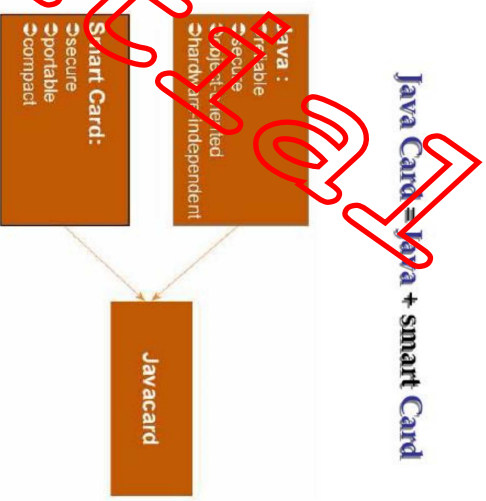
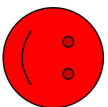
La gestion dynamique de multiples applications

==> possibilité de mises à jour des applications de la Java Card sans avoir besoin de changer de cartes
==> possibilité d'offrir des nouveaux services

La compatibilité avec les standards existants pour les cartes à puce

- ISO7816 :
- Protocole de communication
 - Protocole de haut niveau pour la gestion de fichiers, etc

On verra les inconvénients plus tard.



9

La plate forme Java

- Java est un langage de programmation
- Voir le « white paper » de J.Gosling

Un programme Java est compilé et interprété

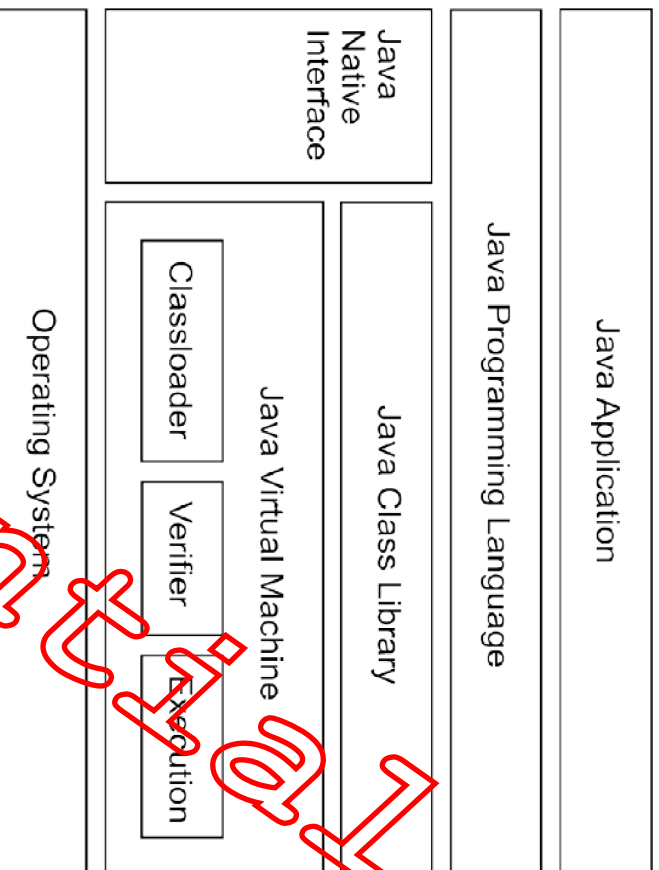
- Java est une plate-forme

La plate-forme Java, uniquement software, est exécutée sur la plate-forme du système d'exploitation,

La « Java Platform » est constituée de:

- La « Java Virtual Machine » (JVM)
- Des interfaces de programmation d'application (Java API),
- Une interface pour les méthodes natives (JNI)

Architecture de la plate-forme



Les différentes plates-formes



Les composants

- Le langage de programmation,
- La librairie,
- La machine virtuelle

Un jeu d'instruction interprétable (*byte codes*)

Un format de chargement, pas forcément équivalent au format d'exécution le *class file*

Un algorithme de vérification des règles de typage Java ramenées au byte code, obligatoire pour les applets,

Un compilateur à la volée.

Java langage de programmation

- Java est un langage de programmation particulier qui possède des caractéristiques avantageuses:

Simplicité et productivité:

- Intégration complète de l'OO
- Gestion mémoire (« Garbage collector »)

Robustesse, fiabilité et sécurité (langage fortement typé)

Indépendance par rapport aux plates-formes

Distribution et aspects dynamiques

Performance

Machine virtuelle Java

- La JVM définit:

Les instructions du CPU

Les différents registres (PC, SP,...)

Le format des fichiers .class

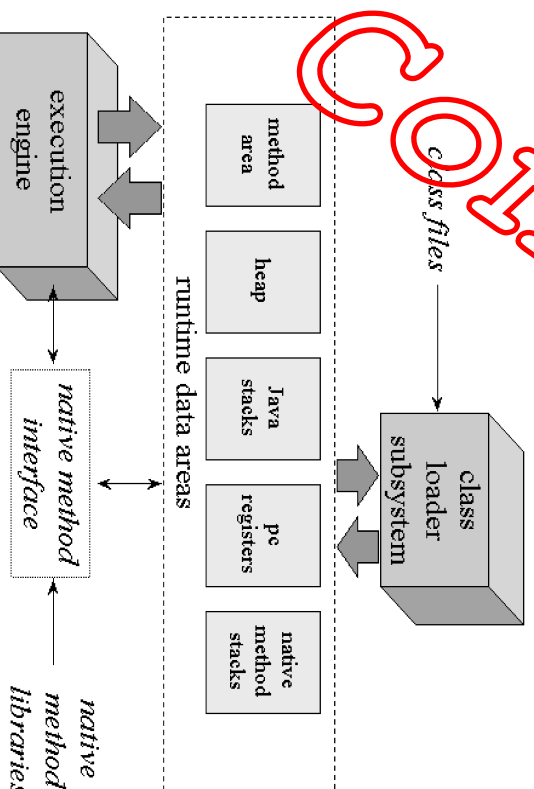
La pile Java et la pile C, en RAM,

Le tas (Heap) des objets « garbage-collectés » en RAM aussi,

L'espace mémoire partagé:

- Stockage des classes et des données statiques,
- Stockage des objets (les instances) dans le tas.

Machine virtuelle Java



Machine Virtuelle Java

- Trois tâches principales:
Charger le code (class loader)
Vérifier le code (byte code verifier)
Exécuter le code (runtime interpreter)
- D'autres threads s'exécutent:
Garbage collector
(JIT compiler)

Sécurité par le typage

- En Java les pointeurs existent mais:
la manipulation explicite d'un pointeur en tant que tel est impossible en Java **le compilateur vérifie** cette règle ainsi que la VM. Ainsi le code suivant n'est pas compilable:

```
p=new Object();  
p++;
```

tous les objets Java sont en fait des pointeurs (que l'on nomme références).

- Les variables locales (variables de types primitifs et références) sont allouées sur la pile (ou dans les registres) et les objets dans le tas.
- On ne contrôle pas la zone d'allocation ni quand sera effectuée la désallocation.

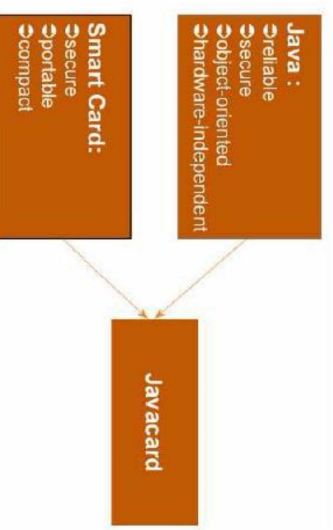
Sécurité par le typage

- Lors du chargement **la VM vérifie** si, au niveau byte code, il n'y a pas de transypage ne correspondant pas aux règles de Java : c'est le rôle du vérifieur de type.
- Après, une référence n'a PLUS besoin de vérifier si le typage est correct.
- Donc la sécurité par le typage repose sur:
Un algorithme dans le compilateur,
Un autre dans la VM.

Confidentiel

Java Card

Java Card = Java + smart Card



Attention

- Les informations concernant Java Card 3.0 Classic Edition (pas Connected Edition) et Java Card 2.X

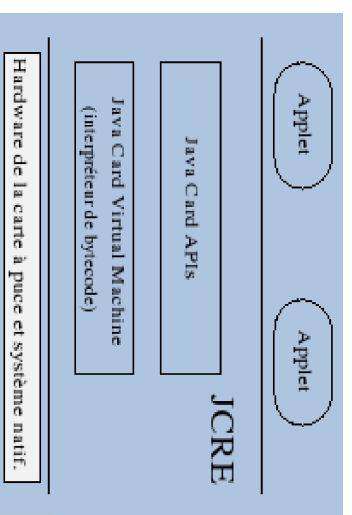
Présentation générale de Java Card

- Un ensemble de spécifications
Publiées par Oracle
Basées sur la plateforme Java
<http://www.oracle.com/technetwork/java/embedded/javacard/index.html>
- Découpées en trois parties

Application programming interfaces (APIs)

Execution environment (JCRE)

Virtual machine (VM)



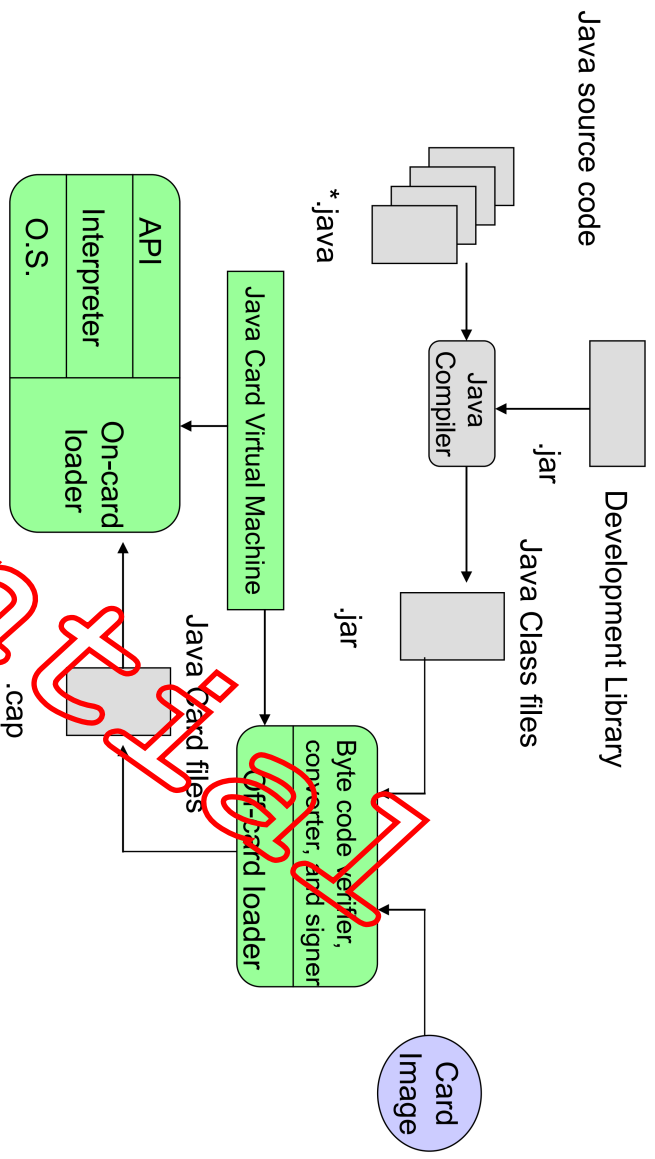
Agenda

- **Introduction**
- JCRE (Runtime Environment)
- JCVm (Virtual Machine)
- APIs

Un autre point de vue

- Architecture
- Mémoire
- Les différentes versions

L'architecture Java Card



Architecture

- Converter (part of the JDK):
 - Class file format takes too much space on Java Card
 - Produces a format that fits the Smart Card constraints,
- Tokenization of the format
 - Need a representation of the content for pre-linking
 - The converter uses all class files of a package and all export files of ALL imported packages,
 - Output an Export file and a Cap file
- Conversion process
 - Verifies the Java Card language restriction
 - Optimize byte code
 - Invokes the off card verifier

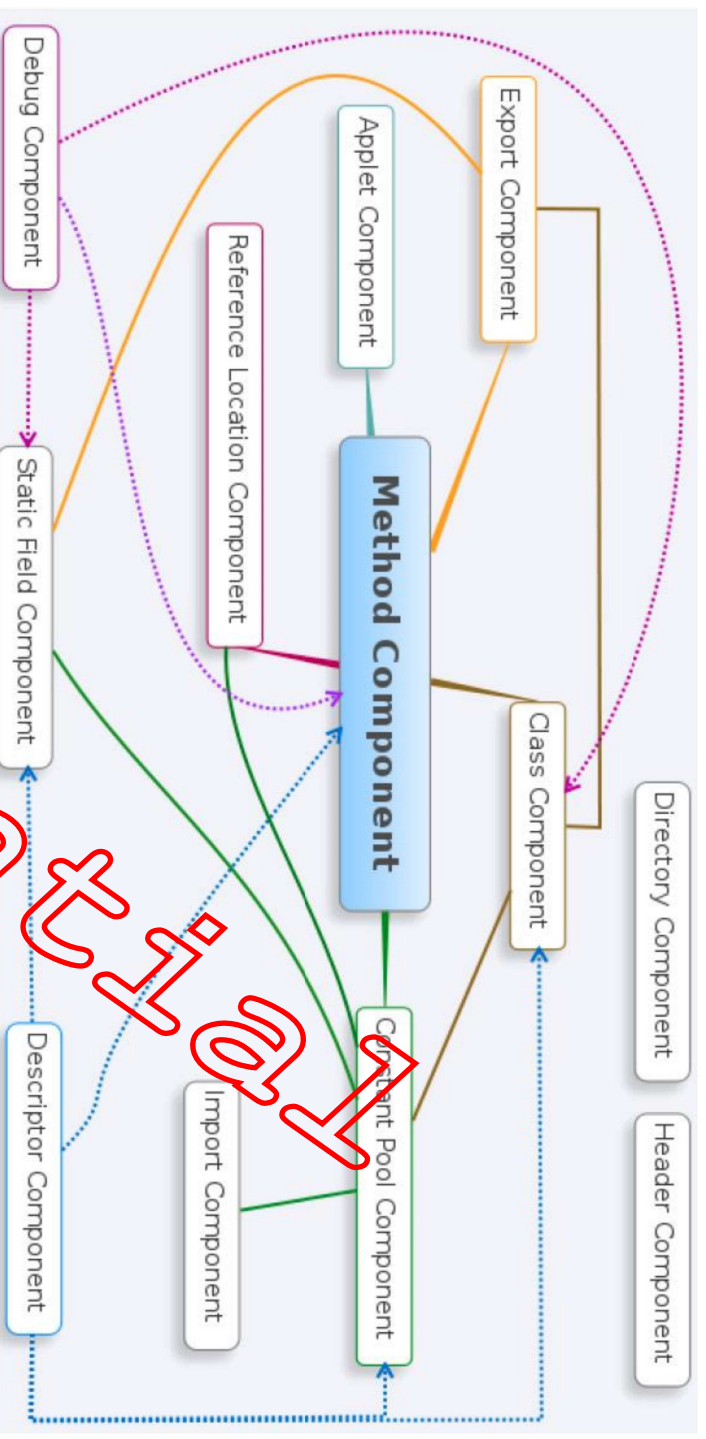
Two specific file formats

- The CAP (Converted Applet) file format
 - **Contains all the classes from one package**
 - Semantically, is equivalent to a set of class (.class) files
 - Syntactically, differs a lot from class (.class) files
 - All “string names” are replaced by “token identifiers”
- The EXP (Export) file format
 - Maintains the consistency between the originated class (.class) files and the resulting CAP file
 - Only for public (exported) data
 - Contains API information for a package of classes (access scope, class name, method signature, ...)
 - **Can be freely distributed**, used during pre-linking phase
 - Not loaded into the card

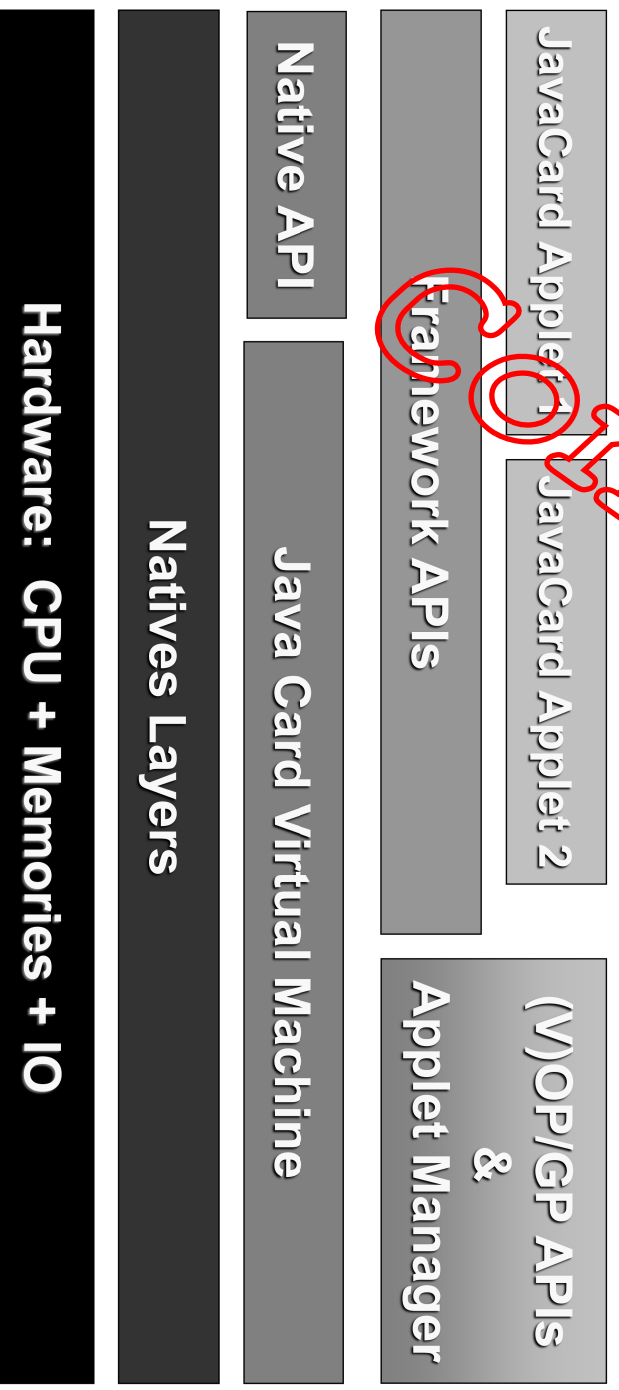
The CAP file

- Contains an executable representation of package classes
- Contains a set of components (11)
- Each component describes an aspect of CAP file
 - Class info
 - Executable byte code
 - Linking info,...
- Optimized for small footprint by compact data structure
- Loaded on card

