UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Java Card

Développement et Communication avec Cartes à Puce

Objectifs de la Formation

Compétences visées

- Comprendre l'architecture Java Card et le middleware
- Maîtriser le protocole APDU et l'interface PC/SC
- Développer des applets Java Card sécurisées
- Utiliser PKCS#11 et java.smartcardio
- Implémenter les bonnes pratiques de sécurité

Approche pédagogique

- Théorie avec démonstrations pratiques
- Exercices guidés sur carte réelle/simulée
- Cas d'usage réels (authentification, cryptographie)
- Best practices et sécurité

Programme de la Formation

- Architecture Java Card
- Middleware (CSP, Minidriver, IAS/ECC)
- Structure carte et PKCS#15
- Protocole APDU
- Commandes et réponses
- Codes d'erreur
- Interface PC/SC
- Outils de trace et débogage
- API java.smartcardio
- Connexions et sessions
- Canaux logiques
- PKCS#11 en détail
- Développement applets
- Gestion mémoire
- Cryptographie sur carte
- Rôles (SO, User)
- Secure Messaging
- Bonnes pratiques sécurité
- Projet complet
- Tests et déploiement
- Troubleshooting

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Architecture Java Card

Qu'est-ce que Java Card ?

Définition

Java Card est une technologie permettant d'exécuter des applications Java (applets) sur des cartes à puce avec ressources limitées.

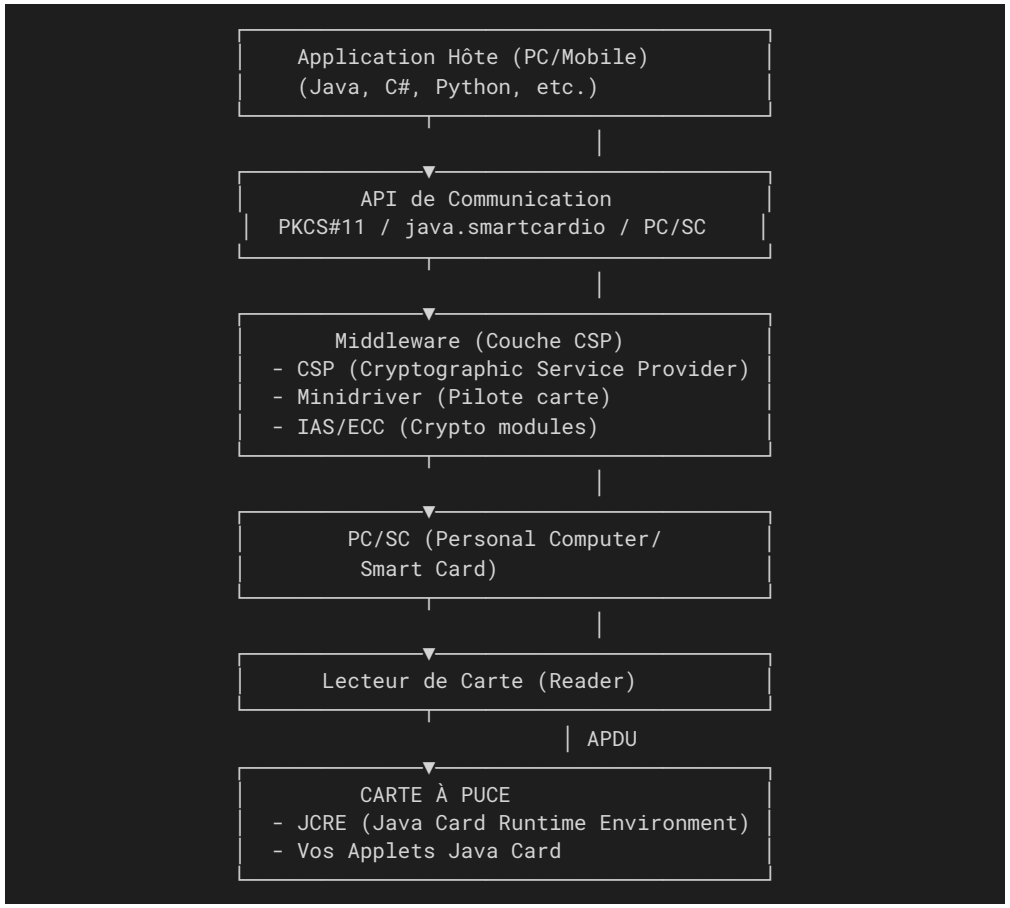
Caractéristiques

- Subset de Java (pas de String, float, garbage collector complet)
- JCRE (Java Card Runtime Environment)
- Sécurité renforcée (isolation, cryptographie)
- Persistance EEPROM
- Standard ISO 7816

Cas d'usage

- Cartes bancaires (EMV)
- SIM cards
- Cartes d'identité électroniques
- Badges d'accès sécurisés
- Signature électronique

Architecture Globale



La Couche Middleware - CSP

CSP (Cryptographic Service Provider)

Rôle

- Interface entre applications et carte
- Fournit services cryptographiques standardisés
- Abstraction des opérations crypto

Fonctions

- Signature électronique
- Chiffrement/Déchiffrement
- Génération de clés
- Hachage

Exemples

- Windows Crypto API (CAPI)
- CNG (Cryptography Next Generation)
- OpenSSL

La Couche Middleware - Minidriver

Minidriver

Rôle

- Pilote léger spécifique à chaque modèle de carte
- Traduit commandes génériques → commandes spécifiques fabricant
- Gère particularités matérielles

Architecture

- Se situe entre CSP et PC/SC
- Un minidriver par type de carte
- Implémentation des commandes APDU spécifiques

