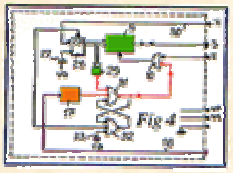
# Smartcards

**Joachim Posegga**

Joachim.Posegga@SAP.com




**30 Jahre...**

## Übersicht Smartcards:

**Aufbau und Funktionsweise**

**Dateisystem (ISO)**

**Kommunikation**

- Protokolle, APDUs, usw.

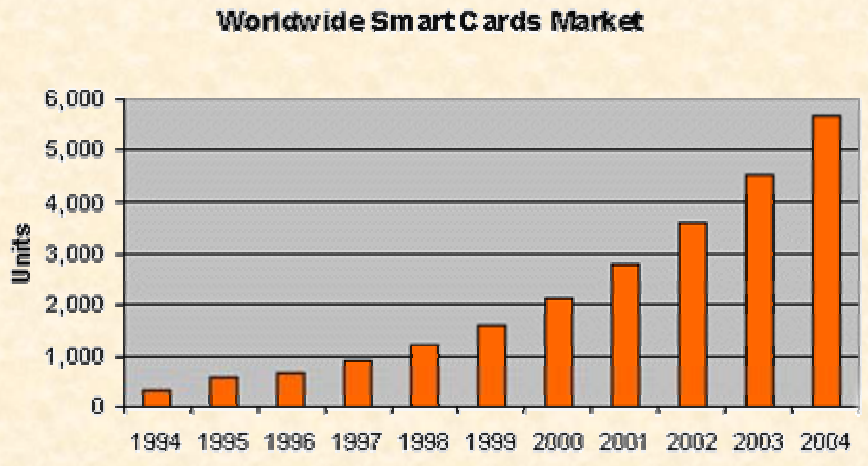
**Beispiele:**

- JavaCard
- Beispiel-Anwendung JavaCard

**Smartcard-Sicherheit**

- Angriffsmethoden, Verteidigungstechniken,...

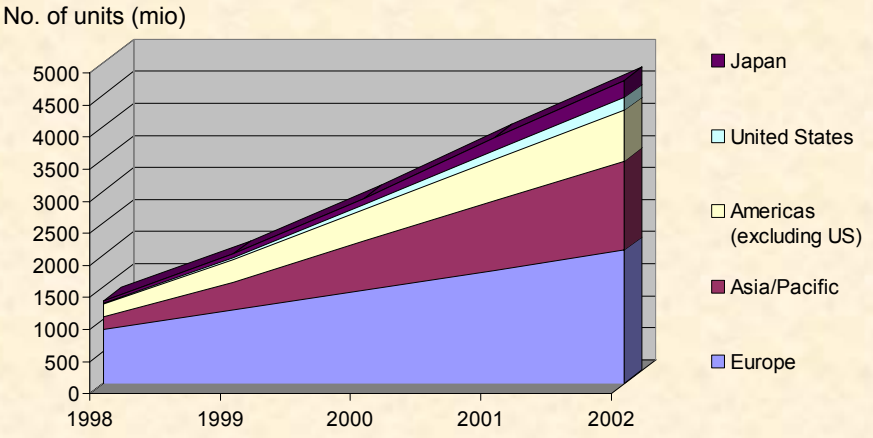
# Frost & Sullivan, 1999






# Worldwide Chipcard Market Forecast

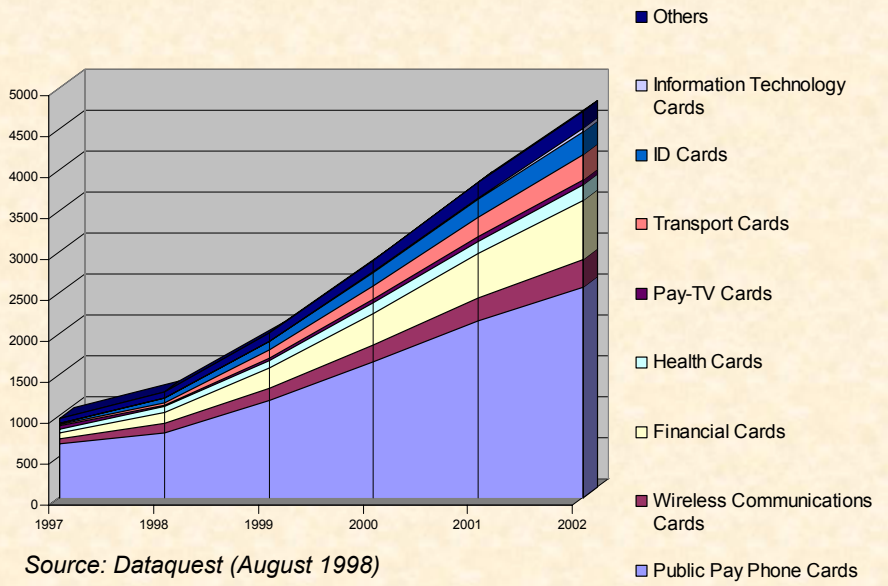
***Regional***




*Source: Dataquest (August 199*

# Worldwide Chipcard Market Forecast

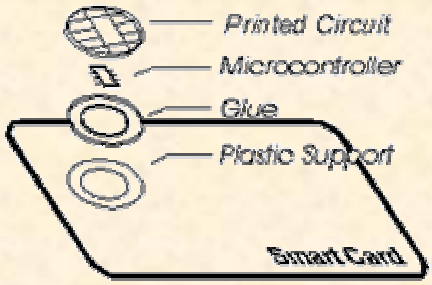
## *Applications*




*Source: Dataquest (August 1998)*



# Chipkarten, IC-Karten, Speicherkarten, Smartcards,...



# Kontaktlose Karten, RFIDs




# RFID Example: Warehouse Management

**Standard barcode label printer fitted with RFID option**

**Special labels with RFID tag embedded**

**In one single operation:**

- Printing of label
- Writing of Data to tag

## Wozu Smartcards ?

**Derzeit:**

- Sicherer "Behälter" für geheime Daten (Schlüssel, etc.)
- Sichere Ausführungsumbegung für Krypto-Algorithmen (Isolierung von sicherheitskritischen Berechnungen)
- **Anwendungsbereiche**:
  - Finanzbereich: EC-Karte, Kreditkarte, Electronic Purse,...
  - Telekommunikation:Telefonkarte, Subscriber-Identifikation im Mobilfunk [*näheres im GSM-Kapitel* ],...
  - Gesundheitswesen, Loyalty-Cards, PayTV,...

**Morgen:**

- Sichere Software-Plattform, "trusted computing base",...?



## Smartcard-"Player"

**„Hardware":**

- Chip-Produzenten (Siemens, Motorola, Thompson,...)
- Kartenhersteller (Gemplus, ORGA, Giesecke & Devrient, Schlumberger,...)
- Kartenherausgeber (Krankenkassen, Banken, Telcos,...)
- Karteninhaber (Kunden)

**„Software":**

- Hersteller Karten-Betriebssystem (MFC, Starcos, TCOS,...)
- Karten-Anwendung (z.B. elektronische Geldbörse)
  - „Application Developer"
  - „Application Provider"

## Lebenszyklus

**Fertigungsphase (Kartenhersteller)**

- Vom Wafer zur Plastikkarte
- Test der Karte, Einbringen d. Personalisierungsschlüssels
- ...

**Personalisierung**

- Einbringen von Dateien, personalisierten Daten
- PIN, PUK, usw.

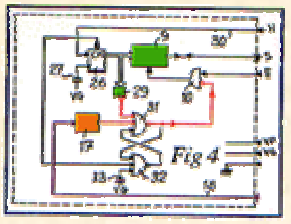
**Nutzung der Karte ...**

**„Invalidation Phase"**

- Sperrung der Karte im Hintergrundsystem



## Typen von Smartcards




**Speicherkarten**

- Magnetstreifen
- IC-Cards

**Prozessorkarten**

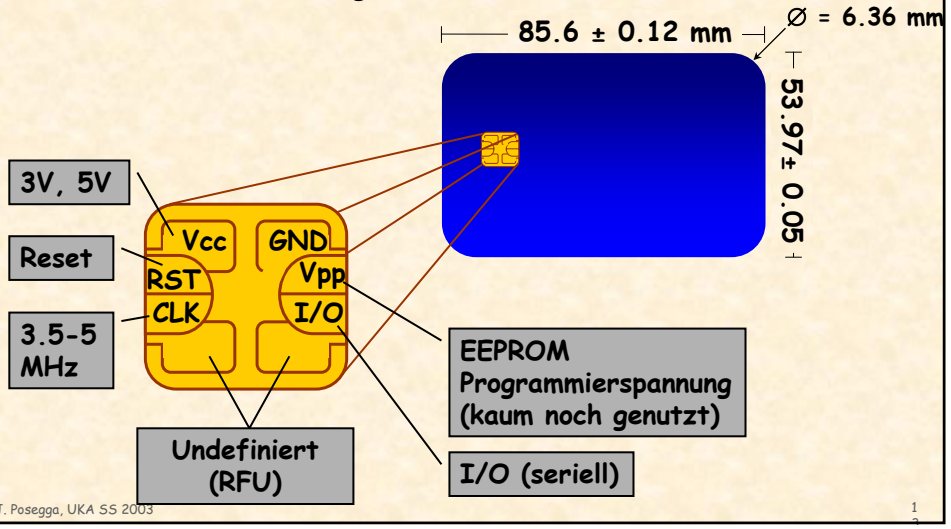
- Kontaktlose Karten (Stromversorgung durch Induktion)
- „Klassische" Prozessorkarten