Interdependences

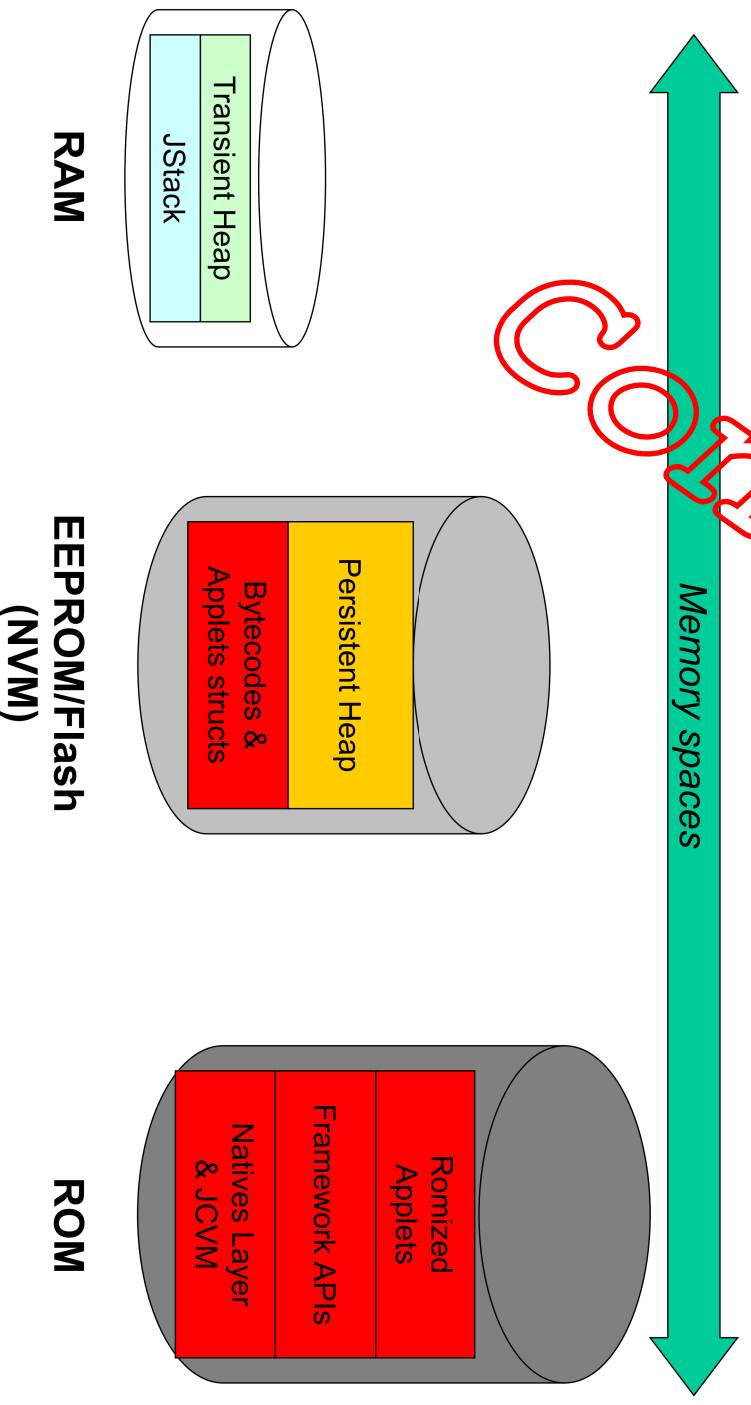


Java Card architecture



Java Card memory model

- By default, all objects are *implicitly* persistent
 - Because we have few RAM
 - Objects must survive between two sessions
- Some arrays can be *transient*
 - For efficiency and security reasons
- Transactional mechanisms are provided
 - All write operations on persistent memory are atomic
 - At the programming level a mechanism to handle transactions is also available



Java Card 2.1

- File format of applet:
 - Standardized
 - file .cap identical to .class excepted:
 - One .cap file per Java package
- « Firewall » between applets
 - The virtual machine must ensure that the code doesn't run out of its execution space (context)
 - Means to switch the execution context
 - Entry point object and global array can be accessed by applets (e.g., APDU)
 - JCRE can access each object
 - Interaction between applet through a shareable interface
 - System.share (Object ...) method suppressed

Java Card 2.2

- Load process of the .cap file standardized
 - Interoperability with JC 2.1 stops at the smart card loading,
- Object, applet and package deletion
- On card verifier (optional),
- Logical channels,
- Optional Garbage collector on demand,
- Support elliptic curves and AES algorithm,
- JC-RMI
 - Skeleton/stub generator 'a la RMI' hide the APDU encoding-decoding
 - Rapid development and integration of SC applets

Agenda

- Introduction
- JCRE (Runtime Environment)
- JCVN (Virtual Machine)
- APIs

Execution environment (JCRE)

- Define how a Java Card manages its resources
- Define constraints on the Java Card operating system
 - Applet lifecycle (installation, register and deletion)
 - Logical channels and applet selection,
 - Transient objects,
 - Applet isolation (firewall) and sharing,
 - Transaction and atomicity,
- RMI
- The JCRE is at the heart of a Java Card

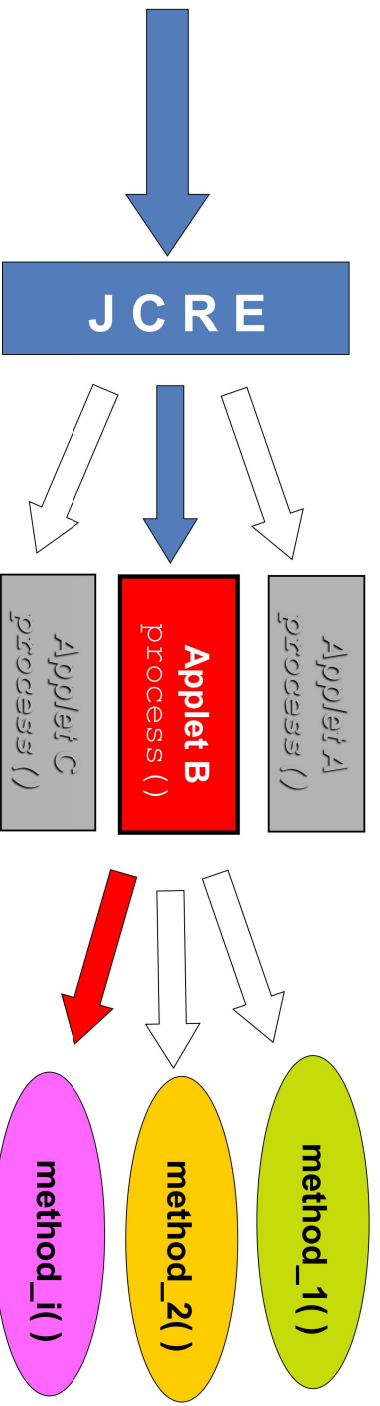
APDU commands



- 2 types of APDU can be sent to the card:
 - **OS/Administrative** commands
 - OS commands available in JCRE and CM
 - Select, Load, Install ...
 - Administrative commands specified by SC manufacturer
 - Get Info, ...
 - **Applicative** commands
 - specific to the JC applets loaded in the card
 - eg: debit, credit, getbalance for an e-purse applet
 - eg: create file, update file for the GSM applet

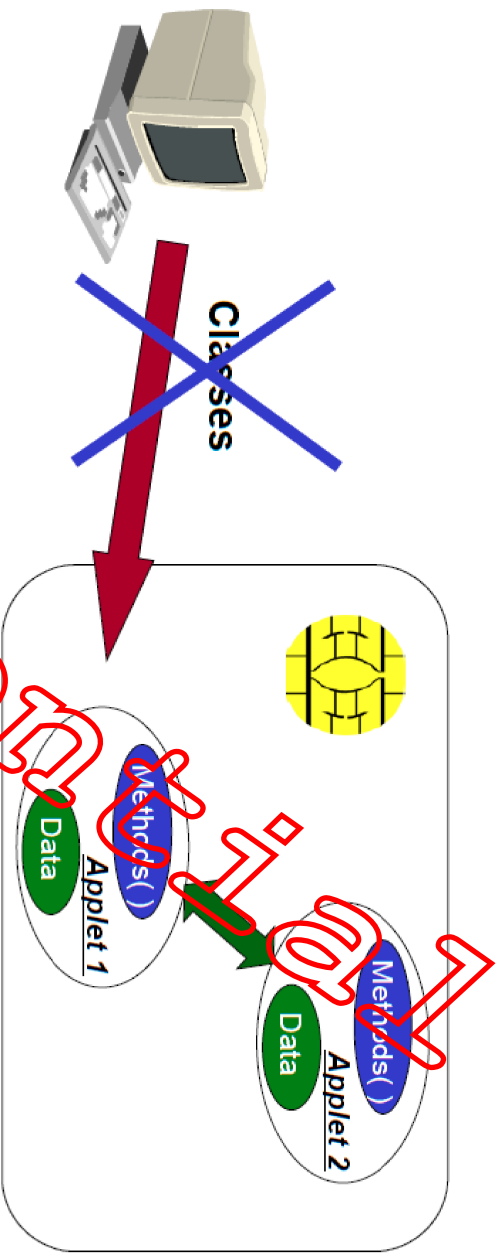
2 command dispatchers

- **JCRE's task: main dispatcher**
 - Route the incoming commands to the JCVM and the selected applet
- **Applet designer's task: second dispatcher**
 - Implement the applet's command dispatcher (extraction of the header information and call of the associated method)

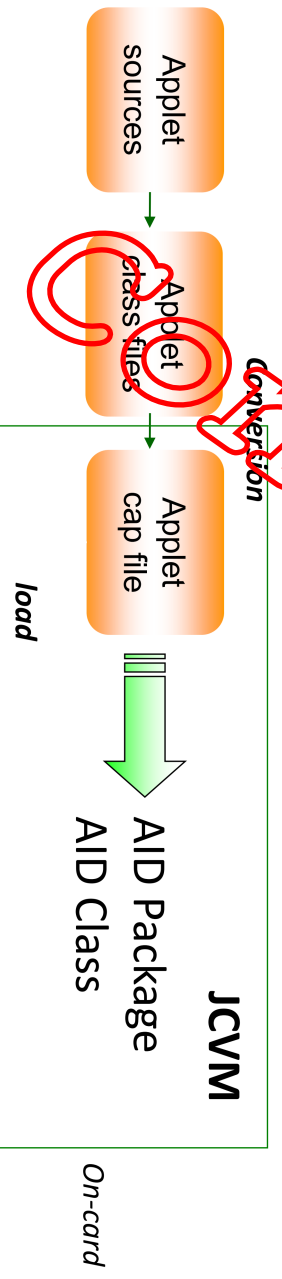


Unsupported features Dynamic class loading

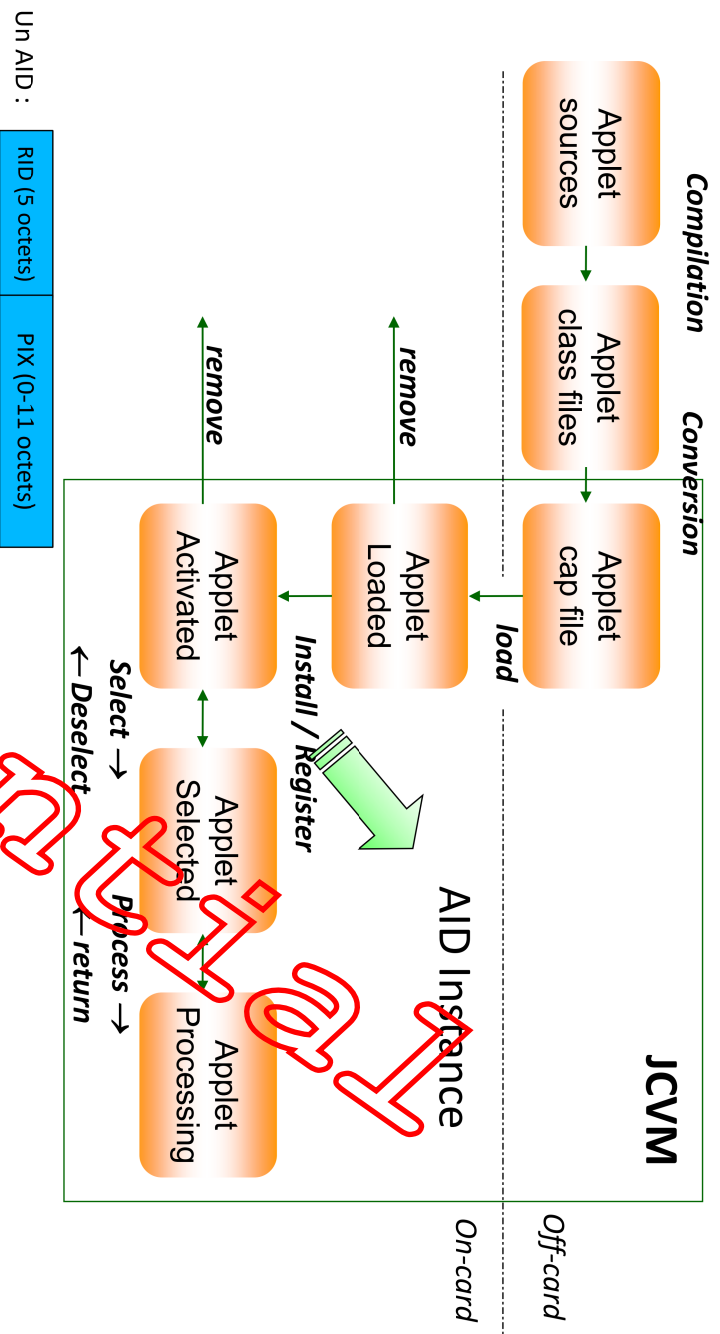
- Classes are **statically linked** before being downloaded
- No way to download classes on the fly as needed...
- Applets only refer to classes which already exist on the card



Java Card applet life cycle



Java Card applet life cycle

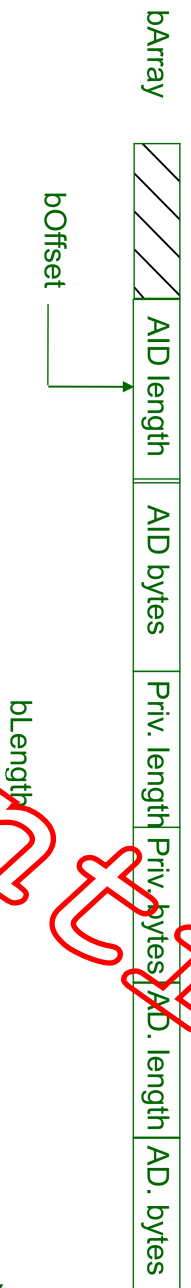


Methods of the applet

- An applet extends `javacard.framework.Applet`,
- It **MUST** implement the method `install`
- It **MUST** call the method `register`
- It **MUST** implement the method `process`
- It **SHOULD** implement `select`, `deselect`
- The `process` method can only be called if the applet has been selected

install method

- JCRE calls this static method first and gives
 - Applet instance AID
 - Applet privilege
 - Applet parameters
- `install` method creates an instance of an `Applet` subclass
 - Performs any necessary initializations,
 - Must call once one of the `register` methods
 - If no parameter is provided, only one installation



register method

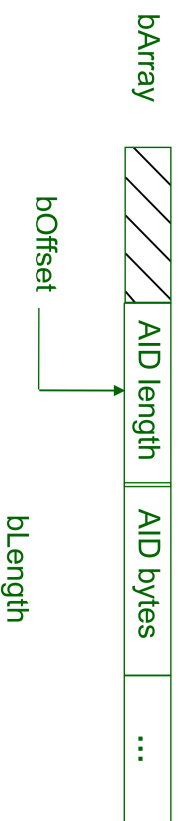
- Used by the applet to register this applet instance
- Interacts with the Java Card Runtime Environment
 - `final void register()`
 - Assign the applet instance AID with default AID bytes defined in CAP file
 - `final void register(byte [] bArray, short bOffset, byte bLength)`
 - Assign applet instance AID with the specified AID bytes

- Warning: when receiving the byte array as parameters of the `install` command, the length is sent before the AID

```
public static void install(byte[] bArray, short bOffset, byte bLength)
throws IOException {
```