Exemples

- Minidriver Gemalto
- Minidriver Oberthur
- Minidriver G&D

La Couche Middleware - IAS/ECC

IAS (Identification, Authentication, Signature)

Modules fonctionnels :

- **Identification** : Vérification identité
- **Authentication** : Prouver identité (PIN, biométrie)
- **Signature** : Signer documents/transactions

ECC (Elliptic Curve Cryptography)

Avantages

- Clés plus courtes pour même sécurité
- 256 bits ECC ≈ 3072 bits RSA
- Idéal pour cartes (mémoire limitée)
- Performances accrues

Algorithmes

- ECDSA (signature)
- ECDH (échange de clés)
- Courbes : P-256, P-384, secp256k1

Structure Matérielle de la Carte

Composants

Microprocesseur

- CPU 8/16/32 bits
- Fréquence : 1-5 MHz

ROM (Read-Only Memory)

- Système d'exploitation
- JCRE
- Taille : 64-256 KB
- Masqué en fabrication

EEPROM (Persistent)

- Applets Java Card
- Données utilisateur
- Taille : 32-512 KB
- Réinscriptible (~500k cycles)

⚠️ **Contraintes importantes**

- Mémoire très limitée

RAM (Volatile)

- Variables temporaires
- Taille : 4-16 KB
- Effacée à la coupure

Coprocesseur Crypto

- Accélération RSA/ECC/AES/DES
- Essentiel pour performances

Interface

- Contact (ISO 7816)
- Sans-contact (NFC/RFID)

Organisation ISO 7816-4

Système de fichiers hiérarchique

```
MF (Master File) - 3F00
│
├── DF (Dedicated File) - Application 1
│   │   AID: A0000000123456
│   ├── EF (Elementary File) - Données 1
│   │   FileID: 0001
│   ├── EF - Clés privées
│   │   FileID: 0002
│   └── EF - Certificats
│       FileID: 0003
│
├── DF - Application 2 (Votre Applet)
│   │   AID: A0000000ABCDEF
│   ├── EF - PIN
│   ├── EF - Compteurs
│   └── EF - Logs
│
└── EF - Fichier global ATR
    FileID: 2F01
```

Organisation ISO 7816-4

Concepts clés

- **MF** : Racine, toujours FileID = 3F00
- **DF** : Répertoire/Application, identifié par AID (5-16 octets)
- **EF** : Fichier de données
- **AID** : Application IDentifier (unique)

Types d'Elementary Files

1. Transparent

- Tableau d'octets brut
- Accès par offset
- Usage : Données binaires quelconques

2. Linear Fixed

- Enregistrements de taille fixe
- Numérotés 1, 2, 3...
- Usage : Logs, historique

3. Linear Variable

- Enregistrements de taille variable
- Chaque enregistrement a sa longueur
- Usage : Messages, données hétérogènes

4. Cyclic

- Buffer circulaire (FIFO)
- Écrase les plus anciennes entrées
- Usage : Logs rotatifs

PKCS#15 - Organisation des Objets

Standard pour organiser objets cryptographiques sur carte

```
Carte PKCS#15
│
├── MF (3F00)
│   └── EF(DIR) - Liste des applications
│
├── DF(PKCS#15) - Application PKCS#15
│   ├── EF(ODF) - Object Directory File
│   │   ├── Pointeur → PrKDF
│   │   ├── Pointeur → PuKDF
│   │   ├── Pointeur → CDF
│   │   ├── Pointeur → DODF
│   │   └── Pointeur → AODF
│   │
│   ├── EF(PrKDF) - Private Key Directory File
│   ├── EF(PuKDF) - Public Key Directory File
│   ├── EF(CDF) - Certificate Directory File
│   ├── EF(DODF) - Data Object Directory File
│   ├── EF(AODF) - Authentication Object Directory File
│   │
│   ├── EF - Clé privée 1 (données réelles)
│   ├── EF - Certificat 1 (données réelles)
│   └── EF - PIN (données réelles)
```

Objets PKCS#15

Clés Privées (PrKDF)

- RSA (1024, 2048, 4096 bits)
- ECC (256, 384, 521 bits)
- Jamais exportables
- Protégées par PIN

Clés Publiques (PuKDF)

- Correspondent aux clés privées
- Exportables
- Pour vérification de signature

Certificats (CDF)

- Format X.509
- Lient clé publique à identité
- Peuvent être compressés

PINs/Auth (AODF)

- User PIN (usage normal)
- SO-PIN (Security Officer)
- PUK (PIN Unblock Key)

Données (DODF)

- Données applicatives
- Configuration

Exercice 1 : Architecture PKCS#15

Énoncé

Vous devez créer une applet Java Card qui stocke :

- 2 clés RSA 2048 bits
- 2 certificats X.509
- 1 User PIN
- 1 SO-PIN

Questions

1. Dessinez l'arborescence PKCS#15 complète
2. Attribuez les FileIDs (convention : 0x4xxx pour directories, 0x3xxx pour données)
3. Estimez l'espace EEPROM nécessaire
4. Quels sont les risques si la RAM est insuffisante ?

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

PKCS#11 et APDU

PKCS#11 - Introduction

Public-Key Cryptography Standards #11

Qu'est-ce que PKCS#11 ?