**Geschichte:**

- Idee: Jürgen Dethloff, Helmut Grötrupp, 1968
- Erstes Patent: Roland Moreno, Frankreich 1970
- FT: erste Telefonkarte 1984, DT: 84/85

# ISO 7816

ISO 7816-1: Physikal. Eigenschaften
-2: Dimension und Ort d. Kontakte
-3: elektron. Eigenschaften






# Kartentypen: Speicherkarten

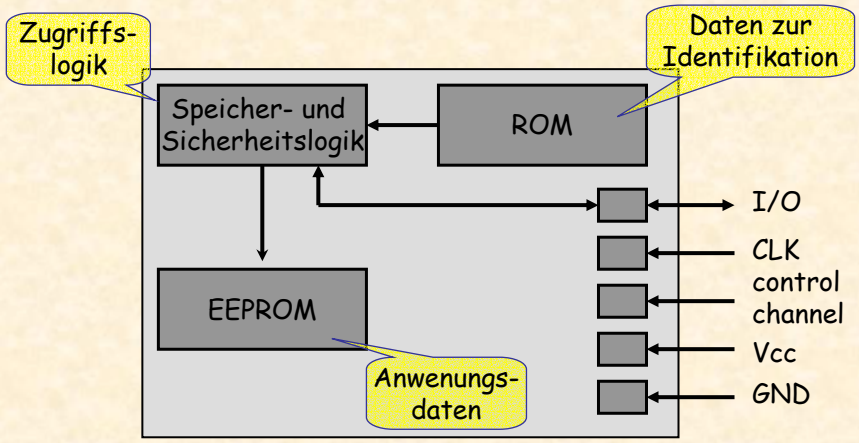
**Speicherkarten ("memory cards")**

- Preiswert (< 1 EURO)
- Stellt geringe Anforderung an Kartenleser ("CAD")
- Reines "Transportmedium" für Daten (üblicherweise mit internem Zugriffsschutz f. best. Seicherbereiche)
- Speicherplatz 100 Byte - 8 KB
- Relativ niedriger Sicherheitslevel

**Typische Anwendungen:**

- "Prepaid-Karten" (Telefonkarte, usw.)
- Krankenversicherungskarte
- "Loyalty-Cards" (LH Miles & More, usw.)

## Speicherkarte





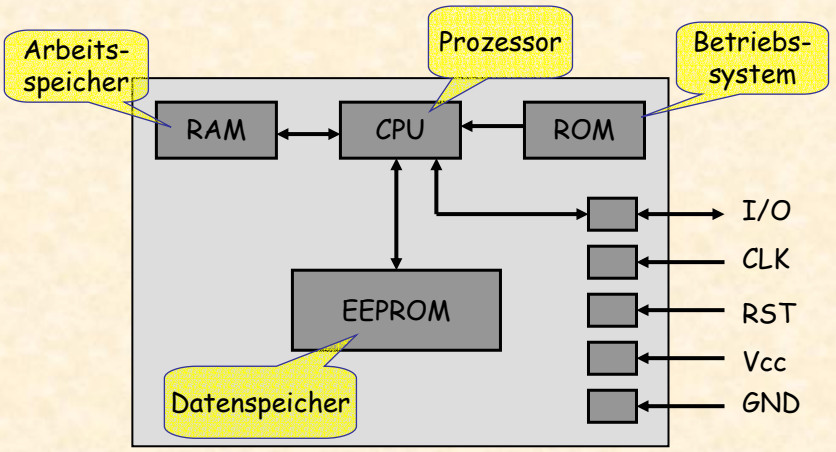

## Kartentypen: Prozessorkarten

**"IC-Karten", Smartcards**

- Karte mit internem Prozessor (typisch: 8-Bit Wortbreite, es gibt aber auch 32-Bit Prozessoren)
- Preis 2-10 EURO
- Karte kann intern Berechnungen dürchführen, es müssen also nicht alle Daten die Karte verlassen (Sicherheit!)
- Speicherplatz typ. 4-64 KByte (EEPROM), 6-32 KByte ROM, 256-2048 Byte RAM

**Trend: Ausführungsplattformen ("virtuelle Maschine") in der Karte (JavaCard, Smartcard for Windows,...)**

## Prozessorkarte






## Randbedingungen bei der Programmierung

- Extrem hohe Sicherheitsanforderungen
- Extreme Beschränkung der Ressourcen in der Karte (I.d.R.: Kampf um jedes Byte)
- Software in der Karte ist schwer zu debuggen (Bei der Entwicklung: Einsatz von Simulatoren)
- "Patches" sind nach der Kartenausgabe schwer möglich
- Bei laufendem Programm kann jederzeit der Strom ausfallen
- Karte "spielt" den Server, das Terminal den Client
- Das Programm "verliert" seinen Zustand nach einer Ausgabe
- Die Berechnungen sollen nicht "beobachtbar" sein

# Smartcard Dateisystem

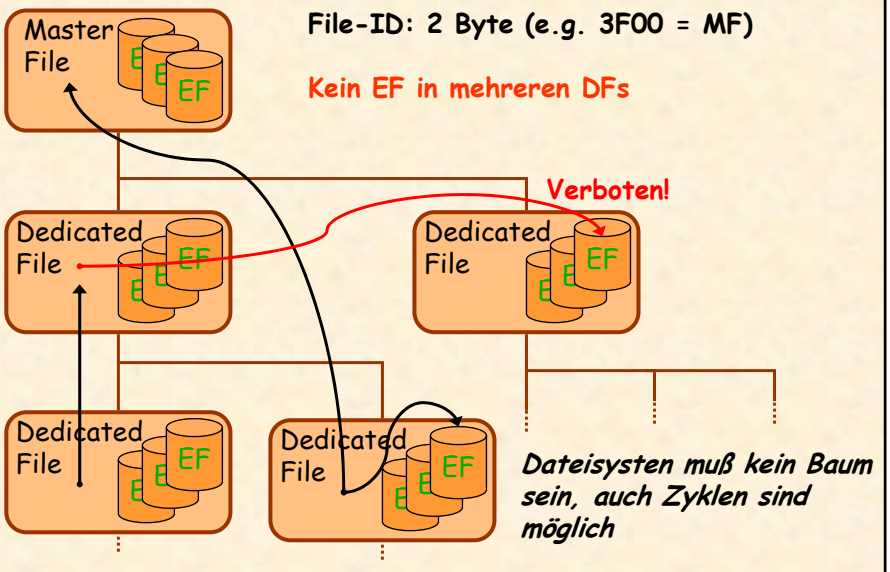
**Dateibaum im EEPROM**
**(mit der Festplatte eines PCs vergleichbar)**

**3 Dateitypen:**

- **Master File (MF):** Wurzel des Dateibaums (wird laut ISO mit 3F00 bezeichnet)
- **Dedicated File (DF):** "Directory für Anwendungen" (Separierung von Daten aus Sicherheitsgründen)
- **Elementary File (EF):** enthält Daten
  - external EF: auch außerhalb der Karte zugänglich
  - internal EF: nur innerhalb der Karte zugänglich



# Smartcard Dateisystem



## Dateitypen

**Transparent (binär, amorph),** unstrukturiert, block-weises Lesen "(read|write|update) binary"

**Linear fixed,** Felder gleicher Länge, "(read|write|update) record"

**Linear variable,** Felder (1 ... FE) unterschiedlicher Länge (1-254 Byte), (read|write|update) record"

**Cyclic,** wie linear fixed, jedoch zyklische Struktur; Die Datei besitzt einen Zeiger auf das zuletzt beschriebene Feld, der nach Dateiende wieder auf das erste Feld der Datei zeigt.



## Dateizugriff

**EF:**

- APPEND — Anhängen an die Datei
- DELETE FILE — Löschen
- INVALIDATE — Sperren der Datei
- LOCK — Reservieren der Datei
- READ/SEEK — Leses/Suchen
- REHABILITATE — Entsperren
- WRITE/UPDATE — Schreiben

**DF:**

- CREATE/DELETE — Erzeugen/Löschen
- REGISTER — In DF bekannt machen

## Zugriffssteuerung („Access Control")

**Always (ALW):** Keine Einschränkung.

**Card holder verification 1 (CHV1):** Zugriff erst nach Eingabe der PIN #1.

**Card holder verification 2 (CHV2):** Zugriff erst nach Eingabe von PIN #2.
(i.d.R. zum Zurücksetzen der Karte nach Falscheingabe von PIN1)

**Administrative (ADM):** Kartenherausgeber, bzw. -hersteller definiert Zugriffsrechte und deren Verifikation.

**Never (NEV):** Niemals zugänglich.



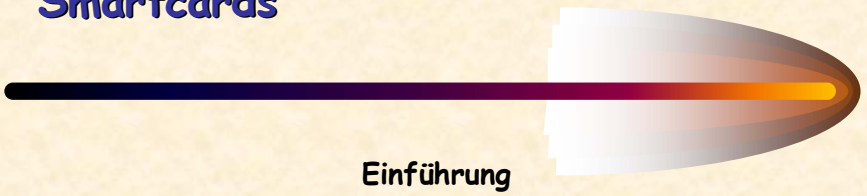
## PIN

**Die PINs liegen üblicherweise in Dateien (z.B. $EF_{CHV1}$ und $EF_{CHV1.}$)**

**Mögliche Zustände der Karte:**

- *PIN nicht eingegeben*
- *PIN-Abfrage erfolgreich.*
  PIN wurde korrekt eingegeben, PIN-Zähler zurückgesetzt (meist auf den Wert 3)
- *PIN falsch eingegeben*
  PIN-Zähler wird herunter gezählt
- *PIN blockiert*
  PIN-Zähler = 0; Zähler kann üblicherweise durch CHV2 („PUK") zurückgesetzt werden.