=>

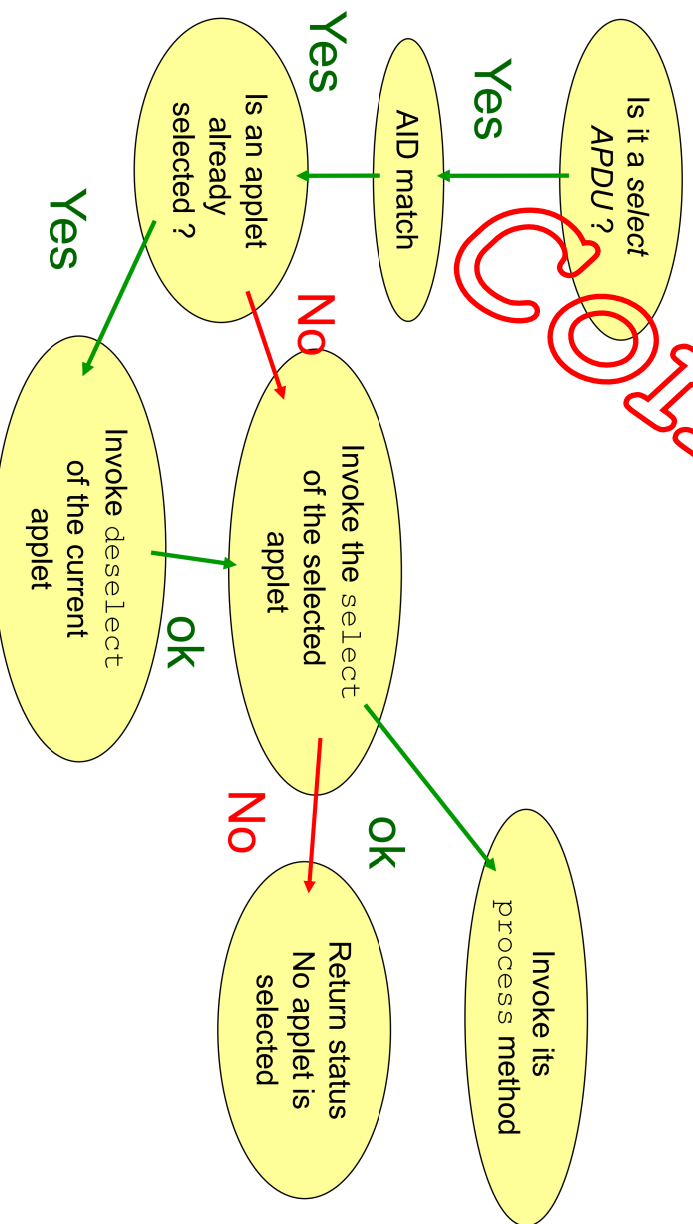
```
register(bArray, (short) (bOffset + 1), (byte) bArray[bOffset]);
```



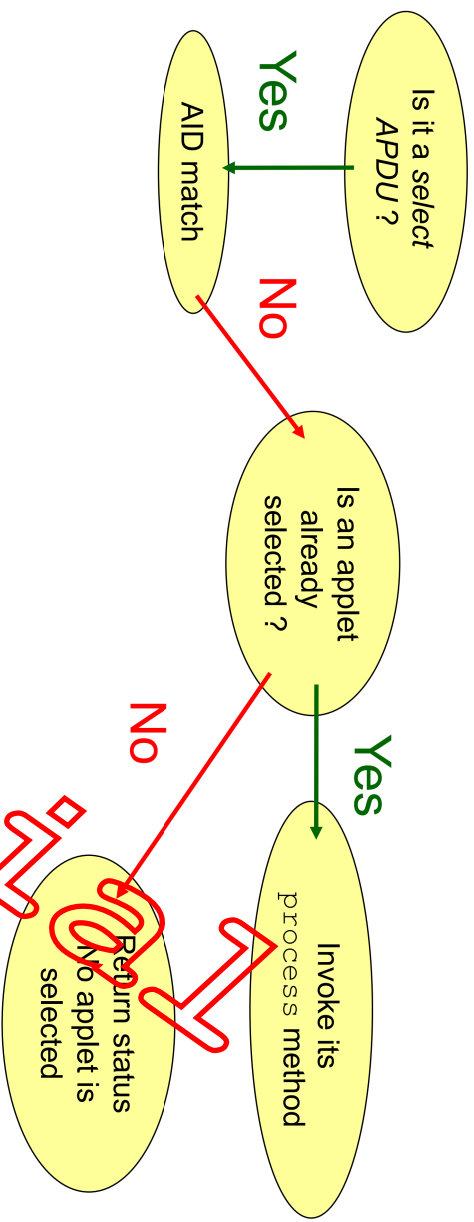
process method

- Contains the core application code of the applet
- Handles all the incoming APDU messages for the applet selected
- Called by the JCIRE
- Upon normal return from this method the Java Card runtime environment sends the ISO 7816-4 success status word 90 00
- If it throws an exception the JCIRE sends the associated reason code as the response status

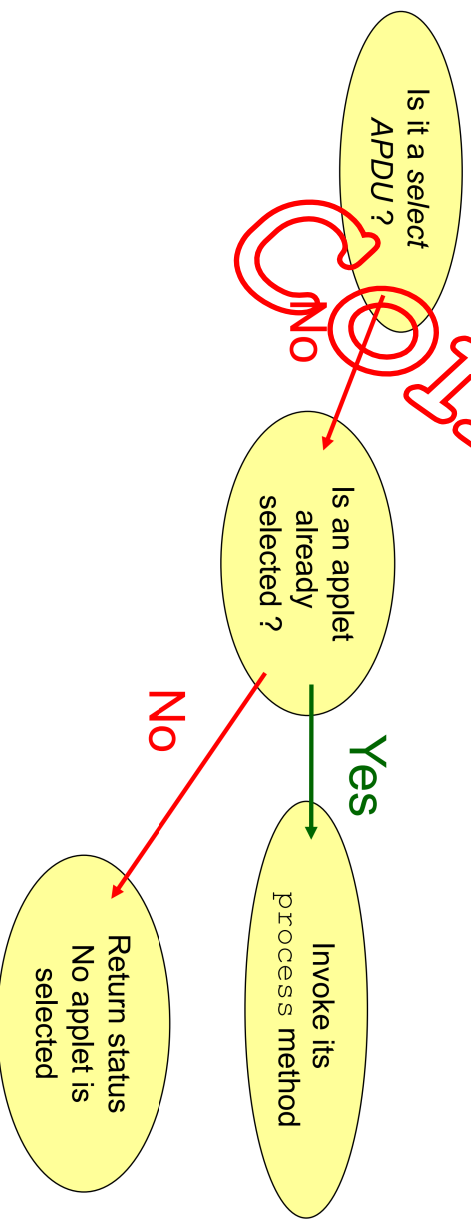
Selection protocol



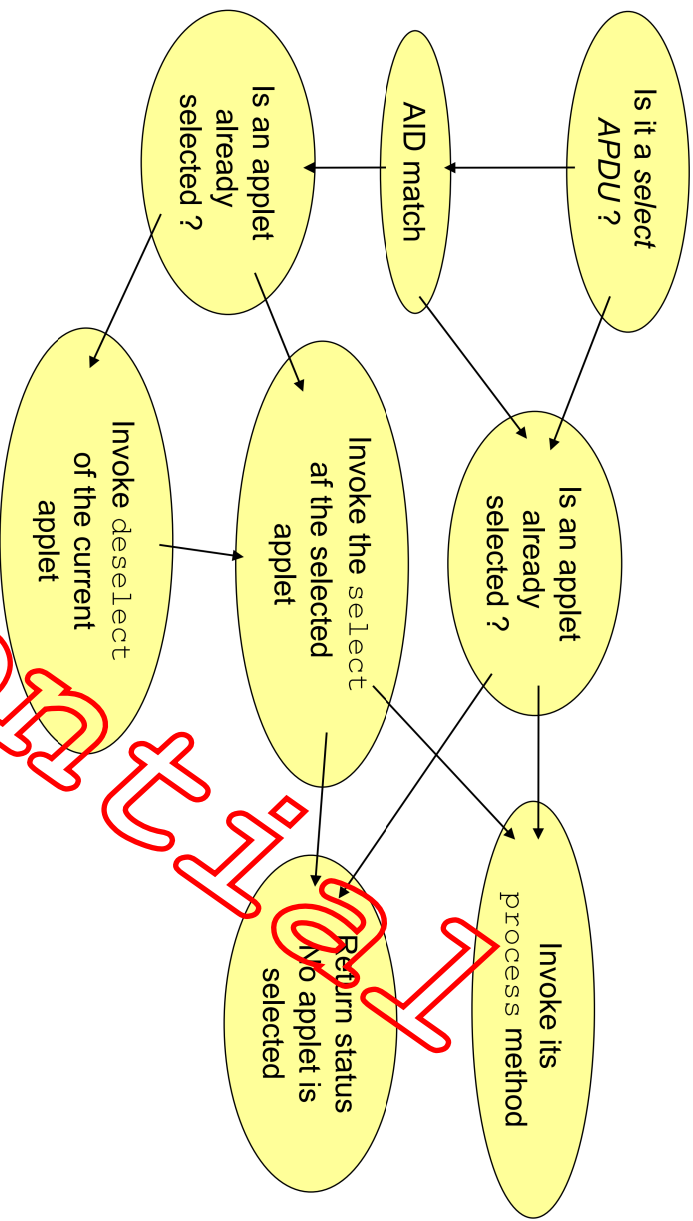
Selection protocol



Selection protocol



Selection protocol



Agenda

- Introduction
- JCRE (Runtime Environment)
- **JCVM (Virtual Machine)**
- APIs

Virtual machine (VM)

- This specification is made of several parts
 - The definition of the Java language subset that is supported
 - The definition of 2 specific file formats
 - The definition of a specific instructions set

Java language subset

- Supported
 - boolean, byte, short, int (optional)
 - Objects
 - Arrays (max size: 2^{15})
 - Virtual methods
 - Dynamic allocation
 - Packages
 - Exceptions
 - Interfaces
- Not supported
 - float, double, long
 - char, String
 - Multi-dimensional arrays
 - **Garbage collector**
 - Finalization
 - Threads
 - Dynamic loading of classes
 - Security manager

BC interpretation

- It is the execution engine for the byte code loaded into the card,
- It controls byte code execution, memory allocation and participate to the security through the firewall,
- Often it includes more tests than the firewall due to the absence of BC verifier...

Note that hardware processors executing Java Card bytecode exist (e.g. ST22).

Agenda

- Introduction
- JCRE (Runtime Environment)
- JCCVM (Virtual Machine)
- **APIs**

APIs (1/2)

- Define a set of Java classes
 - Used to develop Java Card services (applets)
 - Dedicated for a smart card environment
- Those classes comply with current standard
 - Business Java (**java.lang.Object**)
 - ISO-7816 (e.g., APDUs)
 - Cryptographic
- Those classes do not define by themselves services

APIs (2/2)

- Contains 3 mandatory packages
 - **java.lang**
 - Basic classes of the language
 - **javacard.framework**
 - Framework of classes and interfaces for the core functionality of an applet (AID, APDU, Applet, ISO7816, PIN, OwnerPIN, JCSsystem, Util, and exceptions classes)
 - **javacard.security**
 - Core classes dedicated to cryptographic services (public/private key, random number generator,...)
- And one optional package
 - **javacardx.crypto**
 - Implementation classes for ciphering/deciphering (strong cipher)

java.lang

- `Object { public Object();
public boolean equals(Object obj); }`
- `Throwable { public Throwable(); }`
 - `Exception`
 - `RuntimeException`
 - `ArithmeticException`
 - `ClassCastException`
 - `NullPointerException`
 - `SecurityException`
 - `ArrayStoreException`
 - `NegativeArraySizeException`
 - `IndexOutOfBoundsException`
 - `ArrayIndexOutOfBoundsException`

javacard.framework

- `class java.lang.Object`
- `class javacard.framework.MIB`
- `class javacard.framework.APDU`
- `class javacard.framework.K.AApplet`
- `class javacard.framework.JCSystem`
- `class javacard.framework.OwnerPIN` (implements `javacard.framework.PIN`)
- `class java.lang.Throwable`
 - `class java.lang.Exception`
 - `class javacard.framework.CardException`
 - `class javacard.framework.UserException`
 - `class java.lang.RuntimeException`
 - `class javacard.framework.CardRuntimeException`
 - `class javacard.framework.APDUException`
 - `class javacard.framework.ISEException`
 - `class javacard.framework.PINException`
 - `class javacard.framework.SystemException`
 - `class javacard.framework.TransactionException`
- `class javacard.framework.Util`

- `interface javacard.framework.ISO7816`
 - `interface javacard.framework.PIN`
 - `interface javacard.framework.Shareable`
- } Optionally used by the developer

- `class javacard.framework.ISEException` used very often in applet development

javacard.framework

- **public class AID**

Encapsulates constants related to ISO7816-5

Mainly used at beginning of "communication" between applets

- **public class Util**

- Functions for **byte** array manipulation and comparison
 - **arrayCompare()** compares 2 arrays (*in constant time*)
 - **arrayCopy()** copies 1 array into another *atomically*
 - **arrayCopyNonAtomic()** copies 1 array into another *non-atomically*
 - **arrayFillNonAtomic()** fills an array with a **byte** value *non-atomically*
- Functions for type conversion (short/byte)
 - **short getShort()**

 - **short makeShort()**

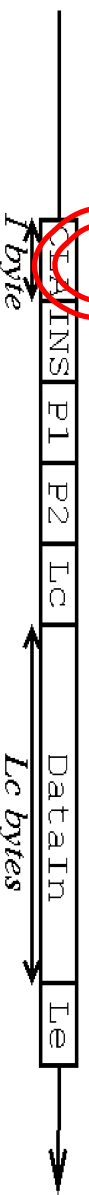
 - **setShort()**


javacard.framework

- **public final class APDU**

- Encapsulates 7816-4 APDU elements in an I/O buffer
- APDUs (Application Programming Data Units) define the format of the «data packets» exchanged between a reader and a card:
 - APDUs are defined for both commands and responses to hide the low level details of the protocol used (T=0 ou T=1)
 - Status words SW1 et SW2 are standardized (OK = 0x9000)

Command APDU with data sent to the card

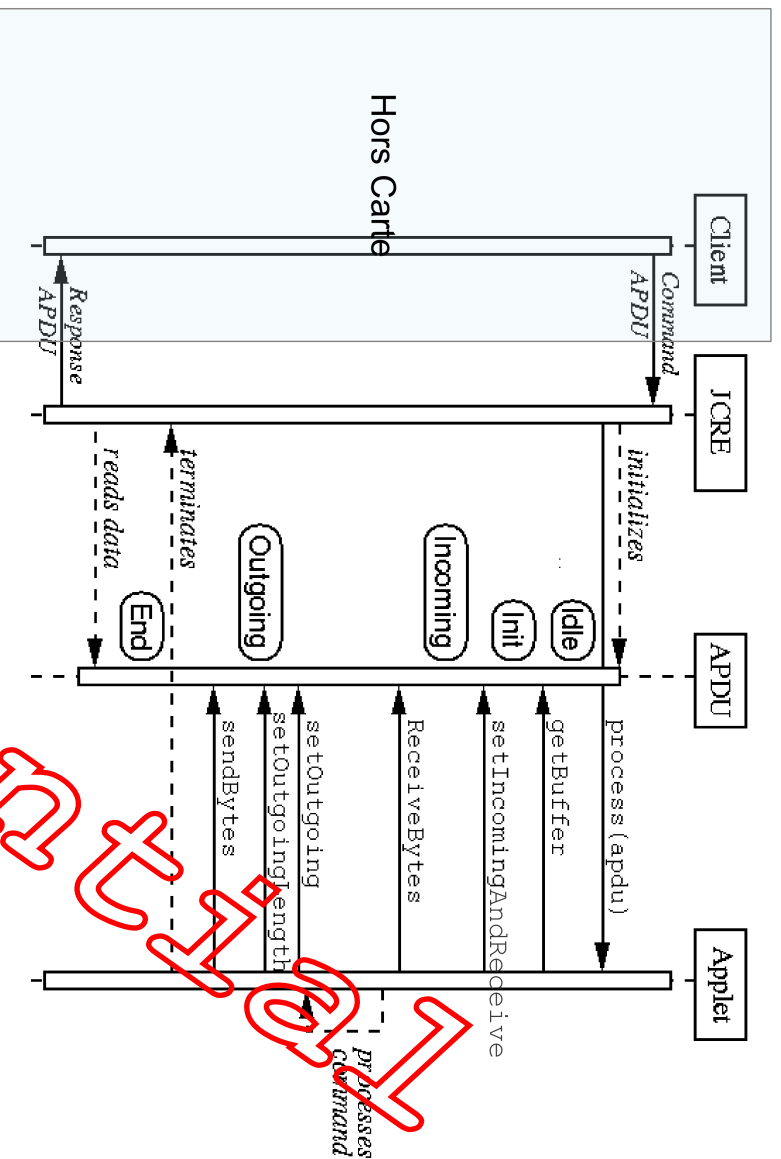


Response APDU with data returned by the card



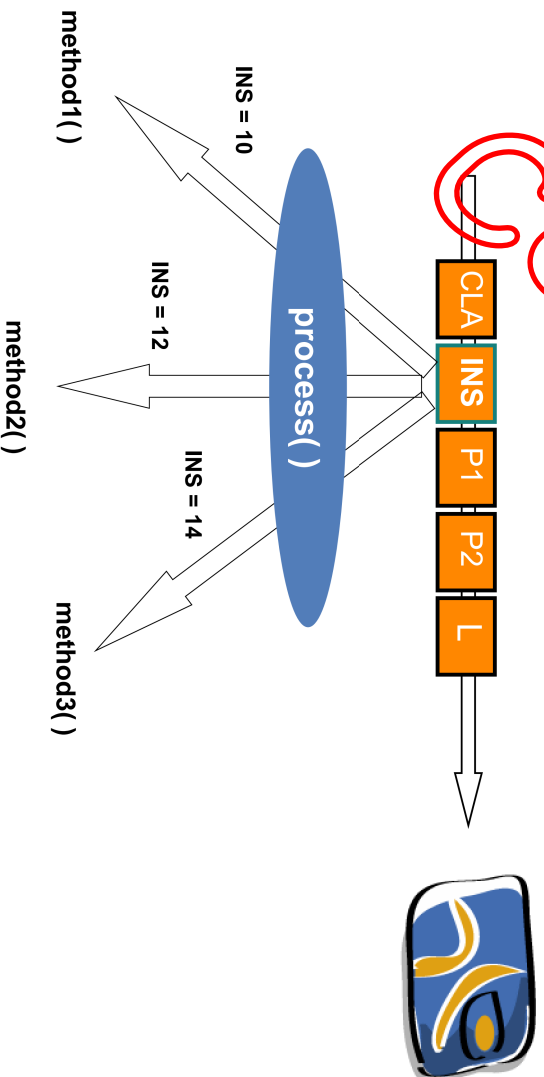
- Provides methods for **receiving** data and **sending** data
- Designed to be protocol independent
- The incoming / outgoing APDU data size may be bigger than the APDU buffer size and need to be read/write in portions by the applet.
- *APDU objects are owned by the JCRC (Temporary JCRC entry point objects)*

Handling the APDU class



Handling the APDU class

- process (APDU (apdu)) method is called
- apdu.getBuffer()
- CLA is checked and then INS is examined



javacard.framework

- **public interface ISO7816**

Contains only static constants

Public final static field with **SW_** prefixes define status words (standardized status response of ISO7816-4)