- Standard **Cryptoki** (Cryptographic Token Interface)
- Défini par RSA Labs, maintenu par OASIS
- API uniforme pour tokens cryptographiques
- Langage C, mais wrappers disponibles

Objectif

Interface standardisée pour :

- Cartes à puce
- HSM (Hardware Security Module)
- Clés USB sécurisées
- Tokens cryptographiques

URL officielle

https://docs.oasis-open.org/pkcs11/

Architecture PKCS#11

Architecture PKCS#11

Concepts clés

- **Slot** : Emplacement physique (lecteur)
- **Token** : Carte insérée
- **Session** : Connexion logique
- **Objets** : Clés, certificats, données

Versions PKCS#11

Évolution du standard

Version	Année	Nouveautés
v2.01	1997	Version initiale largement déployée
v2.20	2004	Améliorations, encore très utilisé
v2.30	2009	Support ECC
v2.40	2015	Nouvelles courbes, AES-GCM
v3.0	2020	Modernisation, nouveaux algos
v3.1	2023	Version actuelle

Versions PKCS#11

Comment identifier la version ?

```
CK_INFO info;
C_GetInfo(&info);
printf("PKCS#11 v%d.%d\n",
    info.cryptokiVersion.major,
    info.cryptokiVersion.minor);
```

Codes d'Erreur PKCS#11

Préfixe CKR_ (Cryptoki Return value)

Succès

- `CKR_OK` (0x00000000) : Succès

Erreurs Courantes

- `CKR_GENERAL_ERROR` : Erreur générale
- `CKR_FUNCTION_FAILED` : Fonction échouée
- `CKR_ARGUMENTS_BAD` : Arguments invalides
- `CKR_DEVICE_ERROR` : Erreur matérielle
- `CKR_DEVICE_MEMORY` : Mémoire carte insuffisante
- `CKR_DEVICE_REMOVED` : Carte retirée

Authentification

- `CKR_PIN_INCORRECT` : PIN incorrect
- `CKR_PIN_LOCKED` : PIN bloqué
- `CKR_USER_NOT_LOGGED_IN` : Non authentifié
- `CKR_USER_PIN_NOT_INITIALIZED` : PIN non init

Session

- `CKR_SESSION_CLOSED` : Session fermée
- `CKR_TOKEN_NOT_PRESENT` : Token absent
- `CKR_TOKEN_NOT_RECOGNIZED` : Token non reconnu

Documentation

Tous les codes : https://docs.oasis-open.org/pkcs11/pkcs11-base/v3.1/

APDU - Introduction

Application Protocol Data Unit

Définition

Protocole de communication entre lecteur (host) et carte selon **ISO 7816-4**

Deux types d'APDU

1. **Command APDU** : Host → Carte
2. **Response APDU** : Carte → Host

Analogie

- APDU = Langage parlé par la carte
- Comme HTTP pour le web
- Requête/Réponse

Importance

Tout passe par APDU :

- Sélection d'applet; Vérification PIN
- Lecture/Écriture données, Opérations cryptographiques

Structure Command APDU

CLA (Class) - Identifie la classe de commande

- `0x00` : ISO 7816-4 standard
- `0x80-0xFF` : Propriétaire / Applet
- `0x0C` : Secure Messaging

INS (Instruction) - Code de l'instruction

- `0xA4` : SELECT
- `0x20` : VERIFY (PIN)
- `0xB0` : READ BINARY
- `0xD6` : UPDATE BINARY

Structure Command APDU

P1, P2 - Paramètres spécifiques (2 octets)
Lc - Longueur des données (1 octet)
Data - Données de la commande (Lc octets)
Le - Longueur réponse attendue (1 octet)

Les 4 Cases d'APDU

Case 1 : Pas de données

```
CLA INS P1 P2
Exemple: 00 A4 00 00  (SELECT sans données)
```

Case 2 : Réponse attendue

```
CLA INS P1 P2 Le
Exemple: 00 C0 00 00 10  (GET RESPONSE 16 octets)
```

Case 3 : Données envoyées

```
CLA INS P1 P2 Lc Data
Exemple: 00 20 00 01 04 31 32 33 34  (VERIFY PIN "1234")
```

Case 4 : Données envoyées ET réponse

```
CLA INS P1 P2 Lc Data Le
Exemple: 00 A4 04 00 06 A0 00 00 00 01 01 00  (SELECT AID)
```

Structure Response APDU

Data : Données retournées (longueur variable)
SW1-SW2 : Code de statut (2 octets)

Structure Response APDU

Codes SW importants

- `90 00` : ✓ Succès
- `63 Cx` : PIN incorrect, x essais restants
- `69 82` : Sécurité non satisfaite
- `69 83` : PIN bloqué
- `6A 82` : Fichier non trouvé
- `6A 86` : P1-P2 incorrect
- `6D 00` : INS invalide
- `6E 00` : CLA invalide

Exemple APDU : SELECT

Command APDU

```
00 A4 04 00 06 A0 00 00 00 01 01
│  │  │  │  │  └──────────────┘
│  │  │  │  │          │
│  │  │  │  │      AID (6 octets)
│  │  │  │  └─ Lc = 6 octets
│  │  │  └─ P2 = 0x00 (Premier/seul)
│  │  └─ P1 = 0x04 (Sélection par AID)
│  └─ INS = 0xA4 (SELECT)
└─ CLA = 0x00 (ISO standard)
```

Response APDU

```
6F 1A 84 06 A0 00 00 00 01 01 A5 10 ... 90 00
└──────────────────────────────────┘  └──┘
              │                        │
      FCI (File Control Info)        SW=OK
```

Interprétation

- Applet avec AID `A0 00 00 00 01 01` sélectionnée avec succès
- FCI contient infos sur l'applet

Exemple APDU : VERIFY PIN

Command APDU

```
00 20 00 01 04 31 32 33 34
│  │  │  │  │  └─────────┘
│  │  │  │  │       PIN "1234" (ASCII: 0x31-0x34)
│  │  │  │  └─ Lc = 4 octets
│  │  │  └─ P2 = 0x01 (Référence PIN)
│  │  └─ P1 = 0x00
│  └─ INS = 0x20 (VERIFY)
└─ CLA = 0x00
```