# Smartcards



Einführung

Kommunikation mit der Karte

Java Smartcards

Smartcard-Sicherheit

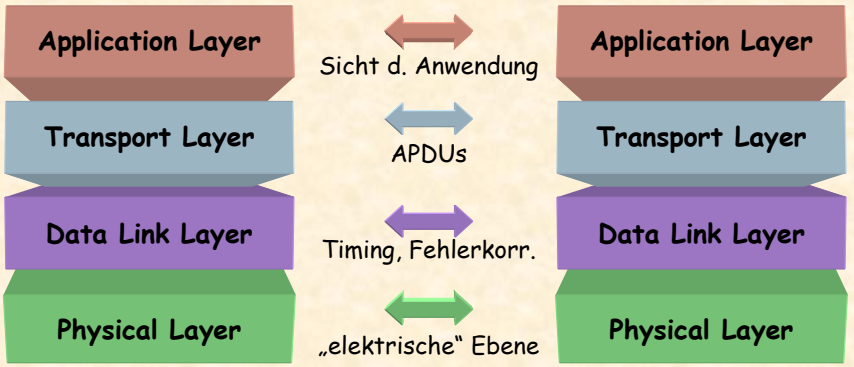
## Kommunikation Karte/Kartenleser

- Existierende Karten bieten half-duplex Protokolle (9,600 bit/s)
- Die Kommunikation wird immer vom Terminal angestoßen (Karte = Server!)

**Vorgehensweise:**

- Karte erhält Vcc und CLK, führt "power-on-reset" durch
- Karte sendet "answer to reset" (ATR) an Terminal (ATR-String enthält informationen über die Karte)
- Optional: Terminal sendet PTR (= protocol type selection) Anweisung an die Karte
- Danach: Terminal sendet erstes Kommando an Karte

# Kommunikationsschichten

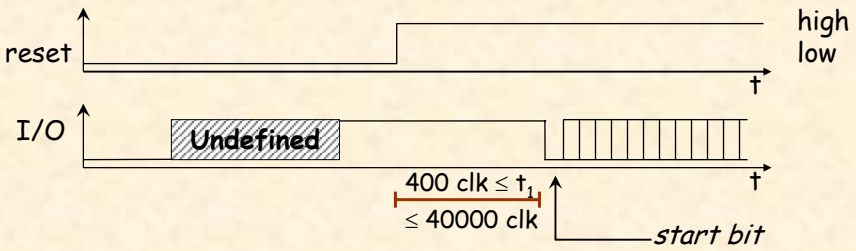





# ANSWER To RESET (ATR)

**Initiale Antwort der Karte nach "power-up/reset"**

- Karte identifiziert sich und gibt dem Terminal Informationen über mögliche Protokolle
- Max. Länge des ATR-Strings: 33 Bytes.
- ATR muß zwischen dem 400. Und 40000. Takt erfolgen (d.h. höchstens nach 10 ms):



## ATR-Format

| TS | T0 | Interface Chars | Historical Chars | TCK |
|---|---|---|---|---|
| 3B | B5 | 110081314615 | 5620312E30 | 1E |

B5 → = |"V 1.0"|

**TS:** "Initial Character", legt "low-level" Protokollparameter fest (z.B. festes Bit-Pattern (3B) für Zeitmessungen

**T0:** "Format Character", kündigt an, welche interface chars folgen und definiert Läne der historic chars

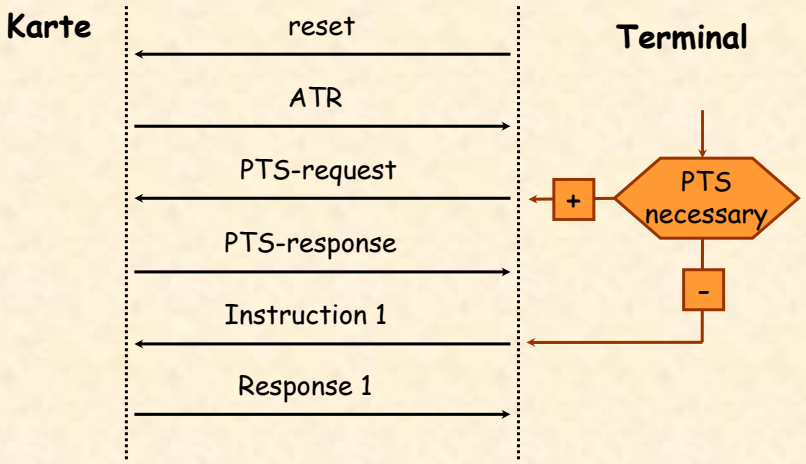
**ICs:** Definition diverser Protokollparameter (T=0,...), Programmierspannung f. EPROM, usw; heute oft nicht mehr notwendig

**HCs:** Information über OS-Version, Chip, Maskenversion, usw.

**TCK:** "Check Character", XOR-Checksumme der vorherigen Bytes



## Initialisierung d. Kommunikation Terminal/Karte



## Übertragungsprotokolle

**T=0** Ansynchron, halb-duplex, **byte**-orientiert (ISO/IEC 7816-3)

**T=1** Ansynchron, halb-duplex, **block**-orientiert (ISO/IEC 7816-3 Adm 1)

**T=2** Ansynchron, voll-duplex, **block**-orientiert (ISO/IEC 7816-4), derzeit nicht implementiert

**T=3... T=13:** "future use"

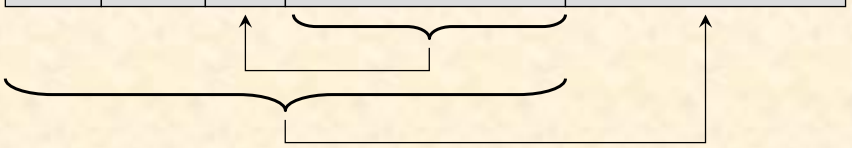
**T=14** Deutsche Telekom proprietär (Telefonkarte)

**T=15** "future use"



## T=1 ("physikalisch")

| *Prolog* | | | *Information* | *Epilog* |
|---|---|---|---|---|
| NAD | PCB | LEN | APDU | EDC |
| 1 Byte | 1 Byte | 1 Byte | 0-254 Bytes | 1-2 Bytes |



**NAD:** Node Address

**PCB:** Protocol Control Byte

**LEN:** Länge APDU

**EDC:** Error Detection Code

## Format einer Instruktions-APDU
## (Application Protocol Data Unit)

| *Header* | | | | *Body* | | |
|---|---|---|---|---|---|---|
| CLA | INS | P1 | P1 | Lc Feld | Daten | Le Feld |
| max. 254 Bytes | | | | | | |

**CLA:** Element Klasse (z.B. GSM = A0, ISO = 0X)

**INS:** Instruktion

**P1,P2:** Parameter (Optionen)

**Lc:** "Length Command", Länge der Daten

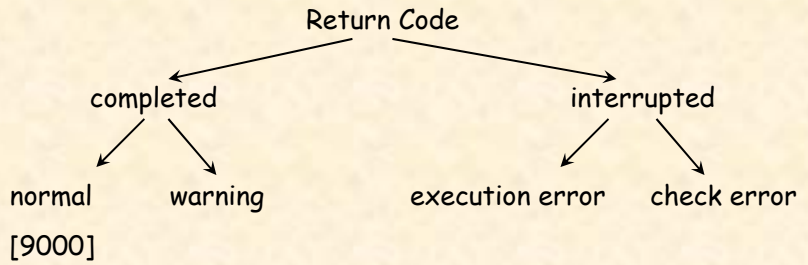
**Le:** "Length expected", erwartete Länge der Antwort der Karte



## Response APDU

| *Header* | *Trailer* | |
|---|---|---|
| Daten | SW1 | SW2 |

**SW1,SW2: Return Code**



## Smartcard Instruktionen

**Interface des Karten-Betriebssystems zum Terminal**

- Diverse Standards (z.B. ISO/IEC 7816-4, GSM 11.11,...) definieren Instruktionssätze
- Verschiedene Karten variieren sehr stark

**Beispiele:**

- Datei-Operationen (Select, Read, Write, Seek,...)
- Authentifikation, Verschlüsselung
- Instruktionen für elektronisches Geld
- Instruktionen zum Test der Karte
- ...