- e.g.: **SW_WRONG_P1P2**

// Incorrect parameters (P1,P2) = 0x6B00

Fields with **OFFSET_** prefixes define constants/offset to use as index in the APDU buffer (byte array)

- e.g.: **OFFSET_CLA**

// APDU header offset: CLA = 0

Fields with “**_00**” suffix require the low order byte to be customized appropriately

- e.g.: **SW_BYTES_REMAINING_00**

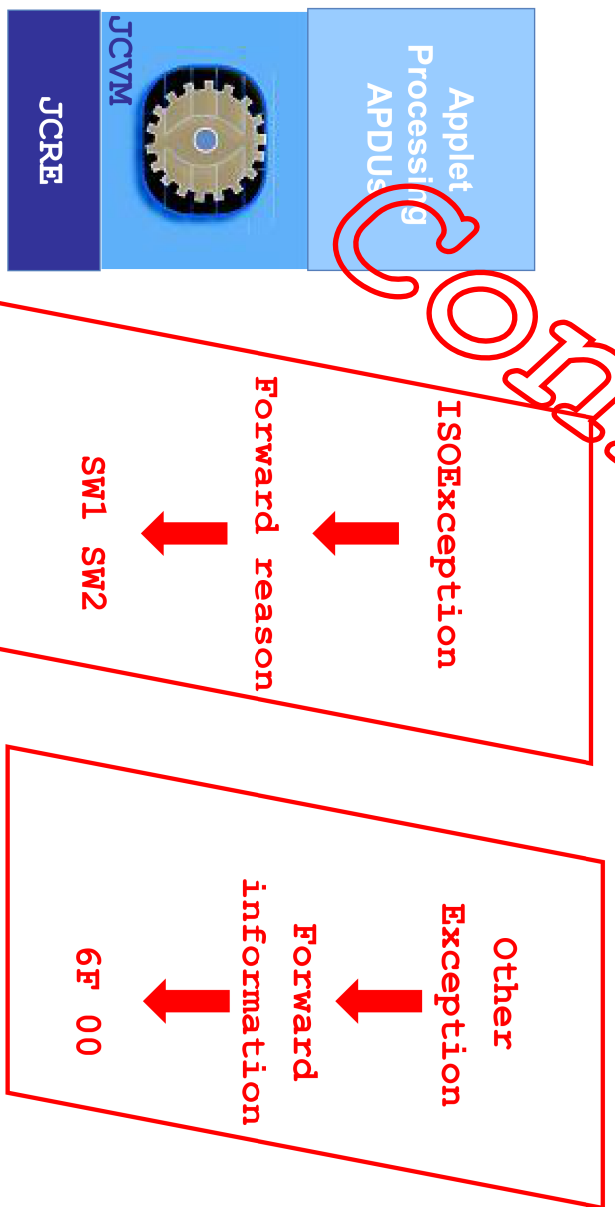
// Response bytes remaining = 0x6100

- **public class ISOException**

Encapsulates an ISO 7816-4 response status word as its reason code

throwIt(short reason) allows to output a status word error

Java Card Exceptions



javacard.framework

- **public interface PIN**
 - Define a PIN behavior and a PIN value
 - Try limit maximum trials of an incorrect PIN before being blocked,
 - Max PIN size, the maximum length of PIN,
 - Try counter, the remaining number of trial,
 - Validated flag => true if a valid PIN has been presented. Flag is reset on every card reset.
 - If a transaction is in progress **update of the try counter shall not participate to the transaction,**

javacard.framework

- **public class OwnerPIN**
 - Implements interface PIN
 - Full rights for PIN code management
 - Update / Read capabilities -> for the code "owner" only
 - Features:
 - direct modification of the validation flag
 - know whether or not the code has been validated
 - securely implemented against attacks

Methods available:

- check()
- isValidated()
- resetAndUnblock()
- setValidatedFlag()
- reset()
- getTriesRemaining()
- getValidatedFlag()
- update()

javacard.framework

public final class JCSystem

- Static methods (natives) for interaction with the JCRE
- Transactions
 - Only one level of transaction allowed
 - **beginTransaction()** , **commitTransaction()** , **abortTransaction()**
 - Limited by the RAM and EEPROM capacity
- Transient array
 - **makeTransientXXXXArray(lenght, event)**
 - Build a transient array reinitialized at reset or deselection
 - **XXX** being Boolean, Byte, Short or Object
 - Event is **CLEAR_ON_DESELECT** or **CLEAR_ON_RESET**
- Object sharing
 - Build an object that inherit from **shareable**
 - Provide a reference with **getAppletShareableInterfaceObject (AID, Parameter)**

javacard.framework

- **public abstract class Applet**
 - Super class of all JavaCard Applet, user applets must subclass Applet class.
 - Methods called by the JCRE
 - User applets MUST implements
 - **static void install(byte[] buf, short off, byte len)**
 - Create a new applet instance
 - **abstract void process (APDU apdu)**
 - Called for requesting the applet instance to execute a received command
 - User applets MAY overrides
 - **boolean select()**
 - Called when the applet instance is selected
 - **void deselect()**
 - Called when the applet instance is selected
- Note that:
 - **protected void final register()**
 - Called during installation for registering the newly created instance

En résumé !

Problème : contraintes mémoires et faible ressource de calculs

Solutions :

- un sous-ensemble des caractéristiques du langage Java
- Au niveau du langage
- Au niveau des APIs
- découper la machine virtuelle Java en deux parties
- Une partie interprète embarquée
- Une partie vérification hors carte

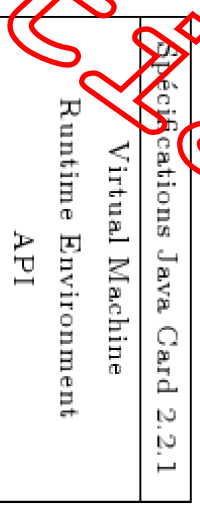
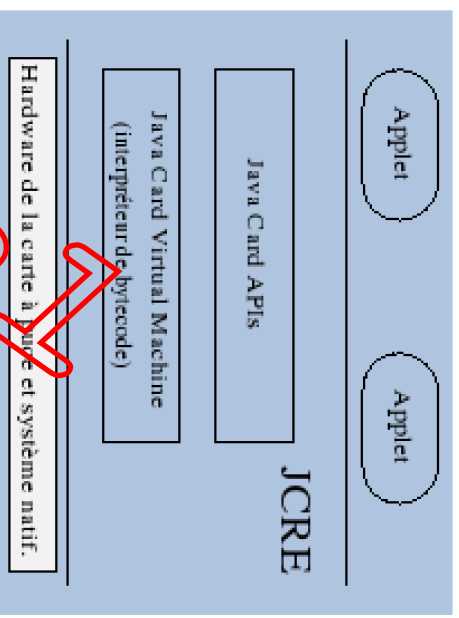
Problème : pas de vérificateur embarqué

Solution :

Fournir des mécanismes sécuritaires avec l'environnement d'exécution
=> **Le firewall !**

Java Card est un sur-ensemble d'un sous-ensemble du langage Java

Nouveau modèle mémoire utilisant la persistance
APIs spécifiques à la carte



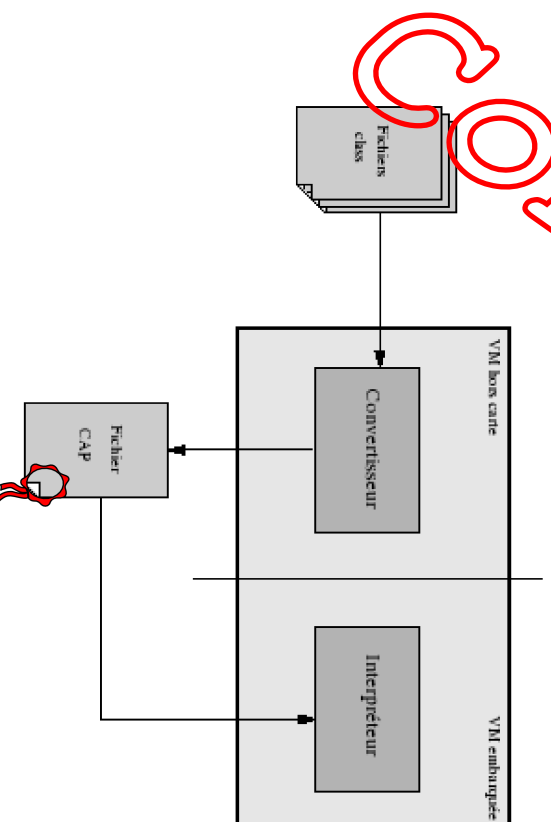
69

En résumé !

Définition de la machine virtuelle Java

Vérifieur de Bytecode
Chargeur dynamique de classes
Interpréteur de Bytecode

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



À cause du découpage de la JVM, la plate-forme est distribuée dans le temps et dans l'espace.

Il faudra des mécanismes pour garantir que ce qui est produit par la partie hors carte n'est modifiée avant d'arriver à la partie embarquée (mécanisme de DAP – Data Authentication Pattern -- et signature) : cf. GlobalPlatform

70

Développement Java Card

71

Construction d'applications carte

- Une application carte
 - Code de la carte (application serveur = applet Java Card)
 - Code dans le terminal (application cliente)
- Construction d'une application Java Card
 - Construction de l'application serveur (applet)
 - Implémentation des services
 - Installation de l'applet dans les cartes
 - Initialisation des services
 - Construction de l'application cliente
 - Invocation des services
- Définir le but de l'application puis spécifier le protocole de communication entre l'application cliente et l'applet Java Card.
 - Commandes à traiter
 - Réponses possibles
 - Cas d'erreurs

72

Construction d'une applet Java Card

- Rôle d'une applet
 - Maintenir son propre état : gestion des champs de l'applet, créer des objets et les référencer pour travailler
 - Répondre à des APDUs de commande (méthode `process ()`) et retourner des APDUs de réponse
- Conception
 - Définir les structures de données nécessaires
 - Créer les objets de base à l'installation, initialiser les champs
 - Implémentation de la méthode `install ()`
 - Définir les APDUs traités par la méthode `process ()`
 - Implémentation d'un analyseur de commandes
 - Utilisation des champs et objets de l'applet
 - Utilisation des APIs
 - Définir les traitements à la sélection et à la désélection

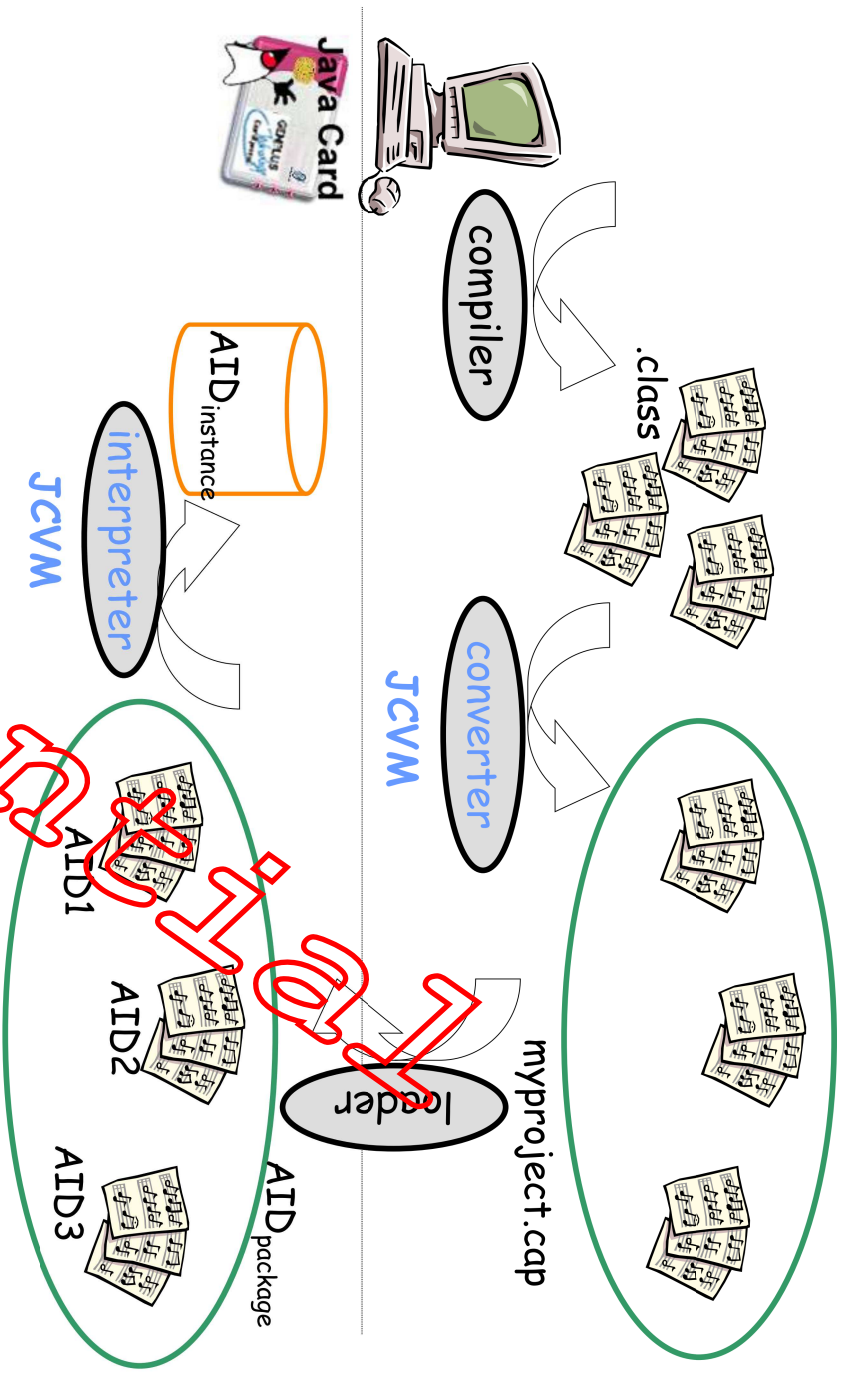
73

Construction d'une applet Java Card

- Fabrication (avec un IDE)
 - Compilation
 - Production de fichier class
 - Conversion
 - Production du fichier CAP
- Initialisation
 - Chargement du fichier CAP
 - Installation de l'applet (instanciation) via un appel à la méthode `install()` en lui donnant un AID
- Utilisation
 - Via l'AID défini pour cette applet

74

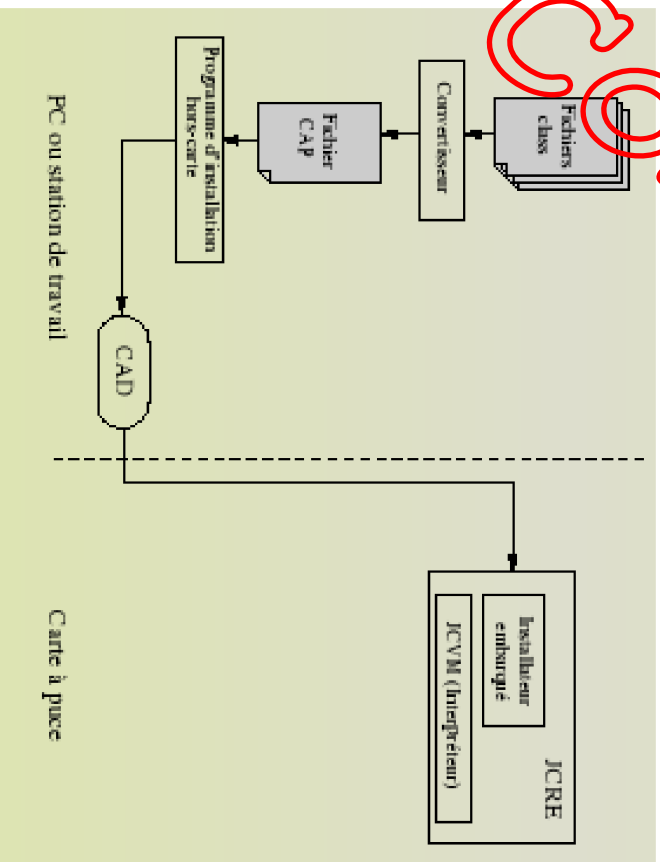
Applet development



75

Installateur Java Card

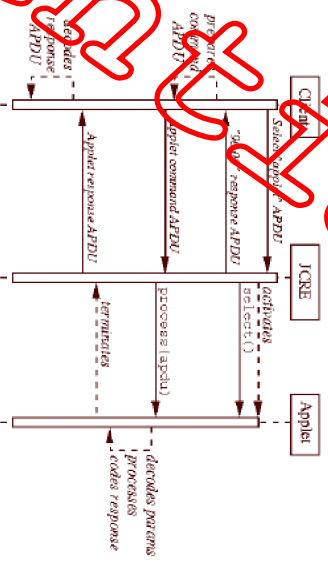
- L'installation se fait grâce à deux entités :
 - l'installateur Java Card et un programme d'installation hors carte
 - Suit quasiment tout le temps le standard GlobalPlatform



76

Construction de l'application cliente

- Rôle de l'application cliente
 - Stimuler l'application serveur et utiliser les résultats pour faire son travail (exemple informer le commerçant que oui l'ordre a bien été accepté)
- Conception
 - Définir les structures de données propre à l'application
 - Choisir son langage, un compilateur, éventuellement un framework développement (.NET, Java, etc.)
 - Choisir un framework de communication : PC/SC, JPC/SC, OCF, javax.smartcardio, etc.
 - Implémentation en utilisant ce framework du protocole de communication avec le serveur (carte)
 - Établissement de la liaison avec le serveur : envoie d'un APDU de sélection non-aid de l'applet
 - Invocation des services de l'applet :
 - Codage et envoie d'APDUs de commande conformes à ceux traités par l'applet
 - Réception et décodage des APDUs de réponse retournés par l'applet
 - Fabrication
 - Avec l'IDE choisi
 - Déploiement
 - sur les machines où le service sera utilisé
 - Utilisation