Réponse si PIN correct

```
90 00  ✓ Succès
```

Réponse si PIN incorrect

```
63 C2  ✗ PIN incorrect, 2 essais restants
       (0x63Cx : x = nombre essais)
```

Exercice 2 : Décoder APDU

Énoncé : Décodez les APDU suivants

```
1. Command:  00 B0 00 00 10
   Response: 48 65 6C 6C 6F 20 57 6F 72 6C 64 00 00 00 00 00 90 00

2. Command:  00 D6 00 00 05 41 42 43 44 45
   Response: 90 00

3. Command:  00 20 00 00 06 31 32 33 34 35 36
   Response: 63 C3

4. Command:  80 10 00 00 08 01 02 03 04 05 06 07 08 00
   Response: 6A 86
```

Questions pour chaque APDU

- Quelle instruction (INS) ?
- Quel est le résultat (SW) ?
- Que signifie l'opération ?

Temps : 10 minutes

Extended APDU

Pour transférer > 255 octets

Structure

Case 2 Extended:

CLA	INS	P1	P2	00	Le ext (2B)

└ Marqueur Extended

Extended APDU

Règles

- `00` après P2 indique Extended APDU
- Lc et Le sur 2 octets (au lieu de 1)
- Lc max = 65535 octets
- Le max = 65536 octets (0x0000 = 65536)

Exemple : Lire 1000 octets

```
00 B0 00 00 00 03 E8
           │  └└─ Le = 0x03E8 = 1000
           └─ Marqueur Extended
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

PC/SC et java.smartcardio

Interface PC/SC

Personal Computer / Smart Card

Définition

Standard multiplateforme pour communication PC ↔️ Lecteurs de cartes

Implémentations

- **Windows** : WinSCard.dll
- **Linux/Mac** : PC/SC Lite (pcscd)

Concepts PC/SC

- **Context** : Environnement d'exécution
- **Reader** : Lecteur physique
- **Card Handle** : Connexion logique à une carte
- **Protocol** : T=0, T=1, T=CL (contactless)

Protocoles

- **T=0** : Character-oriented (byte par byte)
- **T=1** : Block-oriented (par blocs) - Recommandé
- **T=CL** : Contactless (NFC)

Outils de Trace PC/SC

1. pcsc_scan

```
sudo apt-get install pcsc-tools
pcsc_scan
```

- Liste lecteurs et cartes
- Affiche ATR
- Détecte insertion/retrait

2. gscriptor / scriptor

```
gscriptor  # GUI
scriptor   # CLI
```

- Interface pour envoyer APDU
- Historique des commandes
- Débogage interactif

3. apdu4j (Java)

- Bibliothèque Java
- Logging transparent
- Compatible java.smartcardio

4. Wireshark + USB

- Capture trafic USB
- Décode ISO 7816
- Analyse complète

5. DLL Hijacking

- Interception appels PC/SC
- Wrapper custom
- Logging avancé

Démonstration pcsc_scan

```
$ pcsc_scan

PC/SC device scanner
V 1.6.2 (c) 2001-2022, Ludovic Rousseau
Using reader plug'n play mechanism
Scanning present readers...
0: Gemalto PC Twin Reader 00 00

Thu Jan 23 10:15:32 2025
 Reader 0: Gemalto PC Twin Reader 00 00
  Event number: 0
  Card state: Card inserted,
  ATR: 3B 68 00 00 00 73 C8 40 12 00 90 00

ATR: 3B 68 00 00 00 73 C8 40 12 00 90 00
+ TS = 3B --> Direct Convention
+ T0 = 68, Y(1): 0110, K: 8 (historical bytes)
  TB(1) = 00 --> VPP is not electrically connected
  TC(1) = 00 --> Extra guard time: 0
+ Historical bytes: 00 73 C8 40 12 00 90 00
  Category indicator byte: 00 (compact TLV data object)
...
```

ATR = Answer To Reset (carte d'identité de la carte)

API java.smartcardio - Introduction

API Java standard (depuis Java 6)

Package : `javax.smartcardio`

API java.smartcardio - Introduction

Classes principales

Classe	Rôle
`TerminalFactory`	Obtenir la factory PC/SC
`CardTerminals`	Liste des lecteurs
`CardTerminal`	Représente un lecteur
`Card`	Connexion à une carte
`CardChannel`	Canal de communication
`CommandAPDU`	Commande à envoyer
`ResponseAPDU`	Réponse reçue
`ATR`	Answer To Reset

Avantages

- Standard Java
- Multiplateforme
- Simple d'utilisation

Cycle de Vie Connexion

Étapes obligatoires

```
// 1. Obtenir TerminalFactory
TerminalFactory factory = TerminalFactory.getDefault();
CardTerminals terminals = factory.terminals();
// 2. Lister les lecteurs
List<CardTerminal> readers = terminals.list();
// 3. Sélectionner un lecteur
CardTerminal terminal = readers.get(0);
// 4. Vérifier présence carte (optionnel)
if (!terminal.isCardPresent()) {
    terminal.waitForCardPresent(0); // 0 = infini
}

// 5. Se connecter
Card card = terminal.connect("*"); // * = auto-protocole

// 6. Obtenir canal
CardChannel channel = card.getBasicChannel();

// 7. Envoyer APDU
ResponseAPDU resp = channel.transmit(cmdAPDU);