## Datei-Instruktionen

**SELECT FILE [ FID (EF, DF, MF) | AID (DF) | Pfad | ...]**

- Return code zeigt an, ob Datei gefunden

**READ BINARY [Anz. Bytes] [Offset]**

- Ergebnis: Daten oder Fehlercode

**SEEK**

- Länge des Suchstrings
- Suchstring
- Offset
- Modus (vom Anfang, vorwärts, rückwärts,...)
- Länge des Rückgabestrings

## Beispiele Sicherheits-Instruktionen

VERIFY CHV [PIN] [PIN Id]

CHANGE CHV [PIN Alt] [PIN neu] [PIN Id]

UNBLOCK CVH [PUK] [PIN neu] [PIN Id]

(PUK=Personal Unblocking Key)

INTERNAL AUTHENTICATE [RAND] [Alg. Id] [Key Id]

Berechnet $enc_{Alg}$(Key, RAND), Chall./Resp.-Auth. d. Karte

GET CHALLENGE

EXTERNAL AUTHENTICATE

[$enc_{Alg}$(Key, RAND)] [Alg. Id] [Key Id]

Zur Auth. des Terminals gegenüber der Karte



## Protokoll-Instruktionen

GET RESPONSE [Länge]

Anwendungen in einer Karte können nicht "selbstständig" Daten an das Terminal senden. Oft wird das Vorhandensein von Antwortdaten dem Terminal durch Response-Code signalisiert, das Terminal fordert dann Daten an.

ENVELOPE [Instruktions-APDU]

Verschlüsselte Übertragung einer APDU durch Einbettung in eine andere APDU (bei T=0 muß der Instruktions-Byte und die Le unverschlüsselt sein)

## Beispiel "Dialog" mit einer GSM-Karte...

```
r                                  # Reset
3b 83 00 12 10 96                  # Answer to Reset (ATR) der Karte
...
t 7f 20                            # select gsm_dir
9f 19                    # ok, 19 Bytes Resultat verfuegbar
...
t a0 c0 00 00 19                   # get response
c0 00 00 00 00 7f 20 02 00 00 11 00 01 0c 1b 00
12 04 00 83 8a 03 8a 00 03 83 90 00 # uninteressant
...
t 35 35 35 35 ff ff ff ff   # PIN (5555)
90 00                              # ok
```



## Smartcards

Einführung

Kommunikation mit der Karte

Java Smartcards

Smartcard-Sicherheit

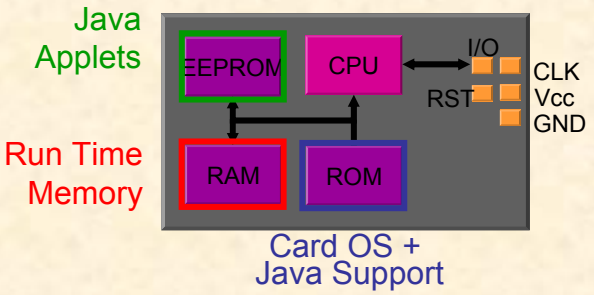
## Chipkarten-Technologie: Trends & Folgen

- Chipkarten vollziehen heute die Entwicklung der Informatik in den 80er Jahren nach:
  - Trennung von Plattform (Hardware, OS) und Anwendung
  - Einsatz höherer Programmiersprachen und -konzepte
  - Weg von der "Geheimwissenschaft", hin zur "Allerweltstechnologie"
- Folge: Marktexpansion, neue Anwendungsentwickler, neue Anwendungsbereiche, Umbau der Branche,...
- Chipkarten entwickeln sich vom Krypto-Server hin zur sicheren Software-Plattformen in verteilten Systemen
  *Forschungsfragen: Middleware, Anpassung v. CADs, usw.*



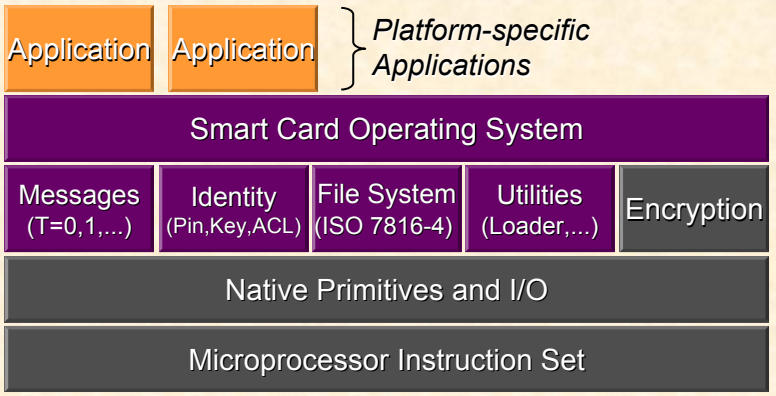
## Java Smart Card Silicon Architecture

Java Smart Card:
Stripped down
Java VM
("JavaCard")
inside Smart Card



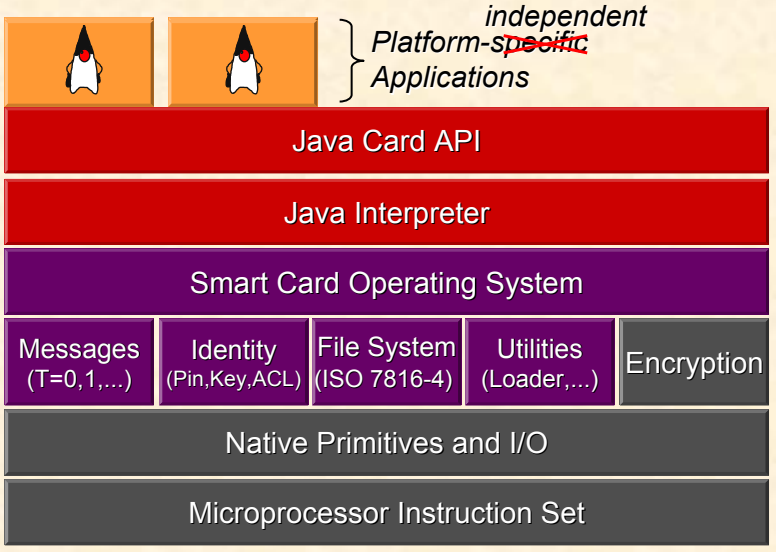

- Easier and faster development of card applications
- Interoperability
- Applications can be loaded at any time into the card
- Integration into state-of-the-art technology (Java, etc.)

# Java Card Software Architecture

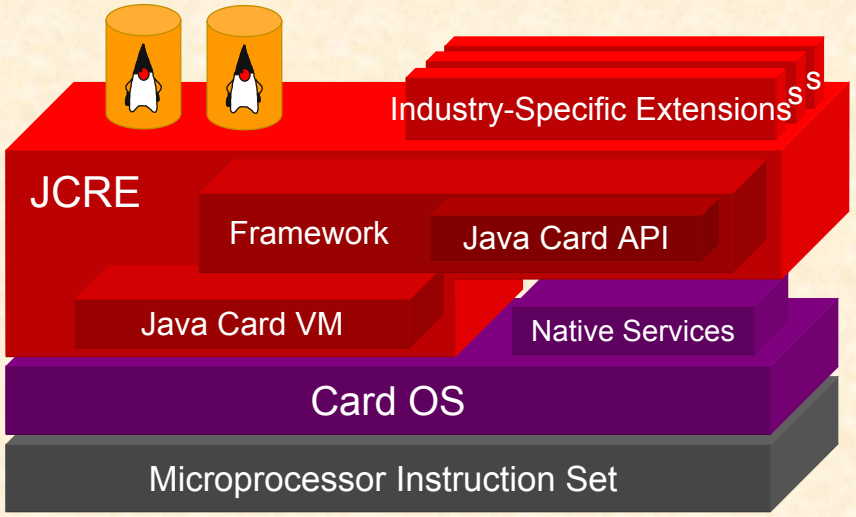





# Java Card Software Architecture



## Java Card Runtime Environment (JCRE)






## Vorteile von Java-Karten

**Entwicklung der Kartensoftware wird wesentlich vereinfacht:**

- Klare Trennung von Plattform (Hardware + OS) und Anwendungssoftware
- Anwendungsentwicklung in Java statt Maschinensprache
- Nutzung von "off-the-shelf"-Karten
- "Time to market" Wochen statt Monate
- Ideale Architektur für Multi-Applikationskarten
- ? Interoperabilität, d.h. Anwendungen sind unabhängig von der konkreten Kartenplattform

## Vorteile von Java-Karten

**Mehr Flexibilität: Kartensoftware kann zu einem beliebigen Zeitpunkt in die Karte geladen bzw. ausgetauscht werden:**

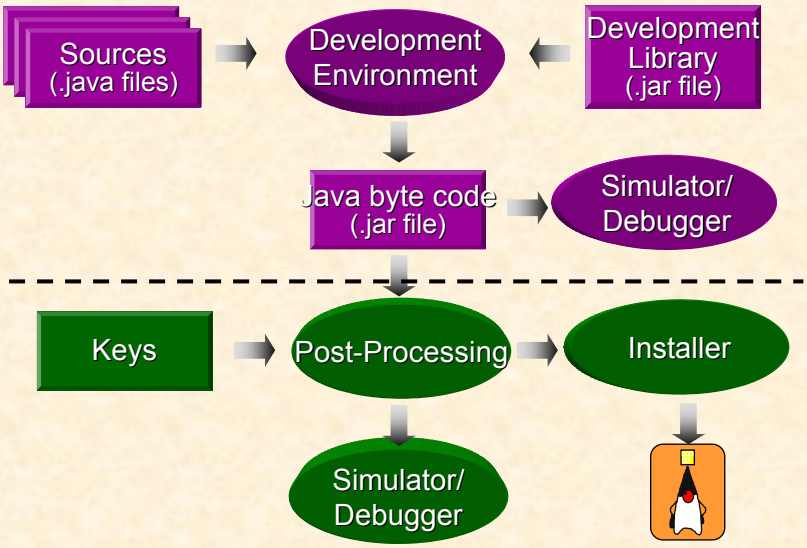
- Bei der Personalisierung
- POS (T-Punkt)
- Beim Kunden
  (Internet-PC mit CAD, ScreenPhone, GSM-ME...)
- In der Telefonzelle (theoretisch...)