Résumé construction d'applications

- Développement de l'applet carte
 - Utilisation de l'API Java Card
 - Essentiellement pour traiter des APDU
 - Conversion et installation dans la carte
 - Format du bytecode et procédure d'installation dans la carte non «encore» définis par Java Card
 - Exécution par le JCR
 - Sélection d'applet définie par Java Card
 - Réception et transmission d'APDUs par l'applet définies par Java Card
- Développement de l'application cliente

Les difficultés du développement

- Client et applet communiquent par APDUs
 - Structure de données pauvre (tableau d'octets), peu explicite (pas de typage), et difficile à manipuler
 - Oblige à définir le contenu et la sémantique des messages échangés : décodage et encodage d'APDUs par le client et l'applet
- La spécification des échanges entre client et applet correspond à un protocole
 - Le format des messages APDUs échangés est utilisé comme spécification commune
 - Travail sur un protocole plutôt que sur des fonctionnalités
 - Nécessite une formation carte
- L'applet et le client implémentent un protocole
 - Code «dupliqué» dans les programmes clients et applets
 - Dans la carte : toutes les applets décodent des APDUs
 - Code «sensible»
 - Difficile à programmer : prévoir tous les cas, pas d'outils
 - Source d'erreurs

79

Exemple «Compteur» : Conception

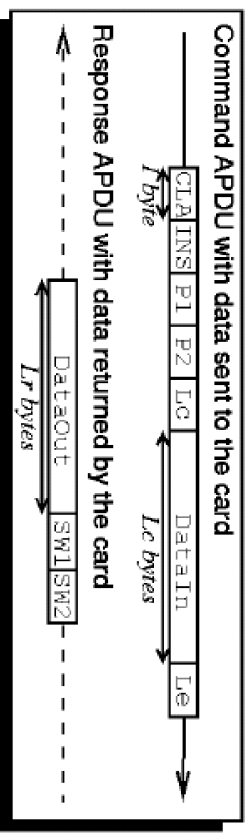
- État de l'applet :
 - Maintient une valeur entière positive ou nulle (pas d'objets)
- Installation : création de l'objet Applet
 - Initialisation de la valeur du compteur
 - Enregistrement auprès du JCRC
- Opérations :
 - Lecture : retourne la valeur du compteur
 - Incrémentation/décrémentation : ajoute/soustrait un montant au compteur et retourne la nouvelle valeur du compteur
- Sélection et désélection :
 - Aucun traitement

80

Exemple «Compteur» : Conception

- APDUs traités par l'applet :

- int lire()
 - Commande : **A0 10** XX XX 00 04
 - Réponse : RV3 RV2 RV1 RV0 **90 00**
- int incrementer(int)
 - Commande : **A0 12** XX XX 04 AM3 AM2 AM1 AM0 **04**
 - Réponse : RV3 RV2 RV1 RV0 **90 00**
- int decremener(int) :
 - Commande : **A0 14** XX XX 04 AM3 AM2 AM1 AM0 **04**
 - Réponse : RV3 RV2 RV1 RV0 **90 00**



Applet «Compteur» : Classe Applet

```
package fr.cryptis.compteur ;
import javacard.framework.* ;

public class Compteur extends Applet {

    private int valeur;

    public Compteur() { valeur = 0; }

    public static void install( byte[] bArray, short bOffset, byte blength)
    { new Compteur().register(); }
```

```
    public void process( APDU apdu ) {
        byte[] buffer = apdu.getBuffer();
        if ( buffer[ISO7816.OFFSET_CLA] != (byte) 0xA0 )
            ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
        switch ( buffer[ISO7816.OFFSET_INS] ) {
            case (byte) 0x10: ... // Opération de lecture
            case (byte) 0x12: ... // Opération d'incrémentatation
            case (byte) 0x14: ... // Opération de décrémentatation
            default: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
        }
    }
}
```

Applet «Compteur» : décrémentation

```
case 0x14: // Opération de décrémentation
{
    // Réception des données
    byte octetsPlus = apdu.setIncomingAndReceive();
    if ( octetsPlus != 4 )
        ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

    int montant = (buffer[ISO7816.OFFSET_CDATAB] << 24) |
        (buffer[ISO7816.OFFSET_CDATAB+1] << 16) |
        (buffer[ISO7816.OFFSET_CDATAB+2] << 8) |
        buffer[ISO7816.OFFSET_CDATAB+3];

    // Traitement
    if ( montant < 0 || valeur-montant < 0 )
        ISOException.throwIt( (short) 0x6910 );
    valeur = valeur - montant;

    // Envoie de la réponse
    buffer[0] = (byte) (valeur >> 24);
    buffer[1] = (byte) (valeur >> 16);
    buffer[2] = (byte) (valeur >> 8);
    buffer[3] = (byte) (valeur);
    apdu.setOutgoingAndSend( (short) 0, (short) 4 );
    return;
}
```

83

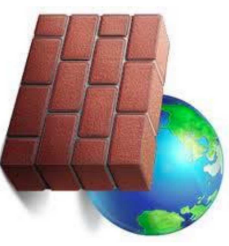
Rappel ♦ Applet Java Card

- Une applet carte est un programme serveur de la Java Card
 - APDU de sélection depuis le terminal
 - Sélection par AID (chaque applet doit avoir un AID unique)
- Une fois installée dans la carte, l'applet est toujours disponible
 - Classe qui hérite de `javacard.framework.Applet`
 - Doit implémenter les méthodes qui interagissent avec le JCRC :
 - `public static void install( byte[] bArray, short boffset, byte blength )`
 - Appelée (une fois) par le JCRC quand l'applet est chargée dans la carte
 - Doit s'enregistrer auprès du JCRC (méthode `register()`)
 - `public boolean select()`
 - Appelée par le JCRC quand un APDU de sélection est reçu et désigne cette applet
 - Rend l'applet active
 - `public abstract void process( APDU apdu )`
 - Appelée par le JCRC quand un APDU de commande est reçu pour cette applet (doit être active)
 - `public void deselect()`
 - Appelée par le JCRC pour désélectionner l'applet courante

84

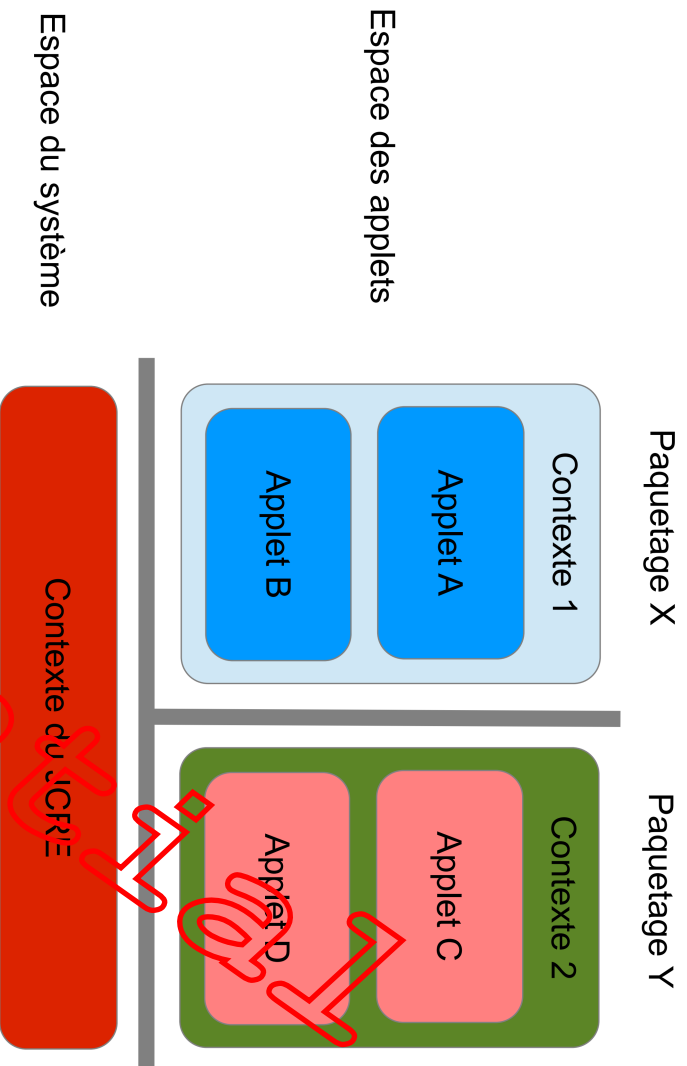
Firewall

Need of a firewall



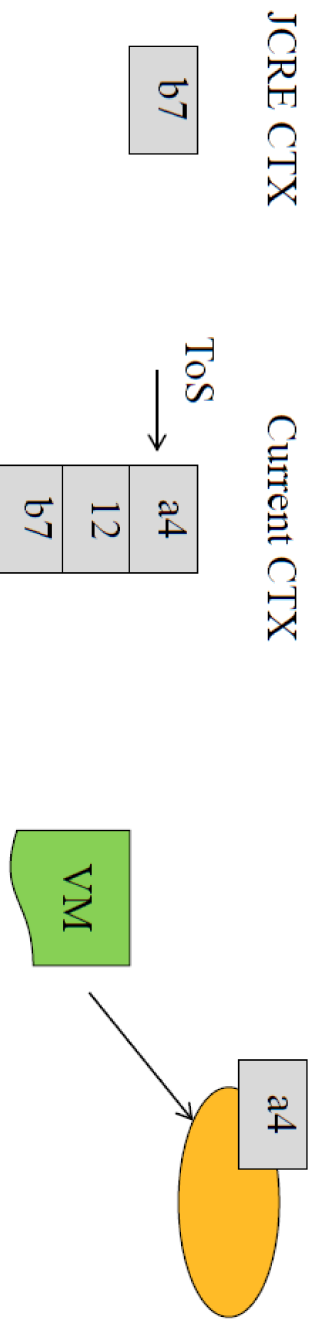
- Add a security level by providing **a protection against developer mistakes and design oversights** that may allowed sensitive data to leaked to another applet,
- Maintain **a strong segregation level between applets designed by different manufacturers,**
- If there is a need, **a mechanism to share information** has been defined in the specification,
- **All the checks are based on the notion of security context.**

La notion de contexte



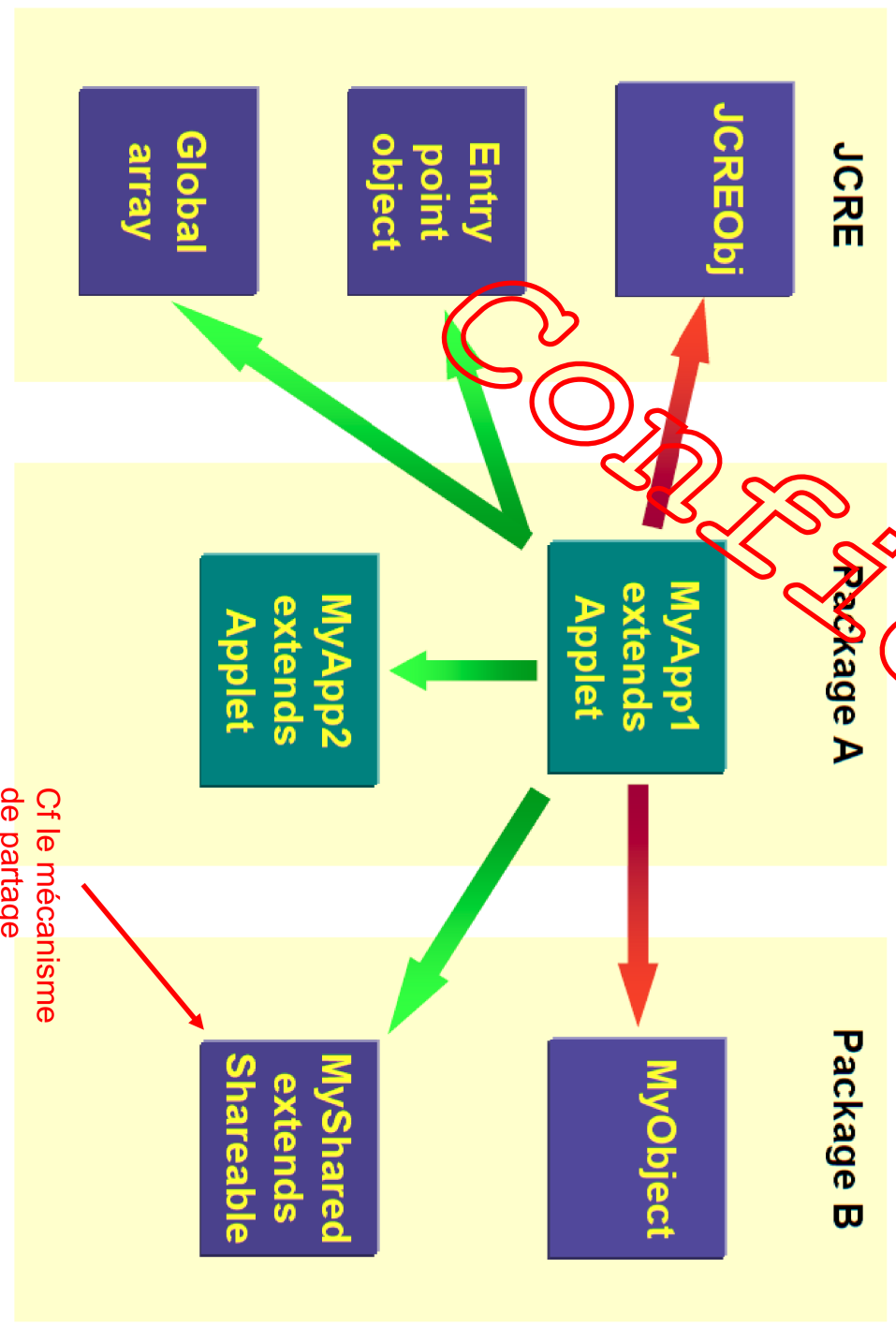
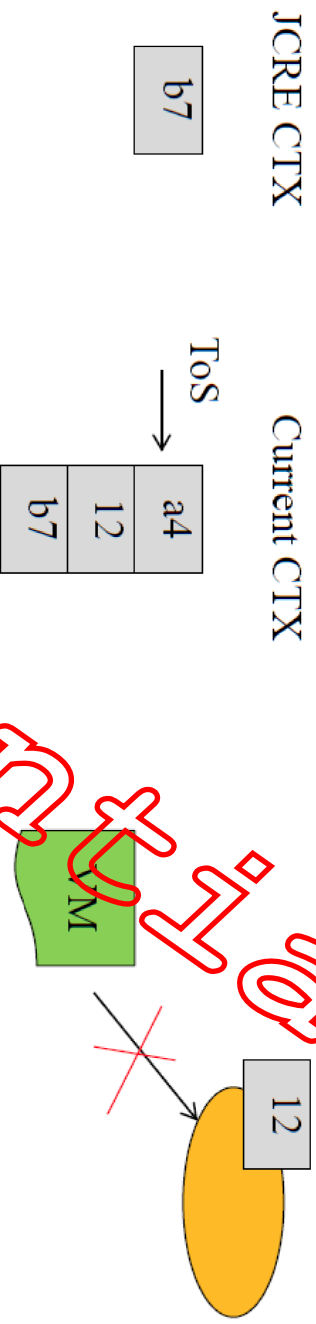
Object ownership

- At any time, there is **only one active context within the virtual machine**: either the JCRE context or an applet's group context.
- When a new object is created, it is assigned an owning context: the **currently active context**.



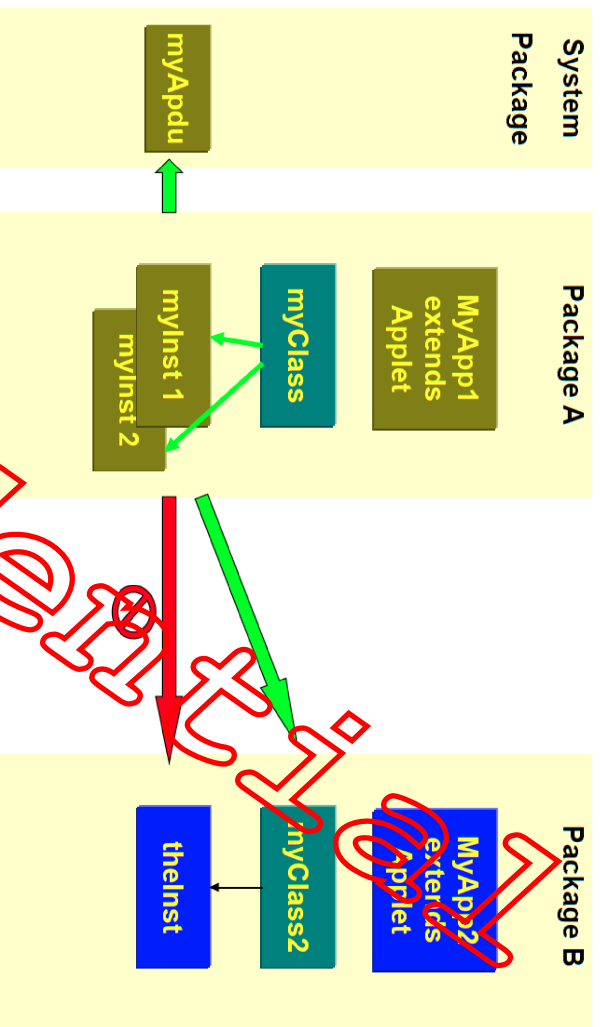
Object ownership

- Every object is owned by an applet instance which is identified by its AID.
- When executing in an instance method of an object the object's owner must be the currently active context,
- If the contexts do not match, the access is denied, and the comparison results in a Security Exception.



Static Fields and Methods

- Only instances of classes i.e. objects, are owned by applets; classes themselves are not.
- Static fields and methods are accessible from any applet context in the defining package (i.e. group context)
 - **Except they are public**: in this case, any applet context can access them.