// 8. TOUJOURS déconnecter
card.disconnect(false); // false = pas de reset
```

Bonnes Pratiques Connexion

✓ À FAIRE

- Toujours `disconnect()` en finally
- Vérifier présence carte avant connexion
- Gérer `CardException` et `CardNotPresentException`
- Utiliser protocole `"*"` (auto-négociation)
- Logger les erreurs

✗ À ÉVITER

- Oublier `disconnect()` (fuite ressources)
- Ignorer les exceptions
- Forcer un protocole spécifique sans raison
- Maintenir connexions inutilement longues

Paramètres disconnect()

```
card.disconnect(false); // NE PAS reset (état maintenu)
card.disconnect(true);  // RESET carte (ATR rejoué, RAM effacée)
```

Modes d'Accès et Canaux

Protocoles

```
// Auto (recommandé)
card = terminal.connect("*");

// Spécifique
card = terminal.connect("T=0");
card = terminal.connect("T=1");
card = terminal.connect("T=CL");
```

Basic Channel (canal 0)

```
CardChannel ch =
    card.getBasicChannel();
```

- Toujours disponible
- Partagé
- Un seul par carte

Logical Channels (1-19)

```
CardChannel log1 =
    card.openLogicalChannel();
CardChannel log2 =
    card.openLogicalChannel();
```

- Canaux supplémentaires
- Isolés
- Max 20 canaux (0-19)
- Support carte requis

Fermeture

```
log1.close();
log2.close();
```

Gestion des Sessions

Session = Durée de vie de l'objet `Card`

```
[Carte insérée]
       ↓
  connect() ──────→ [Session ouverte]
       │                    │
       │                    ├─ SELECT applet
       │                    ├─ VERIFY PIN
       │                    ├─ Opérations...
       │                    │
disconnect() ←──────────────┘
       ↓
[Session fermée]
  • État perdu (si reset)
  • RAM effacée (si reset)
  • PIN perdu (si reset)
```

Important

- `disconnect(false)` : État maintenu sur carte
- `disconnect(true)` : Carte resetée
- Pas de mémoire entre sessions Java

Exemple Complet java.smartcardio

```
import javax.smartcardio.*;
import java.util.List;

public class SmartCardExample {
    public static void main(String[] args) {
        Card card = null;
        try {
            // 1. Setup
            TerminalFactory factory = TerminalFactory.getDefault();
            CardTerminals terminals = factory.terminals();
            List<CardTerminal> readers = terminals.list();

            if (readers.isEmpty()) {
                System.err.println("Aucun lecteur détecté");
                return;
            }

            // 2. Connexion
            CardTerminal terminal = readers.get(0);
            System.out.println("Lecteur: " + terminal.getName());

            if (!terminal.isCardPresent()) {
                System.out.println("En attente carte...");
                terminal.waitForCardPresent(0);
            }

            card = terminal.connect("*");
            System.out.println("Protocole: " + card.getProtocol());
            System.out.println("ATR: " + bytesToHex(card.getATR().getBytes()));
```

Exemple Complet (suite)

```
        // 3. SELECT applet
        CardChannel channel = card.getBasicChannel();
        byte[] aid = {(byte)0xA0, 0x00, 0x00, 0x00, 0x01, 0x01};
        CommandAPDU select = new CommandAPDU(0x00, 0xA4, 0x04, 0x00, aid);

        ResponseAPDU resp = channel.transmit(select);
        System.out.println("SELECT SW: " + String.format("%04X", resp.getSW()));

        if (resp.getSW() != 0x9000) {
            System.err.println("SELECT failed");
            return;
        }

        // 4. VERIFY PIN
        byte[] pin = "1234".getBytes();
        CommandAPDU verify = new CommandAPDU(0x00, 0x20, 0x00, 0x00, pin);
        resp = channel.transmit(verify);

        if (resp.getSW() == 0x9000) {
            System.out.println("✓ PIN correct");
        } else {
            System.out.println("✗ PIN incorrect: " +
                String.format("%04X", resp.getSW()));
        }

    } catch (CardNotPresentException e) {
        System.err.println("Carte retirée");
    } catch (CardException e) {
        System.err.println("Erreur: " + e.getMessage());
```

Exemple Complet (fin)

```
        } finally {
            // 5. TOUJOURS déconnecter
            if (card != null) {
                try {
                    card.disconnect(false);
                    System.out.println("Carte déconnectée");
                } catch (CardException e) {
                    System.err.println("Erreur disconnect: " + e.getMessage());
                }
            }
        }
    }

    private static String bytesToHex(byte[] bytes) {
        StringBuilder sb = new StringBuilder();
        for (byte b : bytes) {
            sb.append(String.format("%02X ", b));
        }
        return sb.toString().trim();
    }
}
```

Compilation & Exécution

```
javac SmartCardExample.java
java SmartCardExample
```

Exercice 3 : Application java.smartcardio

Énoncé

Créez une application Java qui :

1. Liste tous les lecteurs disponibles
2. Permet de sélectionner un lecteur
3. Attend l'insertion d'une carte
4. Se connecte et affiche l'ATR
5. Sélectionne une applet (AID fourni par utilisateur)
6. Demande le PIN et vérifie
7. Si authentifié, permet d'envoyer des APDU personnalisées (mode interactif)
8. Déconnecte proprement

Livrables

- Code source Java complet
- Gestion d'erreurs robuste
- Interface en ligne de commande

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Développement Applets Java Card

Java Card - API Essentielle

Package : `javacard.framework`
Classes principales

Classe	Rôle
`Applet`	Classe de base (extends)
`APDU`	Manipulation des APDU
`OwnerPIN`	Gestion des PINs
`JCSystem`	Services système (transactions, mémoire)
`Util`	Fonctions utilitaires (copie, comparaison)
`ISOException`	Exceptions ISO 7816
`ISO7816`	Constantes ISO (SW, offsets)

Contraintes Java Card

- Pas de String, float, double

Structure d'une Applet