## Nachteile von Java-Karten

- High-end Prozessor erforderlich, daher relativ hoher Preis
- Ausführungsgeschwindigkeit geringer als bei native Code (Faktor 3-5).
- Technologie ist noch realtiv neu, der Einsatz daher mit einem gewissen Risiko verbunden
- Software-Update nach Personalisierung erfordert Infrastruktur
- Management des Karten-Lebenszyklus wird komplizierter

## Development of Java Card Applets






## JavaCard Language Subset

**JavaCard unterstützt nur eine Untermenge von Java, insbesondere fehlen:**

- Dynamic class loading
- Threads, Synchronization
  *Jedoch bei GSM: SIM Application Toolkit erfordert Context-Switching...*
- Object cloning
- Große primitive Datentypen (float, double, long, char,...)

**Schwierig zu realisieren:**

- Garbage-Collection

## JavaCard Framework

**javacard.framework**

- Core package on the card; defines classes such as
  - **Applet**, **PIN:** fundamental building blocks for Java Card programs
  - **APDU**, **System**, **Util:** runtime and system service to Java Card programs

**javacardx.framework**

- provides an ISO 7816-4 compatible file system. Supports elementary files (EF), dedicated files (DF) and file-oriented APDUs as specified in ISO7816

**javacardx.crypto**

- cryptographic functionality



## JavaCard: Interna

**JCRE (Java Card Runtime Environment)**
**= Java Card VM + Klassen im Java Card Framework**

**Ladeprozess:**

- Laden eines Applets in die Karte
- Jedes Applet besitzt eine eindeutige AID
- JCRE ruft die Install-Methode

```
public static install
```

  des neuen Applets als letzten Schritt der Installation auf. Diese Methode initialisiert das Applet und registriert es beim JCRE
- Das Applet bleibt passiv, bis es selektiert wird

## JavaCard: Interna

- Zum Selektieren schickt das Terminal

  **SELECT APDU <AID_neu>**

  an JCRE.

- JCRE ruft

  **<AID_alt>.deselect()**

  **und**

  **<AID_neu>.select()**

  auf.

- Die nächste APDU geht dann an **<AID_neu>** zur Verarbeitung.



## Definition des Package

```
package bank.purse; // Package-Deklaration

        // Import der benötigten Klassen-Dateien:
import javacard.framework.*;
import javacardx.framework.*;

        // Definition der Klasse als Erweiterung von
        // javacard.framework.Applet
public class Wallet extends Applet {
   /* Deklaration von Konstanten */
        // Inhalt des CLA-Bytes in der Kommando-APDU
        // identifiziert die Anwendung
final static byte Wallet_CLA =(byte)0xB0;
```

# Datenstrukturen

```
// Definition der  INS-Bytes der Kommando-APDU, also der
// verfügbaren Instruktionen
final static byte Deposit = (byte) 0x10;
final static byte Debit = (byte) 0x20;
final static byte Balance = (byte) 0x30;
final static byte Validate = (byte) 0x40;
// Anzahl der Versuche bei PIN-Eingabe, PIN-Länge
final static byte PinTryLimit =(byte)0x03;
final static byte MaxPinSize =(byte)0x04;
// Statuswort als Ausgabe an Terminal wenn Guthaben negativ:
final static short SW_NEGATIVE_BALANCE =
  (short) 0x6910;
// Instance-Variablen
OwnerPIN pin;   byte balance;
byte buffer[]; // APDU buffer
```



**Class** `javacard.framework.OwnerPIN`
**Constructor:**

```
public OwnerPIN(byte tryLimit,
    byte maxPINSize) throws PINException
```

Allocates a new PIN instance.

**Parameters:**
`tryLimit` - the maximum number of times an incorrect PIN can be presented.
`maxPINSize` - the maximum allowed PIN size. `maxPINSize` must be >=1.

**Throws:** `PINException`
with the following reason codes:
- `PINException.ILLEGAL_VALUE` if `maxPINSize` parameter is less than 1.

## Der Constructor

```
/* Der Constructor dient dazu, Speicher zu belegen, den das Applet
   in seinem Lebenszyklus benötigt und das Applet in der Karte zu
   registrieren:*/

private Wallet() {
   // Neues PIN-Objekt erzeugen:
   pin = new OwnerPIN(PinTryLimit, MaxPinSize);
   // Setzte den Wert der Geldbörse auf Null:
   balance = 0;
      // Das Applet registriert sich beim JCRE
         // (Methode aus der Oberklasse Applet):
   register();
   } // end of the constructor
```



## Installation des Applets

```
/* Die Install-Methode wird vom JCRE als letzter Schritt bei der
   Installation des Applets aufgerufen.
   Als Parameter wird die entsprechende APDU mitgeliefert, die
   jedoch hier nicht genutzt wird.*/

public static void install(APDU apdu){
   // generiere eine neue Instanz von Wallet
   new Wallet();
        } // end of install method
```

**Class** `javacard.framework.Applet`

**Method:**
```
public static void install(byte bArray[],
      short bOffset, byte bLength)
      throws ISOException
```

To create an instance of the Applet subclass, the JCRE will call the static `install()` method first. The applet should perform any necessary initializations and must call one of the `register()` methods.

The installation is considered successful when the call to `register()` completes without an exception. Successful installation makes the applet instance capable of being selected via a SELECT APDU command.

**Parameters:**

`bArray` - the array containing installation parameters.

`bOffset` - the starting offset in `bArray`.

`bLength` - the length in bytes of the parameter data in `bArray`. The maximum value of `bLength` is 32.



# Selektierung des Applets

```
/* Wird ein installiertes Applet per APDU selektiert, so ruft das
   JCRE die select-Methode des Applets auf.
   In dieser Methode wird das Applet in einen konsistenten
   Initialzustand gebracht, also Daten auf Default-Werte gesetzt,
   usw. und das Applet auf das Lesen von APDUs vorbereitet */

public boolean select() {
   // setze die PIN zurück
   pin.reset();
   // Liefere True zurück, um dem JCRE zu signalisieren, daß das
        // Applet nun APDUs verarbeiten kann
   return true;
  } // end of select method
```

**Class** `sendBytes`

**Method:** `public boolean select()`

Called by the JCRE to inform this applet that it has been selected. It is called when a SELECT APDU command is received and before the applet is selected. SELECT APDU commands use instance AID bytes for applet selection. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

A subclass of Applet should override this method if it should perform any initialization that may be required to process APDU commands that may follow. This method returns a boolean to indicate that it is ready to accept incoming APDU commands via its `process()` method. If this method returns false, it indicates to the JCRE that this Applet declines to be selected.

The implementation of this method provided by Applet class returns true.

**Returns:**

`true` to indicate success, `false` otherwise.

# APDU-Verarbeitung

```
/* Nachdem ein Applet erfolgreich selektiert wurde, kann das JCRE
   APDU an das Applet weiterreichen. Dies geschieht in Form von
   APDU-Objekten, die Ein- und Ausgaben zwischen Applet und
   Terminal transportieren und von dem low-level Protokoll (T=0, etc)
   abstrahieren: */
public void process(APDU apdu) {
        //  Ein APDU-Objekt enthält ein Byte-Array (zugänglich mit der
        // Methode „getBuffer"), das den Inhalt der APDU enthält:
    buffer = apdu.getBuffer();
        // Teste, ob das Applet für diese APDU „zuständig" ist, falls nein:
        // Exception.
        // Eine Exception kann vom Applet selbst abgefangen
        // werden, oder (Default) vom JCRE; die Exception enthält ein
        // Status-Wort, das an das Terminal zurückgeliefert wird.
    if (buffer[ISO.OFFSET_CLA] !== Wallet_CLA)
      ISOException.throwIt(ISO.SW_CLA_NOT_SUPPORTED);
```



**Class** `javacard.framework.Applet`

**Method:** `public abstract void process(APDU apdu) throws ISOException`

Called by the JCRE to process an incoming APDU command. An applet is expected to perform the action requested and return response data if any to the terminal. Upon normal return from this method the JCRE sends the ISO 7816-4 defined success status (90 00) in APDU response. If this method throws an ISOException the JCRE sends the associated reason code as the response status instead.