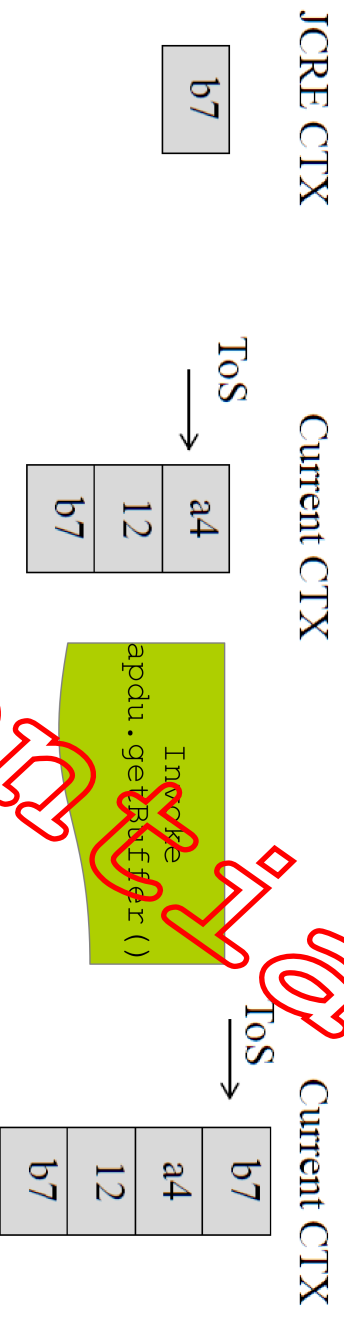


Object Access across Context

- Sharing mechanisms are accomplished by the following means:
 - JCRE privileges
 - JCRE entry point objects
 - Global arrays
 - Shareable interfaces

Context Switch

- When a sharing mechanism is applied, the Java Card virtual machine enables access by performing a context switch.
- Context switches occur
 - only during invocation of and return from instance methods of an object owned by a different context,
 - during exception exits from those methods.



Context Switch

- If Context 1 is the currently active context, and a method m_1 in an object owned by applet A is invoked, no context switch occurs. If method m_1 invokes a method m_2 in an object owned by applet B, again no context switch occurs (in spite of the object "owner" change), and no firewall restrictions apply.
- However, if the method m_2 now calls a method m_0 in an object owned by applet C, firewall restrictions apply and, if access is allowed, a context switch shall occur. Upon return to method m_2 from the method m_0 , the context of applet B is restored.

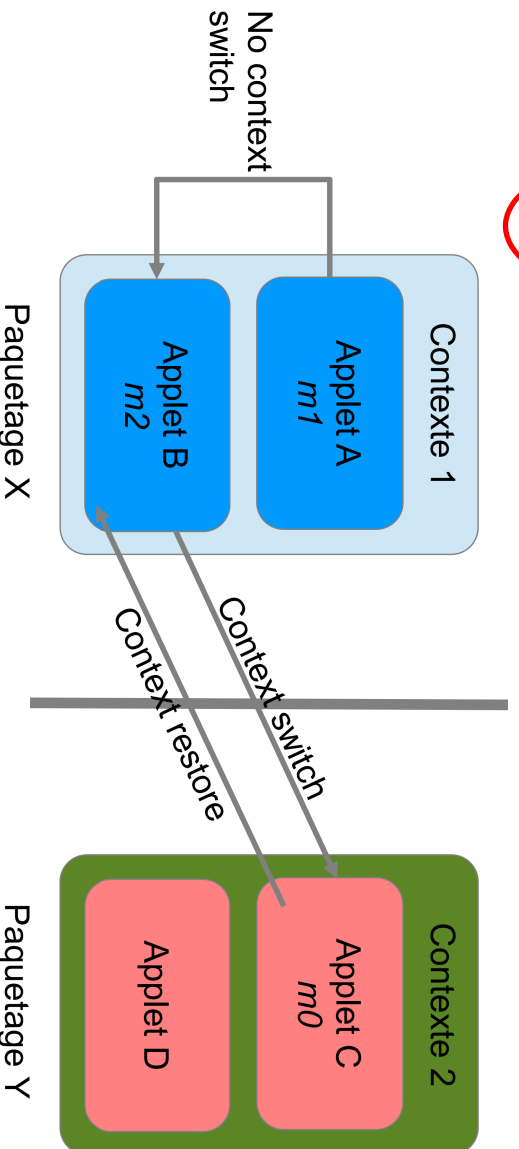


Illustration du fonctionnement du firewall

```
package Z;  
public class Foo {  
    public static Bar bar;  
}  
  
public class Bar {  
    public void test() { ...  
    }  
}
```

```
package X;  
import Z.*;  
  
public class A extends Applet {  
    public void process(APDU apdu) {  
        ...  
        Foo.bar = new Bar();  
        ...  
    }  
}
```

```
package Y;  
import Z.*;  
  
public class C extends Applet {  
    public void process(APDU apdu) {  
        ...  
        Bar bar = Foo.bar;  
        bar.test();  
        ...  
    }  
}
```

Applet isolation

- The firewall provides a strong isolation mechanism between contexts
- Shareable interfaces are a feature in the API to enable applet interaction through a Shareable Interface Object (SIO),
- From the owning context, the SIO is a normal object whose fields and methods can be accessed,
- To any other context, **only the methods defined in the shareable interface are accessible**,
- All other fields and methods are protected by the firewall.

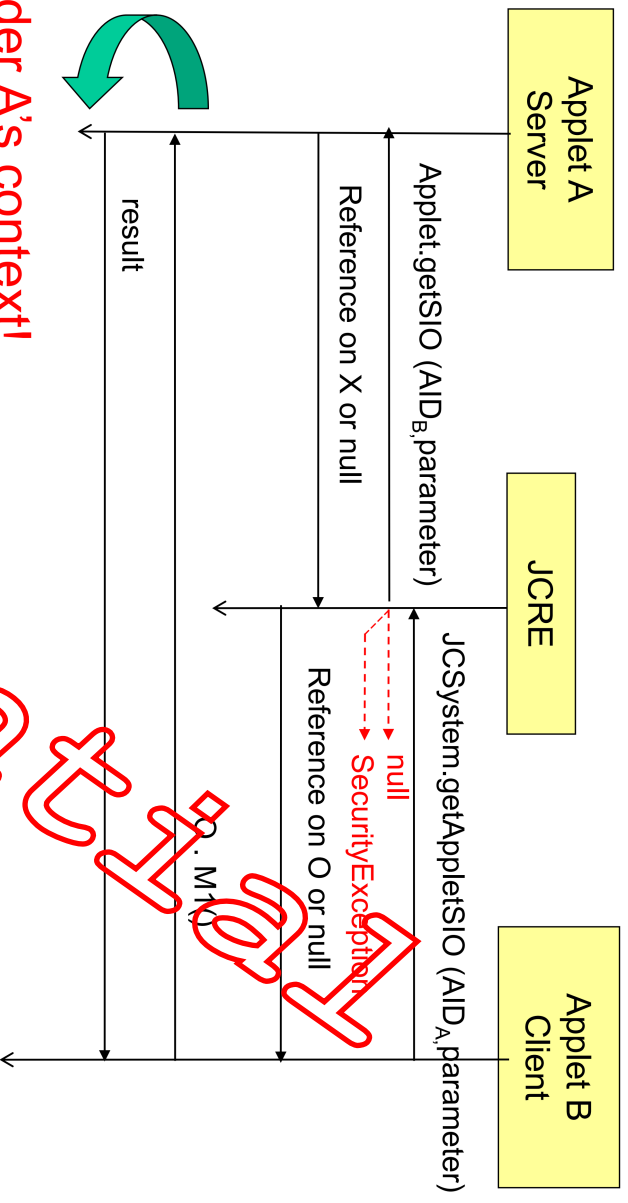
Sharing mechanism

- Methods of an object that implements a shareable interface (SI) can be invoked through the firewall
- **Once shared an object (SIO) cannot be un-shared**
 - In addition it can be given to an untrusted party
- However the caller needs to obtain a reference to this object
 - Applet *A* agree to share with applet *B*
 - Applet *A* declares to implement a shareable interface and implements the services
 - Applet *B* access the services by obtaining an object reference and invoking the service methods.

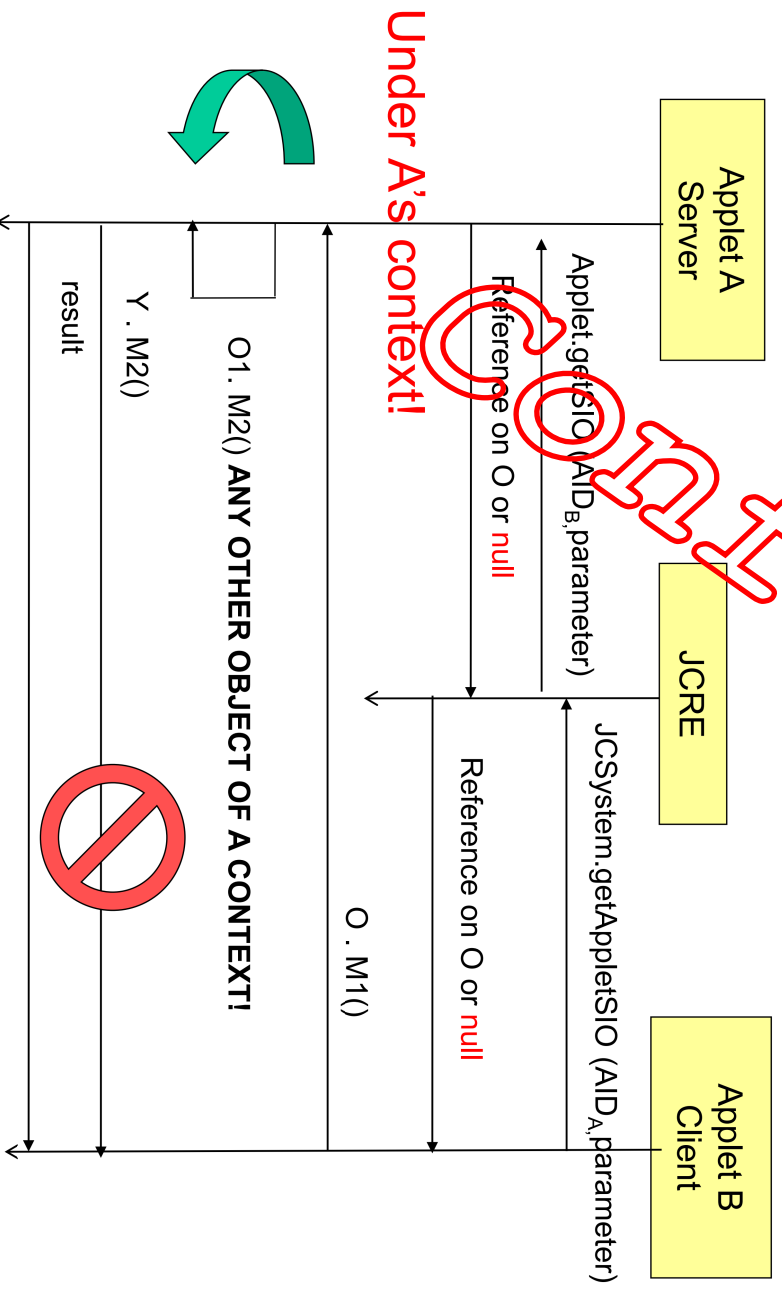
Server Applet A

- *A* defines a Shareable Interface (SI) that extends the interface `javacard.framework.Shareable`
- Defines a class *C* that implements SI with the methods defined in SI,
- Defines other methods protected by the firewall
- *A* creates an object instance of class *C*

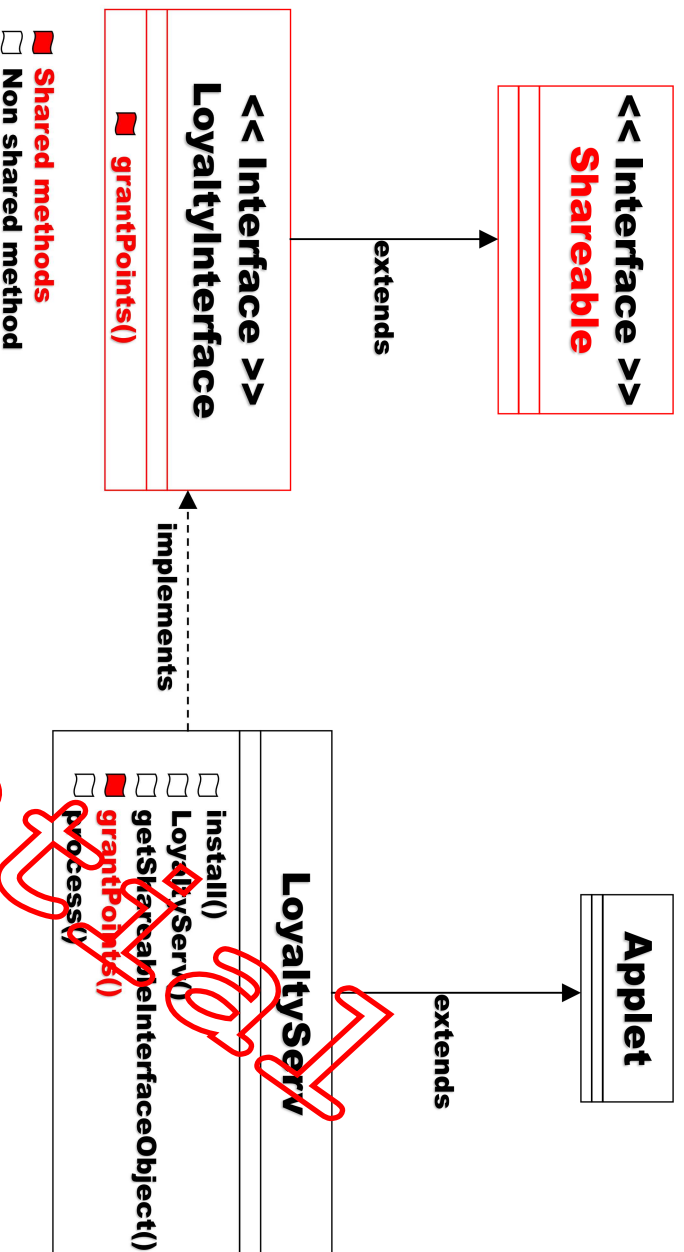
Shareable Interfaces



Shareable Interfaces



Example



Example server side

The server must create a shareable interface that extends `javacard.framework.Shareable`

```
package myPackage
import javacard.framework.Shareable
public interface LoyaltyInterface extends Shareable {
    public void grantPoint (short amount)
}
```

- The server implements the interface

```
package myPackage
public class LoyaltyServ extends Applet implements LoyaltyInterface {
    private short miles;
    public void grantPoint (short amount) {
        miles = (short) (miles+amount);
    }
}
```


Example client side

- The client needs to know the server AID,

```
import myPackage.LoyaltyInterface;

public class LoyaltyClient extends Applet {

    ...

    // request SIO from the server
    LoyaltyInterface sio = (LoyaltyInterface)
        JCSSystem.getAppletShareableInterfaceObject(AIDServer, (byte) 0)
    if (sio==null) {IOException.throwIt((short) 0x6999);}
    sio.grantPoint(theAmount);
```

- The JCRE looks up the server applet and invokes the server `getShareableInterfaceObject` with the client AID as parameter !!!

Example server side

- The server implements the interface

```
package myPackage;

public class LoyaltyServer extends Applet implements LoyaltyInterface{
    private short miles;
    public void grantPoint (short amount) {
        miles = (short) (miles+amount);
    }
    public Shareable getShareableInterfaceObject (AID clientAID,
        byte param){
        if (clientAID.equals(myFriend)){
            return ((Shareable) this)}
        else {return null}
    }
    ...
}
```

Example client side

- The client needs to know the server AID,

```
import myPackage.LoyaltyInterface;

public class LoyaltyClient extends Applet {

    ...

    // request SIO from the server
    LoyaltyInterface sio = (LoyaltyInterface)
        JCSSystem.getAppletShareableInterfaceObject (AIDServer, (byte)
        0);

    if (sio == null) {ISOException.throwIt((short) 0x3999); }
    sio.grantPoint(theAmount);

    ...
}
```

Type confusion with sharing

- Poll *et al.* succeeded with some old cards with type confusion
- They have been able to recover twice the size of the array using applets compiled and loaded separately,
 - Modify the signature and try to perform an array overflow,

```
public interface MyIntextends Shareable {
    void accArray(short[] array);}
```
- with for the client

```
public interface MyIntextends Shareable {
    void accArray(byte[] array);}
```
- The server will treat the byte array as a short array.
- However there is still the firewall... thus the trick is `byte[] giveArray()`;
- They demonstrate that some cards allow it.

- References :

- Wojciech Mostowski and Erik Poll. Malicious Code on Java Card Smartcards: Attacks and Countermeasures. Proceedings, Smart Card Research and Advanced Application Conference CARDIS 2008, Egham, UK, September 2008. LNCS 5189, pages 1-16,

Attention

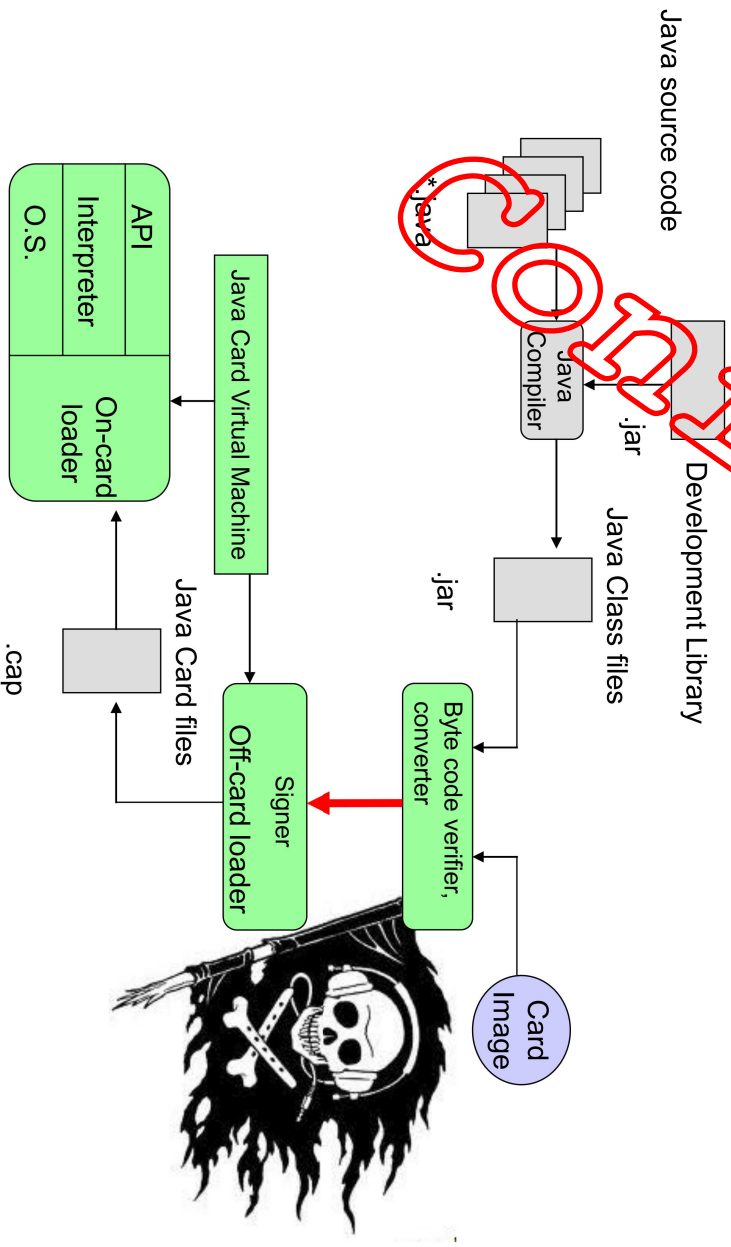
- scan d'applets / usurpation d'identité (spoofing)
 - Possible à une époque à cause de contradiction dans GP et JC.
- Protection :
 - Faire une authentification mutuelle entre les applets
 - Insuffisant s'il y a collusion entre un partenaire et une application qui n'est pas de confiance (ils peuvent s'échanger les clés)
 - Il faut se fier à la plate-forme avec les méthodes suivantes.
 - Utiliser `JCSystem.getAID()` et `JCSystem.getPreviousContextAID()` pour empêcher des accès non souhaités !