```
package com.example.myapplet;
import javacard.framework.*;
public class MyApplet extends Applet {
    // 1. Variables d'instance (EEPROM)
    private OwnerPIN userPIN;
    private byte[] data;
    // 2. Constructeur privé (appelé à l'installation)
    private MyApplet() {
        // Initialisation
        userPIN = new OwnerPIN((byte)3, (byte)4);
        data = new byte[256];
        // Enregistrement
        register();
    }
    // 3. Méthode install (OBLIGATOIRE)
    public static void install(byte[] bArray, short bOffset, byte bLength) {
        new MyApplet();
    }
    // 4. Méthode process (OBLIGATOIRE)
    public void process(APDU apdu) {
        // Gestion de la sélection
        if (selectingApplet()) {
            return;
        }
        // Traitement des commandes
        byte[] buffer = apdu.getBuffer();
        byte ins = buffer[ISO7816.OFFSET_INS];

        switch (ins) {
            //    vos commandes
```

Cycle de Vie Applet

Gestion des APDU

Classe APDU - Méthodes essentielles

```
APDU apdu = ...;

// Obtenir le buffer
byte[] buffer = apdu.getBuffer();

// Lire les champs header
byte cla = buffer[ISO7816.OFFSET_CLA];
byte ins = buffer[ISO7816.OFFSET_INS];
byte p1 = buffer[ISO7816.OFFSET_P1];
byte p2 = buffer[ISO7816.OFFSET_P2];
byte lc = buffer[ISO7816.OFFSET_LC];

// Recevoir les données
short bytesRead = apdu.setIncomingAndReceive();

// Données commencent à ISO7816.OFFSET_CDATA (offset 5)

// Envoyer réponse
apdu.setOutgoingAndSend((short)0, (short)dataLength);

// Lever exception (code SW)
ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
```

Gestion des PINs

Classe OwnerPIN

```
// Création (tryLimit, maxSize)
private OwnerPIN userPIN = new OwnerPIN((byte)3, (byte)8);

// Initialisation
byte[] defaultPIN = {0x31, 0x32, 0x33, 0x34}; // "1234"
userPIN.update(defaultPIN, (short)0, (byte)4);

// Vérification
if (userPIN.check(buffer, offset, length)) {
    // PIN correct
} else {
    // PIN incorrect
    byte remaining = userPIN.getTriesRemaining();
    if (remaining == 0) {
        ISOException.throwIt(ISO7816.SW_AUTHENTICATION_METHOD_BLOCKED);
    } else {
        ISOException.throwIt((short)(0x63C0 | remaining));
    }
}

// Tester si validé
if (userPIN.isValidated()) {
    // Utilisateur authentifié
}
```

Gestion Mémoire Java Card

EEPROM (Persistant)

- Variables d'instance
- Arrays alloués normalement
- Survit aux coupures
- Lent en écriture (~5ms)

```
private byte[] data; // EEPROM

private MyApplet() {
    data = new byte[256];
}
```

RAM (Volatile)

- Variables locales
- Arrays transients
- Rapide
- Effacée régulièrement

```
// Transient array
private byte[] temp =
  JCSystem.makeTransientByteArray(
```

Types Transient

`CLEAR_ON_RESET`

- Effacé au reset carte

`CLEAR_ON_DESELECT`

- Effacé à la désélection

Bonnes pratiques

- Données sensibles → transient
- Buffers temporaires → transient
- Données persistantes → EEPROM
- Minimiser écritures EEPROM

Exemple

```
private byte[] pinBuffer;

private MyApplet() {
    pinBuffer = JCSystem
      .makeTransientByteArray(
        (short)8,
        JCSystem.CLEAR_ON_DESELECT
      );
}
```

Transactions Atomiques

Problème : Coupure pendant opération critique
Solution : Transactions

```
// Opération atomique
JCSystem.beginTransaction();
try {
    // Modifications EEPROM
    balance -= amount;
    transactionCount++;

    // Valider
    JCSystem.commitTransaction();

} catch (Exception e) {
    // Annuler en cas d'erreur
    JCSystem.abortTransaction();
    ISOException.throwIt(ISO7816.SW_UNKNOWN);
}
```

Règles

- Utilisé pour opérations critiques
- Pas de nested transactions

Exemple : Applet Counter

```
package com.example.counter;

import javacard.framework.*;

public class CounterApplet extends Applet {

    // Compteur (EEPROM)
    private short counter;

    // Instructions
    private static final byte INS_INCREMENT = (byte)0x10;
    private static final byte INS_GET_VALUE = (byte)0x20;
    private static final byte INS_RESET = (byte)0x30;

    private CounterApplet() {
        counter = 0;
        register();
    }

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

    public void process(APDU apdu) {
        if (selectingApplet()) {
            return;
        }

        byte[] buffer = apdu.getBuffer();
        byte ins = buffer[ISO7816.OFFSET_INS];

        switch (ins) {
```

Exemple : Applet Counter (suite)

```
            case INS_GET_VALUE:
                getValue(apdu);
                break;
            case INS_RESET:
                reset(apdu);
                break;
            default:
                ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
        }
    }

    private void increment(APDU apdu) {
        // Vérifier overflow
        if (counter == Short.MAX_VALUE) {
            ISOException.throwIt(ISO7816.SW_FILE_FULL);
        }