*APDU buffer[5..] is undefined and should not be read or written prior to invoking the* `APDU.setIncomingAndReceive()` *method if incoming data is expected. Altering the APDU buffer[5..] could corrupt incoming data.*

**Parameters:** `apdu` - the incoming APDU object

**Throws:** `ISOException` with the response bytes per ISO 7816-4

# APDU-Verarbeitung (2)

```
/* Das „INS"-Byte der APDU spezifiziert, welche Instruktion ausgeführt
   werden soll.
   Hier wird eine entsprechende Methode gewählt, die die Instruktion
   ausführt und Anworten für das Terminal in der APDU speichert.
   Diese werden dann vom JCRE zurückgeliefert.*/

switch (buffer[ISO.OFFSET_INS]) {
      case Balance:   getBalance(apdu); return;
      case Debit:   debit(apdu); return;
      case Validate: validate(apdu);return
      default:   ISOException.throwIt
                        (ISO.SW_INS_NOT_SUPPORTED);
      }
 }    // end of process method
```



**Class** `Class javacard.framework.ISOException`

**Method:** `public static void throwIt(short sw)`

Throws the JCRE owned instance of the ISOException class with the specified status word.

**Parameters:**

`sw` - ISO 7816-4 defined status word

**Throws:** `ISOException`

---

**Interface** `javacard.framework.ISO7816`

**Variable:** `public static final short SW_INS_NOT_SUPPORTED`

Response status : INS value not supported = 0x6D00

# Deposit-Methode

```
private void deposit(APDU apdu) {
        // Werfe Exception, falls die PIN noch nicht erfolgreich
        // eingegeben wurde:
    if ( ! pin.isValidated() )
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);
        // Erinnerung: Die ersten 5 Byte der APDU sind:
        // (CLA, INS, P1, P2, Lc/Le); diese sind im Puffer enthalten.
        // Bestimme die Länger der zu lesenden Daten (Lc):
    byte numBytes = (byte) (buffer[ISO.OFFSET_LC]);
        // Das Datenfeld der ADPU ist optional, daher muß das Applet
        // dem JCRE die Daten-Bytes in der APDU
        // explizit anfordern.
    byte byteRead =
      (byte)(apdu.setIncomingAndReceive());
```



**Class** `javacard.framework.APDU`

**Method:** `public short setIncomingAndReceive() throws APDUException`

This method gets as many bytes as will fit without buffer overflow in the APDU buffer following the header. It gets all the incoming bytes if they fit.

This method may only be called once in a `Applet.process()` method.

**Returns:** number of bytes read. Returns 0 if no bytes are available.
Throws: `APDUException` with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setIncomingAndReceive()` already invoked or if `setOutgoing()` or `setOutgoingNoChaining()` previously invoked.
- `APDUException.IO_ERROR` on I/O error.

## Deposit-Methode (2)

```
        // Fehler falls mehr als ein Byte vorhanden ist
    if (byteRead != 1 || numBytes != 1)
      ISOException.throwIt(ISO.SW_WRONG_LENGTh);
        // Addiere den eingezahlten Betrag zum Guthaben
balance = (byte) (balance +
      buffer[ISO.OFFSET_CDATA]);

        // Die Deposit-Methode liefert keine Ausgabe an das Terminal;
        // das JCRE wird den Response-Code für normale Beendigung
        // „9000" zurückliefern.
    return;
} // end of deposit method
```



## Debit-Methode:

```
private void debit(APDU apdu) {
    if ( ! pin.isValidated() )
       ISOException.throwIt(ISO.SW_PIN_REQUIRED);
    byte numBytes = (byte)(buffer[ISO.OFFSET_LC]);
    byte byteRead =
       (byte)(apdu.setIncomingAndReceive());
    if (byteRead != 1 || numBytes != 1)
       ISOException.throwIt(ISO.SW_WRONG_LENGTH);
        // Test ob Geldbörse negativ wird:
  if ( ( balance - buffer[ISO.OFFSET_CDATA]) < 0 )
       ISOException.throwIt(SW_NEGATIVE_BALANCE);
        // Kommando ausführen:
    balance = (byte)(balance -
       buffer[ISO.OFFSET_CDATA]);
} // end of debit method
```

## getBalance-Methode

```
private void getBalance(APDU apdu) {
  if ( ! pin.isValidated() )
     ISOException.throwIt(ISO.SW_PIN_REQUIRED);

     // Informiere JCRE daß eine Response-APDU konstruiert wird
  apdu.setOutgoing();
     // Länge der Response-APDU
  apdu.setOutgoingLength((byte)1);
     // Schreibe Wert in den APDU-Buffer
  buffer[0] = balance;
     // Sende ein Byte mit Offset 0 an das Terminal:
  apdu.sendBytes((short)0, (short)1);
} // end of getBalance method
```



**Class** `javacard.framework.APDU`
**Method:** `public short setOutgoing() throws APDUException`

This method is used to set the data transfer direction to outbound and to obtain the expected length of response (`Le`).

**Notes.**
Any remaining incoming data will be discarded.
In T=0 (Case 4) protocol, this method will return 256.

---

**Returns:** `le`, the expected length of response.

**Throws:** `APDUException` with the following reason codes:
`APDUException.ILLEGAL_USE` if this method or `setOutgoingNoChaining()` method already invoked.
`APDUException.IO_ERROR` on I/O error.

**Class** `javacard.framework.APDU`

**Method:** `public void setOutgoingLength(short len) throws APDUException`

Sets the actual length of response data. Default is 0.

**Note:**
In T=0 (Case 2&4) protocol, the length is used by the JCRE to prompt the CAD for GET RESPONSE commands.

**Parameters:** `len` - the length of response data.

**Throws:** `APDUException`
with the following reason codes:
`APDUException.ILLEGAL_USE` if `setOutgoing()` not called or this method already invoked.
`APDUException.BAD_LENGTH` if `len` is greater than 256.
`APDUException.IO_ERROR` on I/O error.



**Class** `javacard.framework.APDU`

**Method:** `public void sendBytes(short bOff, short len) throws APDUException`

Sends `len` more bytes from APDU buffer at specified offset `bOff`.

**Notes:**
In T=0 protocol, if this method throws an `APDUException` with NO_T0_GETRESPONSE reason code, the JCRE will restart APDU command processing using the newly received command. No more output data can be transmitted. No error status response can be returned.

**Parameters:**
`bOff` - the offset into APDU buffer.
`len` - the length of the data in bytes to send.

**Throws:** ...

# Validate-Methode

```
private void validate(APDU apdu) {
        // Lese die PIN aus der APDU
    byte byteRead =
      (byte)(apdu.setIncomingAndReceive());

        // pin.check() testet die PIN und wirft entsprechende
        // Exceptions bei Fehlern
    pin.check(buffer, ISO.OFFSET_CDATA, byteRead);

  } // end of validate method

} // end of class Wallet
```



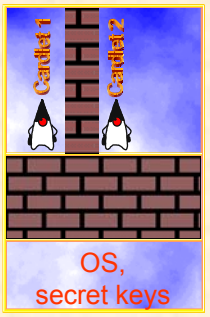
# Sicherheitsaspekte von Java-Karten

☺ Der Java-Interpreter bildet einen Sicherheits-Puffer:

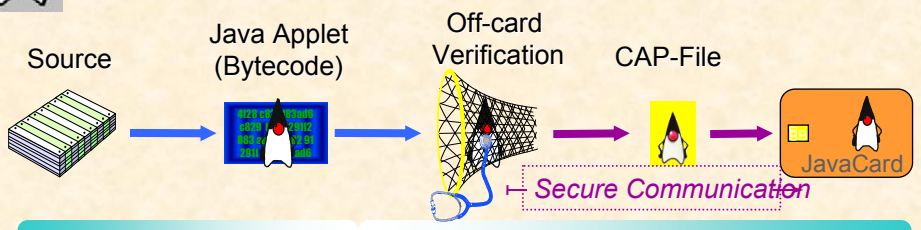
- zwischen Anwendungen und Plattform (“Sandbox”)
- zwischen den einzelnen Anwendungen (“Firewall”)

😐 Die austauschbare Software öffnet zwar ein neues “Tor” zur Karte, erlaubt aber auch Bug-Fixes und Software-Updates

☹ Das Gesamtsystem wird deutlich komplizierter (betrifft Karten und Karten-Management)



# JavaCard Security: a Bird's Eye View




Application Developer

Card Manufacturer

Card Issuer, Service Provider

Correctness of Application

Verification of Bytecode Properties

Correctness of Java-VM and APIs



# Smartcards

Einführung

Kommunikation mit der Karte

Java Smartcards

Smartcard-Sicherheit

## Problemstellung

**Karten dienen...**

- ...zur sicheren Aufbewahrung von geheimen Schlüsseln
- ...zur sicheren Durchführung von Berechnungen mit diesen Schlüsseln
- ...als sichere Ausführungsplattform (derzeit haupts. GSM)

**Forderungen:**

- Die Daten der Karte dürfen nicht auslesbar sein
- Die Berechnungen in der Karte dürfen nicht beobachtbar sein

**Problem:**

- Es gibt keine absolute Sicherheit
- Sicherheit ist teuer, also Kosten/Nutzen-Abwägung