III typed Applets

Introduction

- Java Card security
 - Strong typing → byte code verification
- Java is a strongly typed language,
- These properties are verified at the source level by the compiler and at the BC level by the verifier,
- Unable to forge or manipulate references.
- For development cards it is possible to modify the CAP file after the verification

L'architecture Java Card



First step retrieve array address

```
public short getMyAddressstabByte(byte[] tab)
{
    short toto=(byte) 0xAA;
    tab[0] = (byte) 0xFF;
    return toto;
}

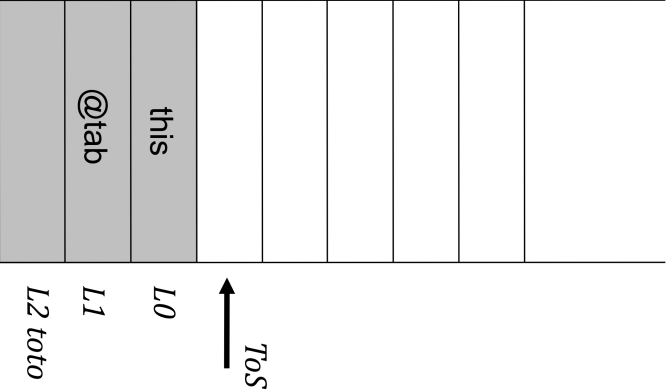
...

public void process (APDU apdu) throws ISOException
{
    ...
    case (byte) 0x29: // provide an array address
        short val = getMyAddressstabByte(tab);
        Util.setShort(apduBuffer, (short) 0, val);
        apdu.setOutgoingAndSend( (short) 0, (short) 2);
        break;
    ...
}

...
```

```
public short getMyAddressstabByte(byte[] tab)
{
    short toto=(byte) 0xAA;
    tab[0] = (byte) 0xFF;
    return toto;
}
```

```
getMyAddressstabByte (byte[] tab)
{
    03 // flags: 0 // max_stack: 3
    21 // nargs: 2 // max_locals: 1
    10 AA bspush -86
    31 sstore_2
    19 aload_1
    03 sconst_0
    02 sconst_m1
    39 sastore
    1E sload_2
    78 sreturn
}
```



```

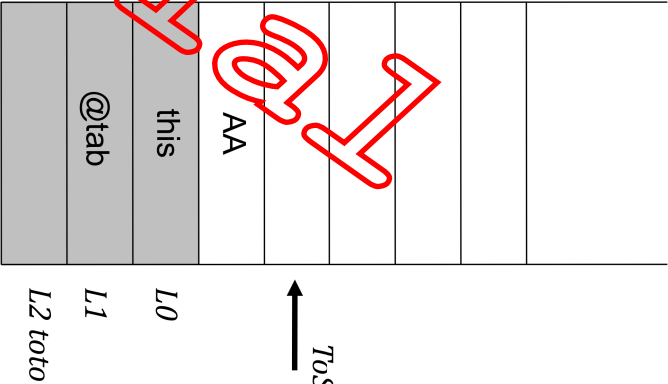
public short getMyAddressTabByte(byte[] tab)
{
    short toto=(byte) 0xAA;
    tab[0] = (byte) 0xFF;
    return toto;
}

```

```

getMyAddressTabByte (byte[] tab)
{
    // flags: 0 // max_stack: 3
    03 // nargs: 2 // max_locals: 1
    21
    10 AA bspush -86
    31 sstore_2
    19 aload_1
    03 sconst_0
    02 sconst_m1
    39 sastore
    1E sload_2
    78 sreturn
}

```



```

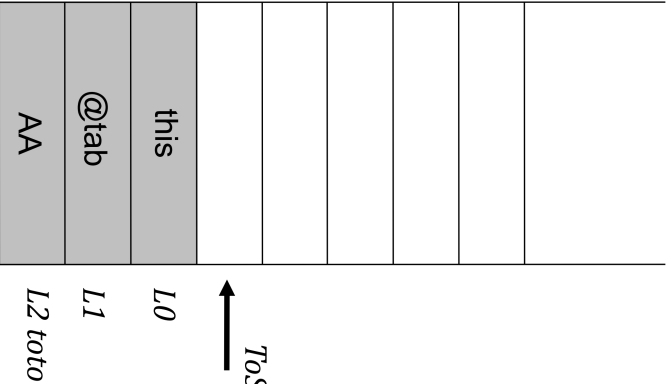
public short getMyAddressTabByte(byte[] tab)
{
    short toto=(byte) 0xAA;
    tab[0] = (byte) 0xFF;
    return toto;
}

```

```

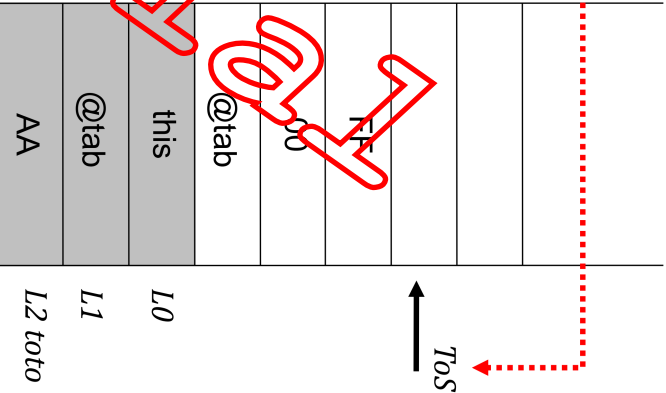
getMyAddressTabByte (byte[] tab)
{
    // flags: 0 // max_stack: 3
    03 // nargs: 2 // max_locals: 1
    21
    10 AA bspush -86
    31 sstore_2
    19 aload_1
    03 sconst_0
    02 sconst_m1
    39 sastore
    1E sload_2
    78 sreturn
}

```



```
short toto=(byte) 0xAA;
tab[0] = (byte) 0xFF;
return toto;
```

```
// flags: 0 // max_stack: 3
// nargs: 2 // max_locals: 1
    bspush -86
    sstore_2
    aload_1
    sconst_0
    sconst_m1
    sastore
    sload_2
    sreturn
```

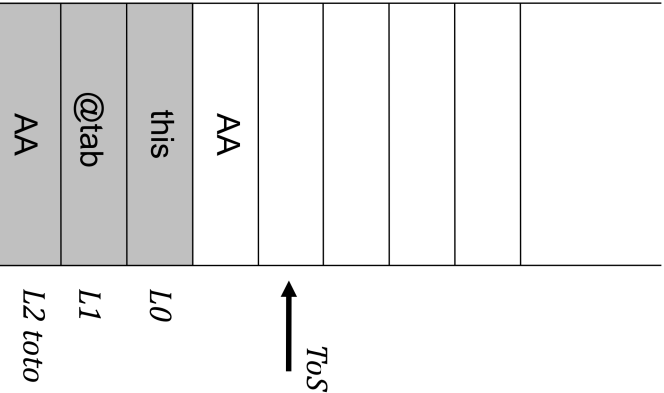


```
short toto=(byte) 0xAA;  
tab[0] = (byte) 0xFF;  
return toto;
```

```

esstabByte (byte[] tab)
// flags: 0 // max_stack: 3
// nargs: 2 // max_locals: 1
    bspush -86
    sstore 2
    aload_1
    sconst_0
    sconst_m1
    astore
    sload_2
return

```



```

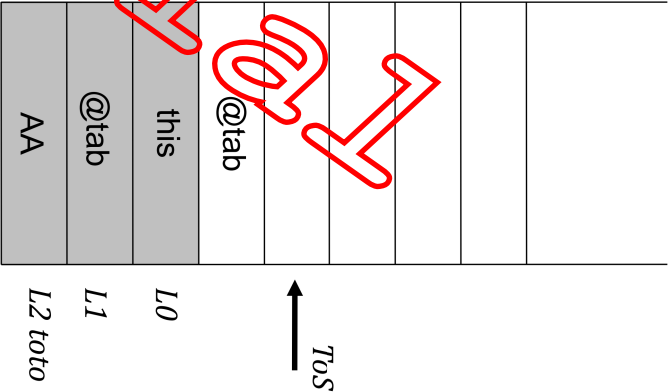
public short getMyAddressTabByte(byte[] tab)
{
    short toto=(byte) 0xAA;
    tab[0] = (byte) 0xFF;
    return toto;
}

```

```

getMyAddressTabByte (byte[] tab)
{
    // flags:  0 // max_stack:  3
    03          // nargs:  2 // max_locals:  1
    21          bspush -86
    10  AA
    31          sstore_2
    19          aload_1
    00          nop
    00          nop
    00          nop
    00          nop
    00          nop
    78          sreturn
}

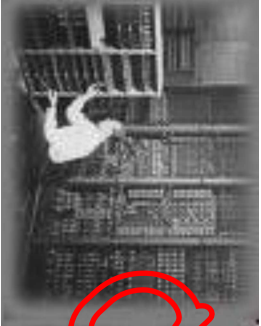
```



Usage

Array address?

80 29 00 00 00



94 4C 90 00



The address is 0x944C

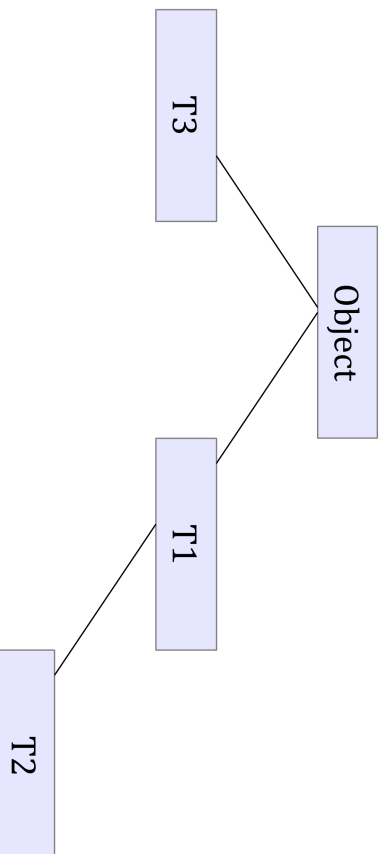
- We succeeded to retrieve a reference in the card memory.
- **This should be impossible if a verifier was embedded**
- This is only the first part of an attack trying execute the byte array!

The Oberthur attack: Principle

- The Oberthur attack is based on type confusion,
- The applet loaded in the card is correct i.e. cannot be rejected by a byte code verifier,
- **The idea is to bypass** the run time check made if the code impose a type conversion,
 - Inject the energy during the check,
 - It is a transient fault,
 - The result can be the dump of the memory.

Java Type conversion

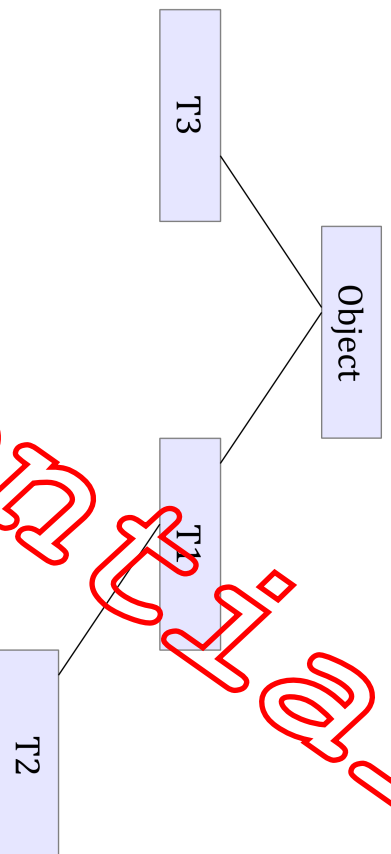
- Java imposes a type hierarchy:



Java Type conversion

- Java imposes a type hierarchy:
- Polymorphism allows type conversion checked at run time

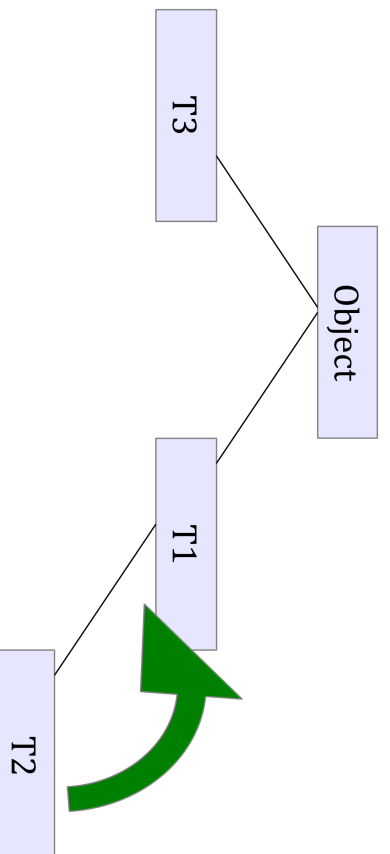
```
T2 t2;  
T1 t1 = (T1) t2;  
↔  
aload t2  
checkcast T1  
astore t1
```



Java Type conversion

- Java imposes a type hierarchy:
- Polymorphism allows type conversion checked at run time

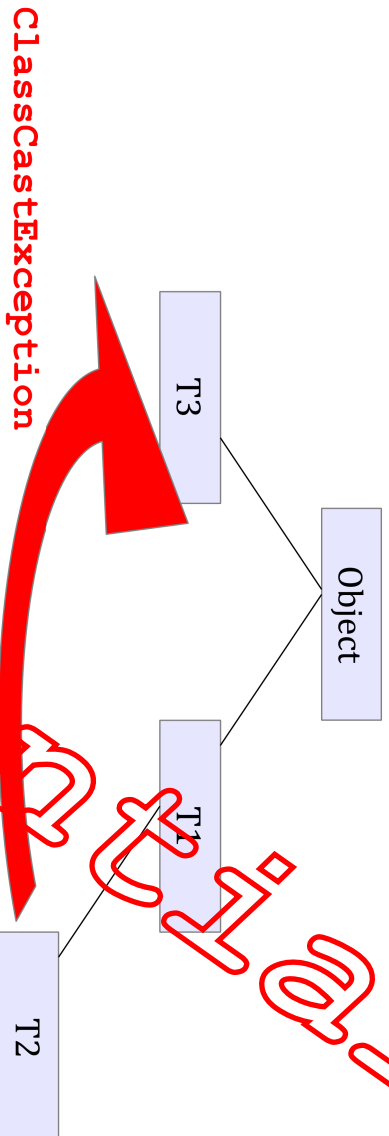
```
T2 t2;  
T1 t1 = (T1) t2;  
↔  
aload t2  
checkcast T1  
astore t1
```



Java Type conversion

- Java imposes a type hierarchy:
- Polymorphism allows type conversion checked at run time

```
T2 t2;  
T3 t3 = (T3) t2;  ⇔  aload t2  
                   checkcast T3  
                   astore t1
```



The following class

- Define the class `A` with one field of type short,

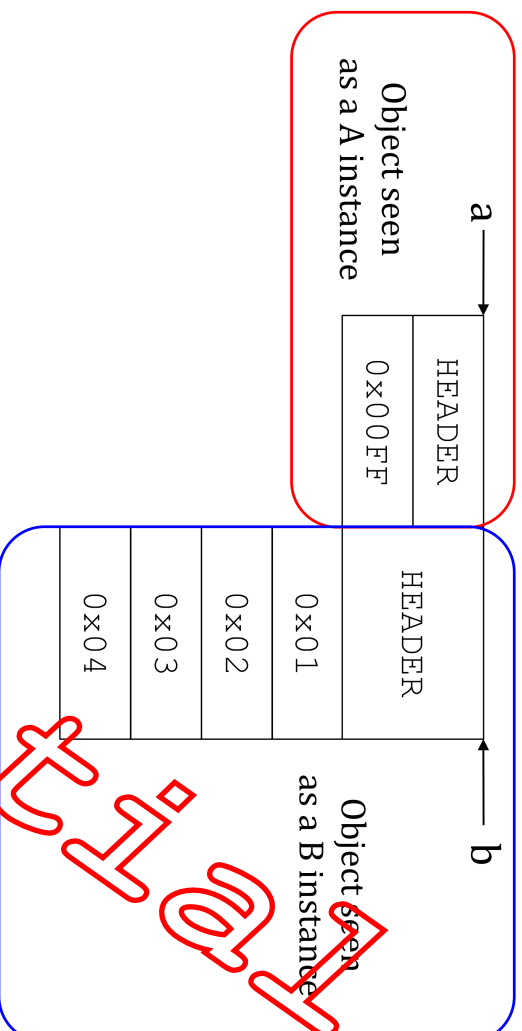
```
public class A {short theSize = 0x00FF;}  
}
```

- In the application defines instances,

```
...  
A a = new A();  
byte[] b = new byte[10]; b[0] = 1; b[1]=2;...  
...  
a = (A) ((Object)b); // a & b point on the same object  
a.theSize = 0xFFFF; // increases the size of the []  
// read and write your array...
```

The Hazardous Type Confusion

- Confusion between a and b (header compatible)

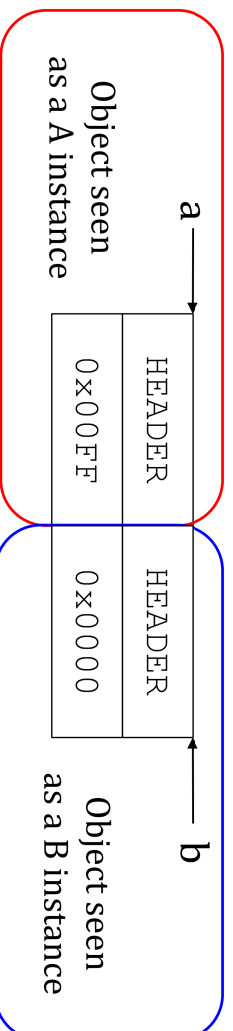


The Hazardous Type Confusion

- Confusion between a and b (incompatible)

```
public class A {short theSize = 0x00FF;}  
public class B {C c = null;}
```

Warning the firewall will play its role!