        // Transaction atomique
        JCSystem.beginTransaction();
        try {
            counter++;
            JCSystem.commitTransaction();
        } catch (Exception e) {
            JCSystem.abortTransaction();
            ISOException.throwIt(ISO7816.SW_UNKNOWN);
        }
    }

    private void getValue(APDU apdu) {
```

Exemple : Applet Counter (fin)

```
        // Écrire valeur (2 octets)
        Util.setShort(buffer, (short)0, counter);
        apdu.setOutgoingAndSend((short)0, (short)2);
    }

    private void reset(APDU apdu) {
        JCSystem.beginTransaction();
        try {
            counter = 0;
            JCSystem.commitTransaction();
        } catch (Exception e) {
            JCSystem.abortTransaction();
            ISOException.throwIt(ISO7816.SW_UNKNOWN);
        }
    }
}
```

Test

```
// INCREMENT
CommandAPDU inc = new CommandAPDU(0x80, 0x10, 0x00, 0x00);
ResponseAPDU resp = channel.transmit(inc);

// GET VALUE
CommandAPDU get = new CommandAPDU(0x80, 0x20, 0x00, 0x00, 0x02);
resp = channel.transmit(get);
short value = getShort(resp.getData(), 0);
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Sécurité et Bonnes Pratiques

Rôles : Security Officer vs User

Security Officer (SO)

Rôle

- Administrateur carte
- Initialisation
- Déblocage PINs
- Gestion clés

SO-PIN

- PIN administration
- Plus d'essais (15+)
- PUK dans certains contextes

Opérations

- Initialiser User PIN
- Débloquer User PIN bloqué
- Réinitialiser carte
- Générer/Injecter clés

Code PKCS#11

Utilisateur Final (User)

Rôle

- Utilisation quotidienne
- Opérations crypto
- Accès données

User PIN

- PIN quotidien
- Moins d'essais (3-5)
- Bloqué après échecs

Opérations

- S'authentifier
- Signer documents
- Chiffrer/Déchiffrer
- Lire données

Implémentation Rôles Java Card

```
public class SecureApplet extends Applet {

    private OwnerPIN userPIN;
    private OwnerPIN soPIN;

    private static final byte USER_PIN_TRY_LIMIT = 3;
    private static final byte SO_PIN_TRY_LIMIT = 15;

    private SecureApplet() {
        userPIN = new OwnerPIN(USER_PIN_TRY_LIMIT, (byte)8);
        soPIN = new OwnerPIN(SO_PIN_TRY_LIMIT, (byte)8);

        // SO-PIN par défaut
        byte[] defaultSOPIN = {0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38};
        soPIN.update(defaultSOPIN, (short)0, (byte)8);
    }

    private void unblockUserPIN(APDU apdu) {
        // Vérifier que SO est authentifié
        if (!soPIN.isValidated()) {
            ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
        }

        // Débloquer et réinitialiser User PIN
        byte[] buffer = apdu.getBuffer();
        byte lc = buffer[ISO7816.OFFSET_LC];
        apdu.setIncomingAndReceive();

        JCSystem.beginTransaction();
        try {
            userPIN.resetAndUnblock();
            userPIN.update(buffer, ISO7816.OFFSET_CDATA, lc);
            JCSystem.commitTransaction();
        } catch (Exception e) {
            JCSystem.abortTransaction();
```

Secure Messaging - Introduction

Protège les APDU entre lecteur et carte

Protection

- **Chiffrement** des données (confidentialité)
- **MAC** (Message Authentication Code) - intégrité
- **Compteur** anti-rejeu

Protocoles

- Global Platform SCP02/SCP03
- ISO 7816-4 Secure Messaging

Clés utilisées

- **ENC** : Chiffrement (3DES, AES)
- **MAC** : Authentification (CMAC)
- **DEK** : Data Encryption Key (optionnel)

Indication dans APDU

```
CLA modifié:
0x00 → 0x0C (Secure Messaging actif)
```

Secure Messaging - Flux

```
1. INITIALIZE UPDATE
   Host → Carte : Challenge host (8 octets)
   Host ← Carte : Challenge carte + Cryptogramme carte

2. Dérivation clés session
   Host : Calcule clés de session (ENC, MAC)
   Carte : Calcule clés de session (ENC, MAC)

3. EXTERNAL AUTHENTICATE
   Host → Carte : Cryptogramme host (prouve connaissance clés)
   Host ← Carte : 90 00 (Authentification mutuelle OK)

4. Communication sécurisée
   Toutes les APDU suivantes sont protégées:
   - Données chiffrées
   - MAC ajouté
   - Compteur incrémenté
```

Overhead

- +16-32 octets par APDU
- Latence augmentée
- Performance réduite

Contraintes Secure Messaging

Performance

- Chiffrement/déchiffrement lent sur carte
- CPU limité (1-5 MHz)
- Overhead significatif

Complexité

- Gestion des clés
- Synchronisation compteurs
- Renouvellement de session
- Debugging difficile

Compatibilité

- Pas toutes les cartes supportent
- Différentes versions (SCP01, 02, 03)
- Configuration requise

Quand utiliser ?