## Klassifikation von Angreifern

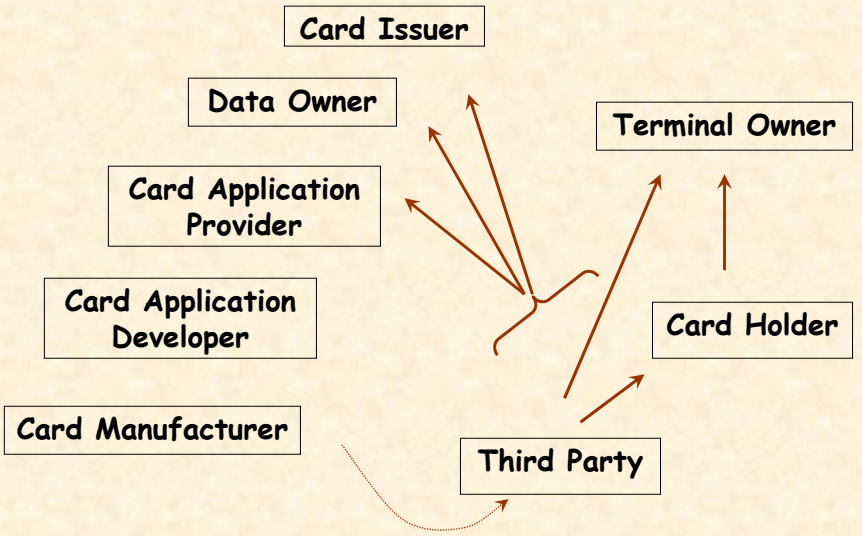
Class I: "Clever Outsiders". Intelligent, besitzen Fachkenntnisse, Zugriff auf bescheidene Ausrüstung, nutzen bestehende Schwächen von Systemen.

Class II: "Knowledgeable Insiders". Technische Ausbildung und Erfahrung, unterschiedliches Wissen über Systeme, möglicherweise Zugriff auf Spezialausrüstung.

Class III: "Funded Organizations". Spezialisten-Teams, Zugriff auf relevante (Insider-) Informationen, Spezialausrüstung, Zugriff auf Geld und Personalressourcen

[Nach: Abraham, Dolan, Double, Stevens: Transaction Security System, IBM Systems Journal, Vol. 30, No. 2, 1991.]

## Player im Sicherheits-"Spiel"






## Beispiele für Typen von Angriffen

**Terminal -> Card Holder/Issuer (Data-Owner)**

- Manipulation der zur Karte übertragenen Daten

**Card Holder -> Terminal**

- Simulation einer Karte durch Angreifer

**Card Holder -> Card Issuer/Data Owner**

- Kartenmanipulation, Re-Engineering, DPA,...

**Card Issuer -> Card Holder**

- Meist Privacy-bezogen

## Aktive Komponenten auf Hardware-Ebene

**Beispiele:**

- Sensoren, die auf Licht reagieren
- Wärme-Sensoren gegen Übertaktung des Chips
- Widerstands- oder Kapazitätsmessung um zu erkennen, ob die Schutzschicht über der Schaltung noch vorhanden ist.
- Überwachung von Spannung/CLK-Frequenz, um Angriffe auf der "elektrischen" Ebene zu erkennen
- Funktionstest bestimmter Teile des Chips

**Generelles Problem:**

- Karten haben keine eigene Stromversorgung, aktive Komponenten funktionieren nur, wenn die Karte mit Spannung versorgt ist



## Passive Hardware-Komponenten

- Irreversible Konversion von Test- zu User-Mode bei der Chip-Fertigung
- "Scrambling" des Datenbusses
- Sicherstellung eines in etwa identischen Stromverbrauchs bei allen Chip-Instruktionen
- ROM ist im Prinzip optisch auslesbar, sollte daher schwer zugänglich sein
- ...

## Software: Betriebssystem-Ebene

- Generell: Sauberes, robustes Design, klare Strukturierung, usw.
- Checksummen der "kritischen" Inhalte in ROM und EEPROM bilden und prüfen
- Test des RAMs, da dort Informationen für die Zugriffskontrolle zur Laufzeit liegt (Lesefehler!)
- Speicherung von Teilen der kritischen Daten und des Betriebssystems im EEPROM, um dem Chiphersteller nicht alle sensiblen Information geben zu müssen.
- Beachten der Tatsache, daß das Schreiben ins EEPROM die Stromaufnahme erhöht (z.B. kann dadurch u.U. der Zeitpunkt des PIN-Vergleichs erkannt werden)
- Möglichkeit zum kompletten "Stilllegen" der Karte muß vorhanden sein



## Software: Anwendungsebene

- Wieder: Sauberes, robustes Design, klare Strukturierung, usw.
- Kompromittierung einer Anwendung sollte nicht die Kompromittierung des Gesamtsystems nach sich ziehen
- Überwachung der Kommunikation mit endlichem Automaten, um trial-and-error Angriffe zu erkennen
- Anlegen von Log-Files um Informationen über den "Power-up" retten zu können
- Sensible Daten verschlüsselt speichern

## Simpler PIN-Angriff

**Idee:**

- Gegeben: Karte mit bekannter PIN
- Beobachten der Stromaufnahme bei Eingabe von richtiger und falscher PIN, finden von Charakteristika
- Systematisches Ausprobieren von PINs, RESET wenn falsche PIN erkennbar (vor dem Blockieren der Karte)

**Abhilfe:**

- Vermeidung solcher charakteristischer Verhaltensweisen
- Erhöhung des (persistenten) Zählers vor PIN-Prüfung



## "Signal Glitches" (Markus Kuhn)

**Mögliche Ausgabe-Routine für Daten:**

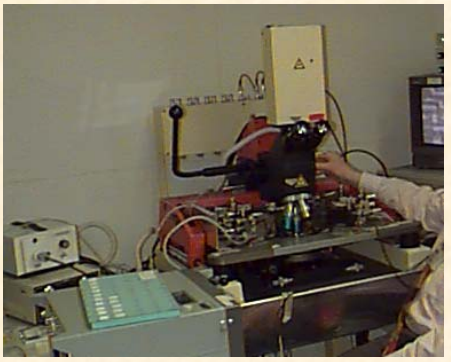
```
1    b = answer_ address
2    a = answer_ length
3    if (a == 0) goto 8
4    transmit(* b)
5    b = b + 1
6    a = a - 1
7    goto 3
8    ...
```

**Manipulation von CLK oder Vcc beim Instruktions-Fetch von 3 oder 6 kann zur Ausgabe von mehr Daten führen**

## Die Ausrüstung...

Einfaches Reinraum-Labor
(z.B. an Unis zu finden)

Gängiger Spitzenmeßplatz
(Micro-Prober)

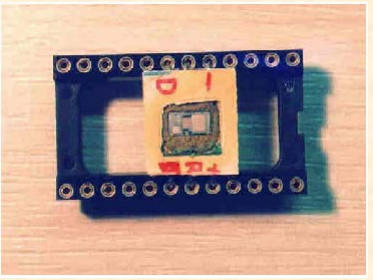




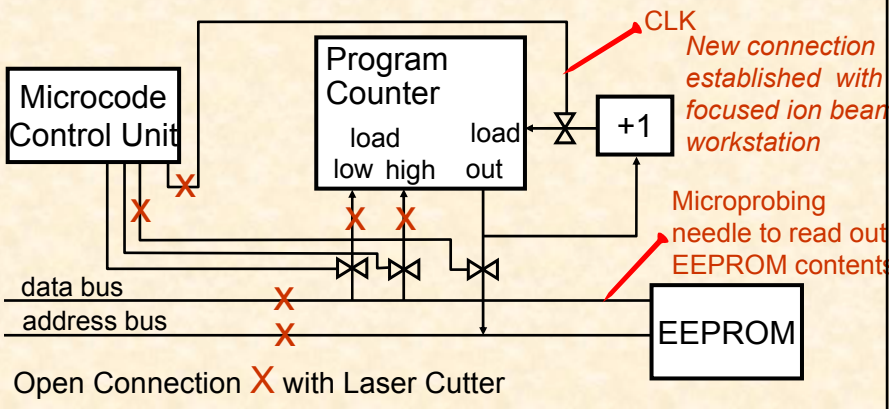
## Chipcard Hacking: Basics (nach: M. Kuhn)

- Carefully remove the chip's covering manually
- *Repeat until surface is fully exposed:*
  - Apply fuming nitric acid (> 98% HNO ) on remaining plastic
  - Remove acid and dissolved plastic with acetone

*Hint:*
*IR radiation accelerates etching*



## Accessing the EEPROM... (nach: M. Kuhn)



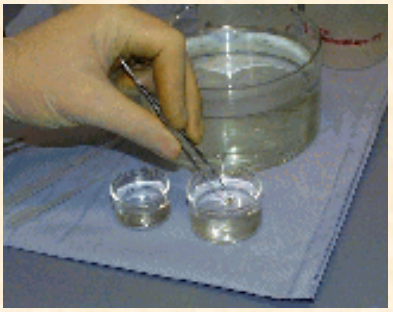



## Source: http://www.cl.cam.ac.uk/~mgk25/sc99-tamper-s

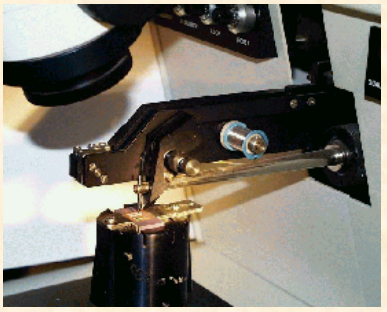
**Heat up card plastic, bend it, and remove chip module**