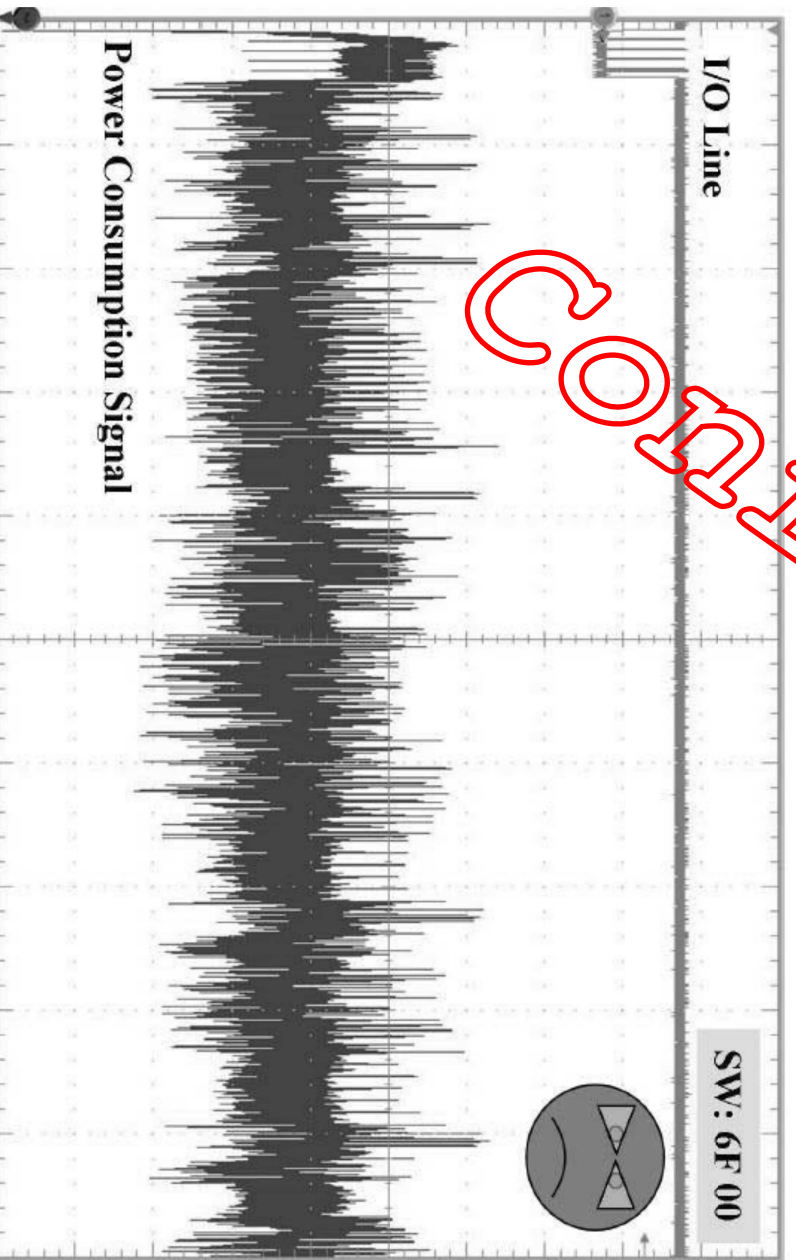
All what you need is... type confusion

- To force the type confusion

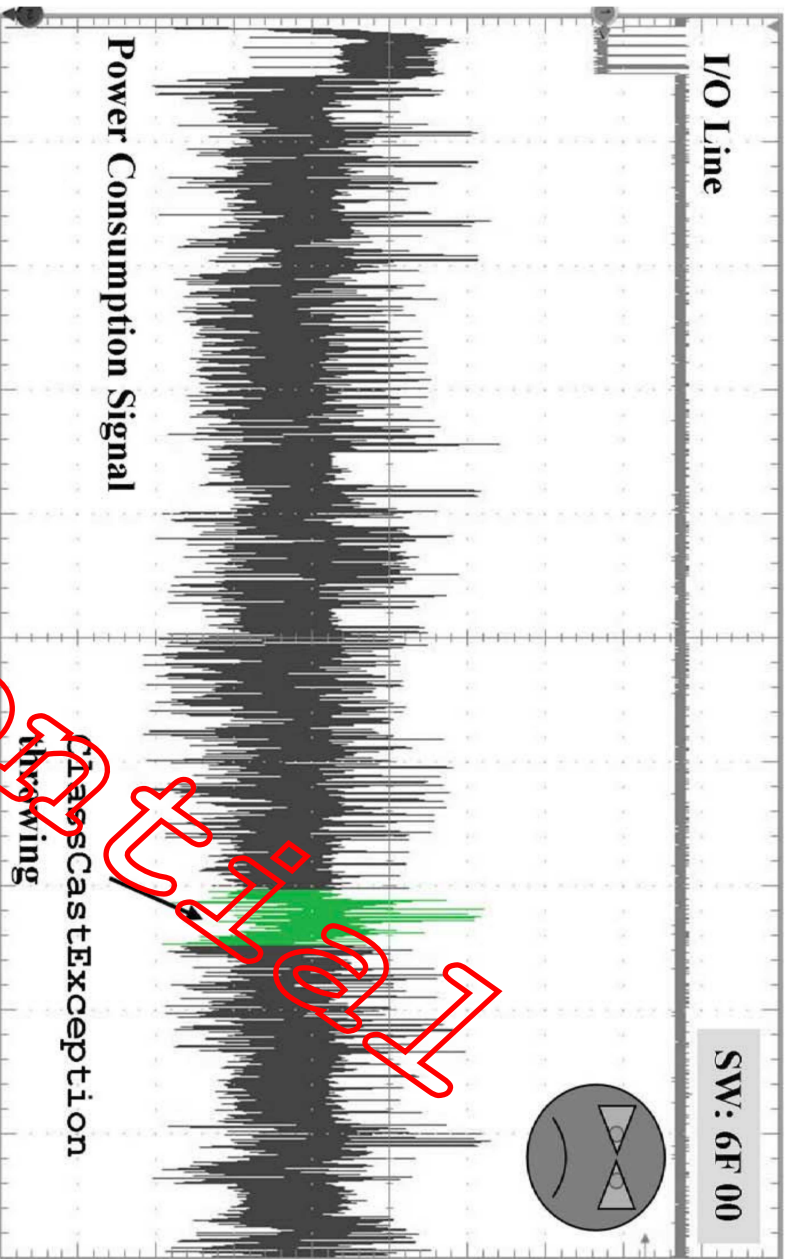
A a = (A) b;
↔
aload b
checkcast A
astore a

- The BCV can check the applet it is a legal one,
- During runtime the checkcast instruction will generate an exception `ClassCastException`

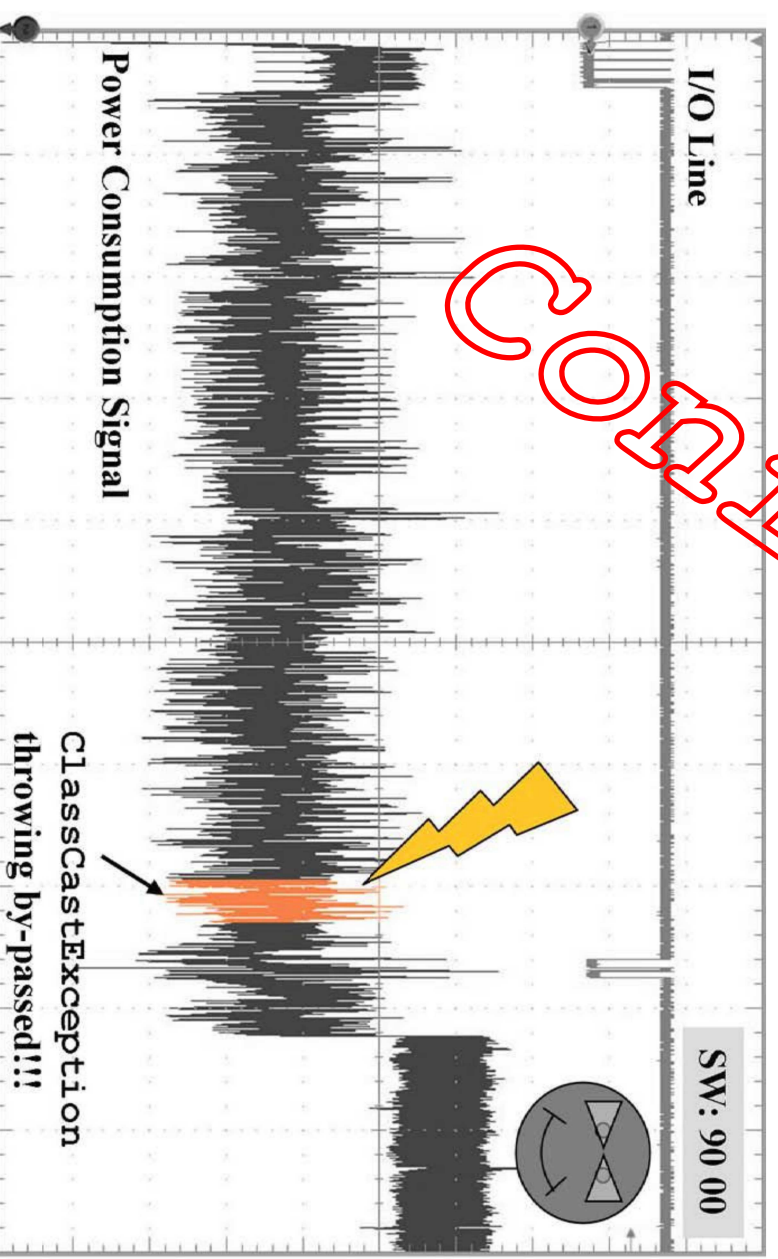
Power analysis of the checkcast



Power analysis of the checkcast



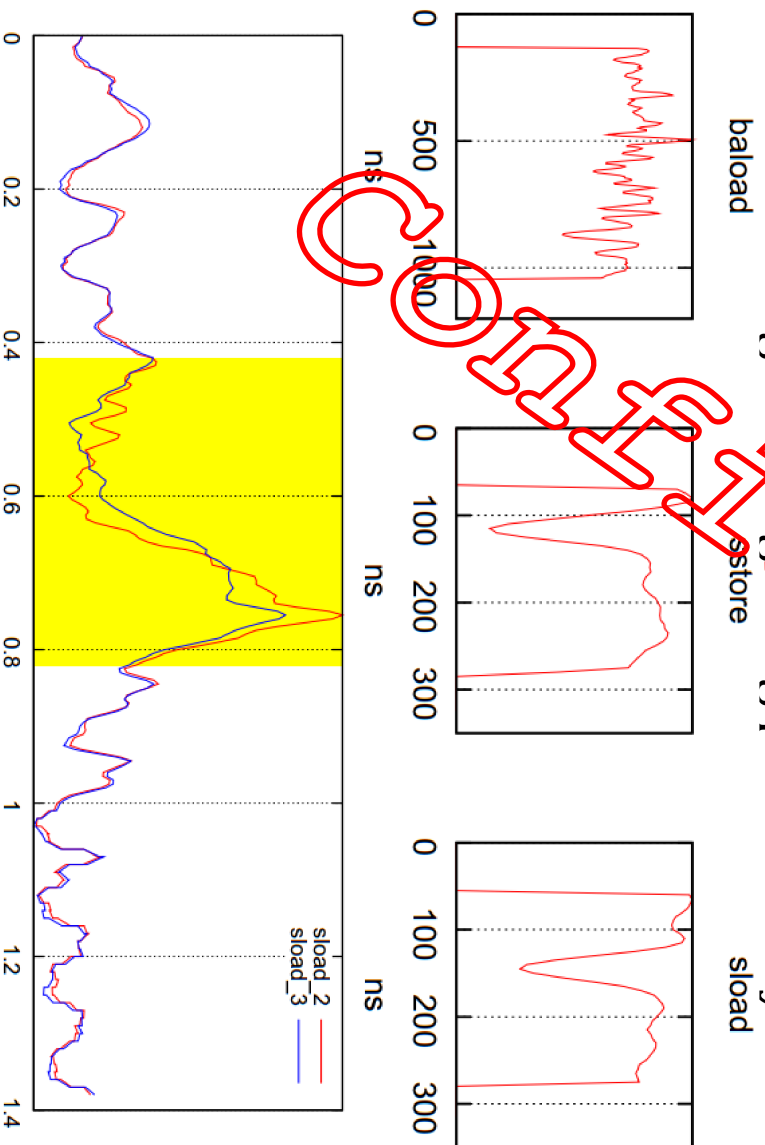
Practical Laser Fault Injection



Conclusion

- Il est possible de faire des attaques même si la carte dispose d'un vérifieur embarqué !
- Vous pouvez ensuite essayer de lire la mémoire ! ☺

Reverse engineering using power analysis



- Référence

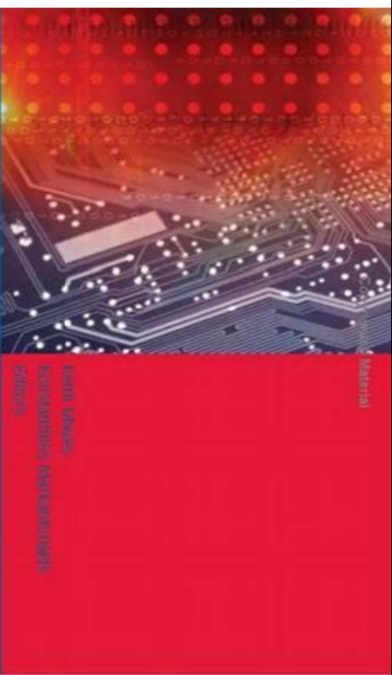
- Dennis Vermoen, Marc Wittenman and Georgi N. Gaydadjiev
Reverse engineering Java Card applets using power analysis
Workshop in Information Security Theory and Practice 2007

Conclusion Java Card

- Malgré toute l'attention portée à la carte à puce et à Java Card, il y a encore tant à faire (si on considère un périphérique ouvert)



Smart Cards, Tokens, Security and Applications
Mayes, Keith; Markantonakis, Konstantinos (Eds.)
2008, XXXVIII, 392 p. 128 illus., Hardcover
ISBN: 978-0-387-72197-2



Smart Cards, Tokens, Security and Applications

 Springer

Copyrighted Material

Secure Smart Embedded Devices, Platforms and Applications
Konstantinos Markantonakis, Keith Mayes
2014, XLI, 568 p. 135 illus., Hardcover
ISBN: 978-1-4614-7914-7

Secure Smart Embedded Devices, Platforms and Applications

Konstantinos Markantonakis
Keith Mayes Editors

 Springer

Les cartes à puce
BOUZEFRANE Samia, PARADINAS Pierre
Ouvrage 326 p. 15.6x23.4 cm Relié



Références

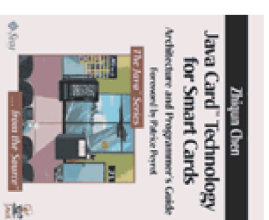
Site de la technologie Java Card chez Sun Microsystems :
<http://java.sun.com/javacard/>

Technology for Smart Cards: Architecture and Programmer's Guide
Zhiqun Chen
18 Septembre 2000

Thèse de Damien Sauveiron
<http://damien.sauveiron.fr/>

Cours du CNAM (Pierre Paradinas) :
<http://deptinfo.cnam.fr/~paradinas/cours/>

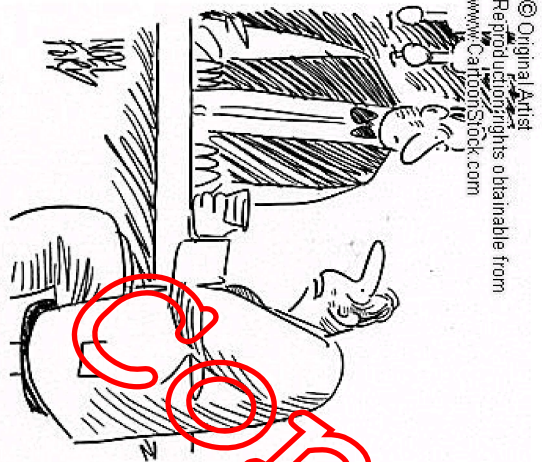
Cours de Didier Donsez :
<http://www-adele.imag.fr/~donsez/cours/#smartcard>



Disclaimer

Ce cours a été réalisé grâce :
aux transparents que j'ai pu présenter dans différents séminaires
aux transparents de cours de Samia Bouzefrane, Didier Donsez, Gilles Grimaud, Sylvain Lecomte, Pierre Paradinas
et Jean-Jacques Vandewalle.
à quelques illustrations trouvées sur Internet.

Questions ?



**Si votre carte ne vous comprend pas,
la comprenez vous mieux ?**