- ✓ Transactions sensibles

Bonnes Pratiques - PINs

✓ À FAIRE

1. Utiliser `OwnerPIN` avec limitations

```
userPIN = new OwnerPIN((byte)3, (byte)8);
```

2. Validation longueur PIN

```
if (lc < 4 || lc > 8) {
    ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
}
```

3. Vérification constant-time (éviter timing attacks)

```
// OwnerPIN.check() est déjà constant-time
if (!userPIN.check(buffer, offset, length)) {
    // Erreur
}
```

4. Ne JAMAIS logger/stocker PIN en clair

Bonnes Pratiques - Clés Crypto

Génération sur carte

```
// Générer paire RSA 2048 sur carte
KeyPair keyPair = new KeyPair(
    KeyPair.ALG_RSA_CRT,
    KeyBuilder.LENGTH_RSA_2048
);
keyPair.genKeyPair();

RSAPublicKey publicKey = (RSAPublicKey)keyPair.getPublic();
RSAPrivateKey privateKey = (RSAPrivateKey)keyPair.getPrivate();

// Exporter publique (OK)
short len = publicKey.getModulus(buffer, (short)0);

// Ne JAMAIS exporter privée
// Elle reste sur carte !
```

Tailles recommandées

- RSA : ≥ 2048 bits
- ECC : ≥ 256 bits
- AES : 128 ou 256 bits

Protection

Bonnes Pratiques - Validation Entrées

TOUJOURS valider P1, P2, Lc

```
public void process(APDU apdu) {
    byte[] buffer = apdu.getBuffer();
    byte ins = buffer[ISO7816.OFFSET_INS];
    byte p1 = buffer[ISO7816.OFFSET_P1];
    byte p2 = buffer[ISO7816.OFFSET_P2];
    byte lc = buffer[ISO7816.OFFSET_LC];

    // Validation Lc
    if (lc > MAX_DATA_LENGTH || lc < MIN_DATA_LENGTH) {
        ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
    }

    // Validation P1
    if (p1 != EXPECTED_P1_VALUE) {
        ISOException.throwIt(ISO7816.SW_INCORRECT_P1P2);
    }

    // Validation P2
    if (p2 < MIN_P2 || p2 > MAX_P2) {
        ISOException.throwIt(ISO7816.SW_INCORRECT_P1P2);
    }

    // Validation données
    apdu.setIncomingAndReceive();
    if (buffer[ISO7816.OFFSET_CDATA] < 0) {
```

Bonnes Pratiques - Sécurité Mémoire

Effacer données sensibles

```
// Buffer temporaire en RAM transient
private byte[] tempBuffer;

private MyApplet() {
    tempBuffer = JCSystem.makeTransientByteArray(
        (short)256,
        JCSystem.CLEAR_ON_DESELECT
    );
}

// Effacement manuel si nécessaire
private void clearSensitiveData() {
    Util.arrayFillNonAtomic(tempBuffer, (short)0, (short)256, (byte)0x00);
}

// Utilisation
public void process(APDU apdu) {
    try {
        // ... opérations avec tempBuffer ...
    } finally {
        clearSensitiveData(); // Toujours effacer
    }
```

Bonnes Pratiques - Protection Attaques

Side-Channel Attacks

- DPA (Differential Power Analysis)
- Timing attacks

Contre-mesures

```
// ❌ MAUVAIS : Temps variable (timing attack)
if (pin[0] == userPin[0]) {
    if (pin[1] == userPin[1]) {
        // ...
    }
}

// ✓ BON : Temps constant
boolean match = true;
for (byte i = 0; i < pinLen; i++) {
    match &= (pin[i] == userPin[i]);
}

// ✓ ENCORE MIEUX : Utiliser OwnerPIN (déjà protégé)
if (userPIN.check(pin, offset, length)) {
    // OK
}
```

Fault Injection

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Projet Final : E-Wallet Sécurisé

Projet E-Wallet - Spécifications

Fonctionnalités

1. **Gestion Solde**
   - Balance initiale : 100 unités
   - Crédit (ajout de fonds)
   - Débit (retrait de fonds)
   - Consultation solde
2. **Sécurité**
   - User PIN (3 essais)
   - Authentification obligatoire
   - Transactions atomiques
   - Validation des montants
3. **Historique**
   - 10 dernières transactions
   - Montants signés (+ crédit, - débit)

Projet E-Wallet - Spécifications

4. **Contraintes**
    - Solde max : 10000
    - Pas de solde négatif
    - Montants strictement positifs

Projet E-Wallet - Instructions

Instructions APDU

INS	Nom	P1	P2	Lc	Data	Le	Description
0x20	VERIFY_PIN	00	00	4	PIN	-	Vérifier PIN
0x30	CREDIT	00	00	2	Montant	-	Ajouter fonds
0x40	DEBIT	00	00	2	Montant	-	Retirer fonds
0x50	GET_BALANCE	00	00	-	-	02	Consulter solde
0x60	GET_HISTORY	00	00	-	-	14	Historique (20 octets)

Projet E-Wallet - Instructions

Format données

- Montants : 2 octets (short) big-endian
- PIN : 4 octets ASCII ("1234")
- Historique : 10 × short (20 octets)

Codes SW

- `9000` : Succès
- `6300` : Vérification échouée
- `63Cx` : PIN incorrect, x essais restants
- `6982` : Sécurité non satisfaite
- `6985` : Conditions non remplies (fonds insuffisants)
- `6A84` : Mémoire insuffisante (solde max dépassé)

Projet E-Wallet - Exercice

Travail à réaliser

1. **Implémenter l'applet complète**
   - Méthodes : verifyPIN, credit, debit, getBalance, getHistory
   - Gestion transactions atomiques
   - Validation entrées
   - Gestion erreurs
2. **Créer application Java de test**
   - Connexion carte
   - SELECT applet
   - Tester toutes les opérations
   - Affichage résultats

Projet E-Wallet - Exercice

3. **Scénarios de test**
    - PIN correct/incorrect
    - Crédit/Débit normaux
    - Tenter débit > solde
    - Tenter crédit > MAX_BALANCE
    - Consulter historique

Livrables

- Code applet Java Card
- Code test Java
- Documentation tests