**Dissolve package in 60 °C fuming nitric acid,...**

Source: http://www.cl.cam.ac.uk/~mgk25/sc99-tamper-s



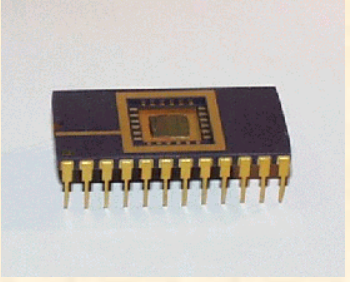
**...then wash in acetone, deionized water, and finally isopropanol.**



**A manual bonding station establishes reliable contacts to the supply, communication, and test pads of the microprocessor using ultrasonic welding of a fine aluminium wire.**

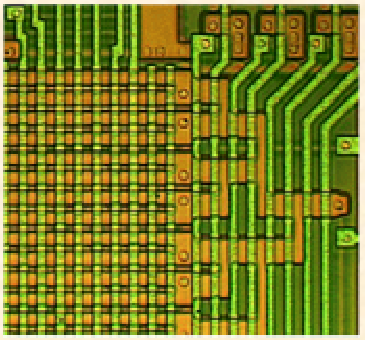


Source: http://www.cl.cam.ac.uk/~mgk25/sc99-tamper-s

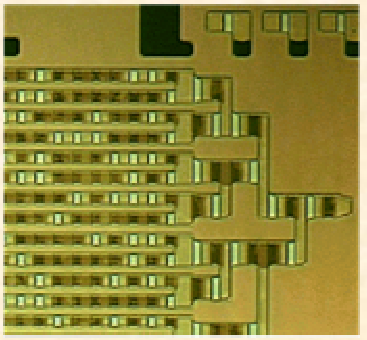


Source: http://www.cl.cam.ac.uk/~mgk25/sc99-tamper-s

## Optical Reconstruction of Ion Implantation ROM Content



View of ROM with polysilicon intact



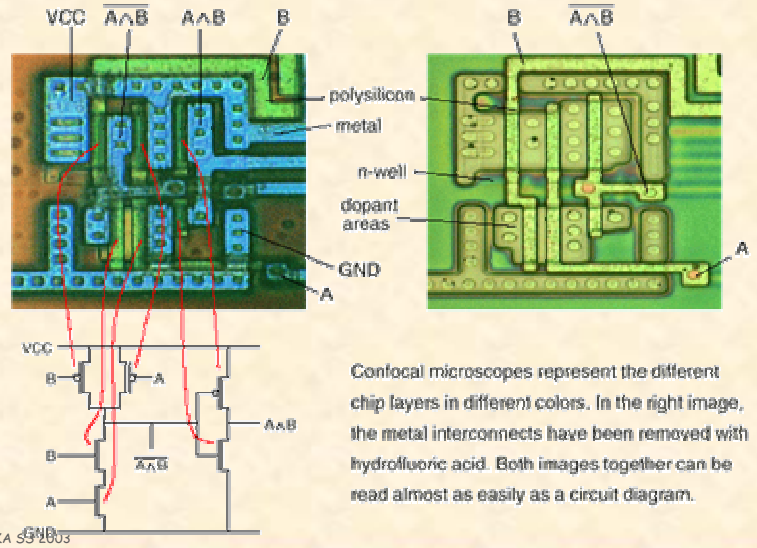
Diffusion layer after crystallographic etch

This type of ROM does not reveal the bit pattern in the shape of the diffusion areas, but a crystallographic staining technique (Dash etchand) that etches doped regions faster than undoped regions will still show the ROM bits.



Source: http://www.cl.cam.ac.uk/~mgk25/sc99-tamper-s

## Optical Reverse-Engineering of VLSI Circuits




Confocal microscopes represent the different chip layers in different colors. In the right image, the metal interconnects have been removed with hydrofluoric acid. Both images together can be read almost as easily as a circuit diagram.

# Differential Power Analysis (DPA)

**Paul Kocher, Joshua Jaffe, Benjamin Jun**
Cryptography Research (www.cryptography.com) 1998.

**"Introduction to Differential Power Analysis and Related Attacks", Zitat:**

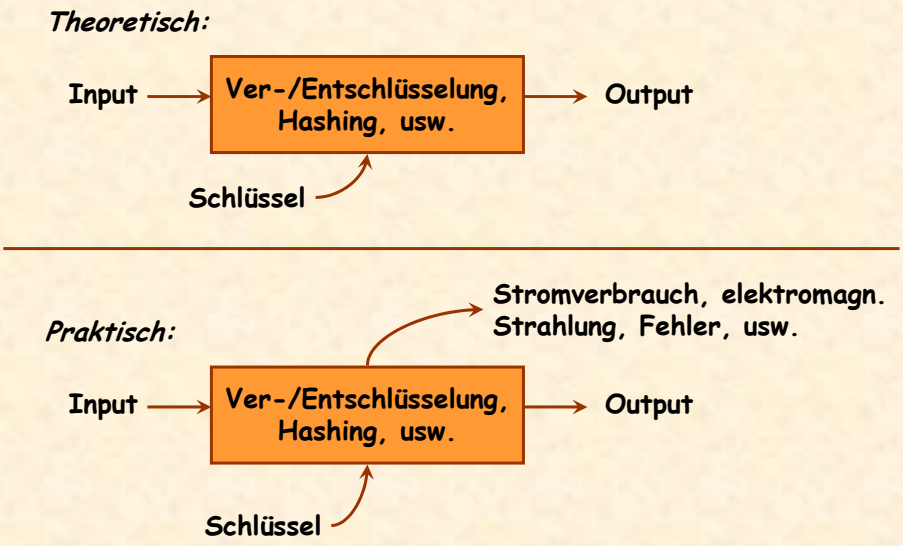
[...] we have been analyzing how to improve security of portable cryptographic tokens, including smart cards [and] have been working with the smart card vendor community to address attacks we have developed.

These are not theoretical attacks. Cryptography Research has successfully used these attacks to analyze a large number of smart card products. [...] we have not found any commercially-available products that resist DPA.

It is expected that a few systems currently being engineered using techniques licensed by Cryptography Research will be resistant to power analysis attacks.



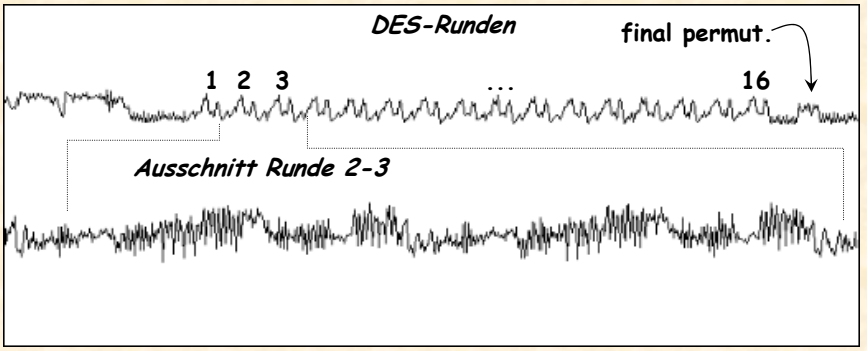
# Krypto-basierte Systeme: Meta-Level Perspektive



## Grundidee der Power Analysis

**Beobachte den Stromverbrauch von Chipkarten bei der Ausführung von Verschlüsselungsalgorithmen.**

**Bspl: DES**






## Moral...

**Chipkarten sind lediglich ein relativ sicherer Aufbewahrungsort für Geheimnisse.**

**Ein Gesamtsystem muß so gestaltet sein, daß die Kompromittierung einer Komponente nicht das gesamte System kompromittiert.**

**Unabdingbare Voraussetzungen:**

- Viel Erfahrung,
- Expertise,
- kritische Analysen durch Dritte,
- Mißtrauen gegenüber Behauptungen von Herstellern

## "Security by Obscurity"

**These:**

**"Nur öffentlich bekannte Sicherheitsverfahren und -algorithmen sind sicher."**

**Pro:**

- Schwächen werden erkannt und behoben
- Insider-Wissen wird weniger hilfreich für Angriffe

**Contra:**

- Es ist unklar, **wann** ein Verfahren als hinreichend sicher anzusehen ist.
- Neu bekannt gewordene Schwächen müssen **schnell** behoben werden (Durchführbarkeit? Kosten?)

**Kocher-Atacke als Gegenbeispiel für die These?**



## Industrielle Praxis

**Aufgabe: Entwickle Authentizierungsalg. f. neues Produkt, Einsatz in 6 Monaten**

- Einsatz eines Standardalgorithmus oft nicht möglich wg. Randbedingungen (technische und rechtliche)
- Latenzzeit bei der Veröffentlichung einer Eigenentwicklung wäre zu hoch
- Implementierungen in Produkten sind of schwer austauschbar
  (Bspl: Austausch von ca. 2 Mio GSM SIMs dauert Monate und kostet leicht mehrere 100 Mio €)

**Wichtig: "Verfallsdatum" bei proprietären Verfahren!**