CONTACTLESS SMART CARD AND JAVA CARD

Syndicate

- 1. Maj Chetan Dewan
- 2. Deepak Shinde
- 3. Ravikant Kr Nirala
- 4. Sudheer Kr

Guide

- Prof. Bernard Menezes

(Dept of Information Technology)

IIT Bombay

Coverage

Contactless Smart cards

- Introduction
- Architecture
- Comparison to RFID
- Functioning

Java card

- Introduction
- Architecture
- Security
- Attacks
- Future card
- Comparison
- Standards

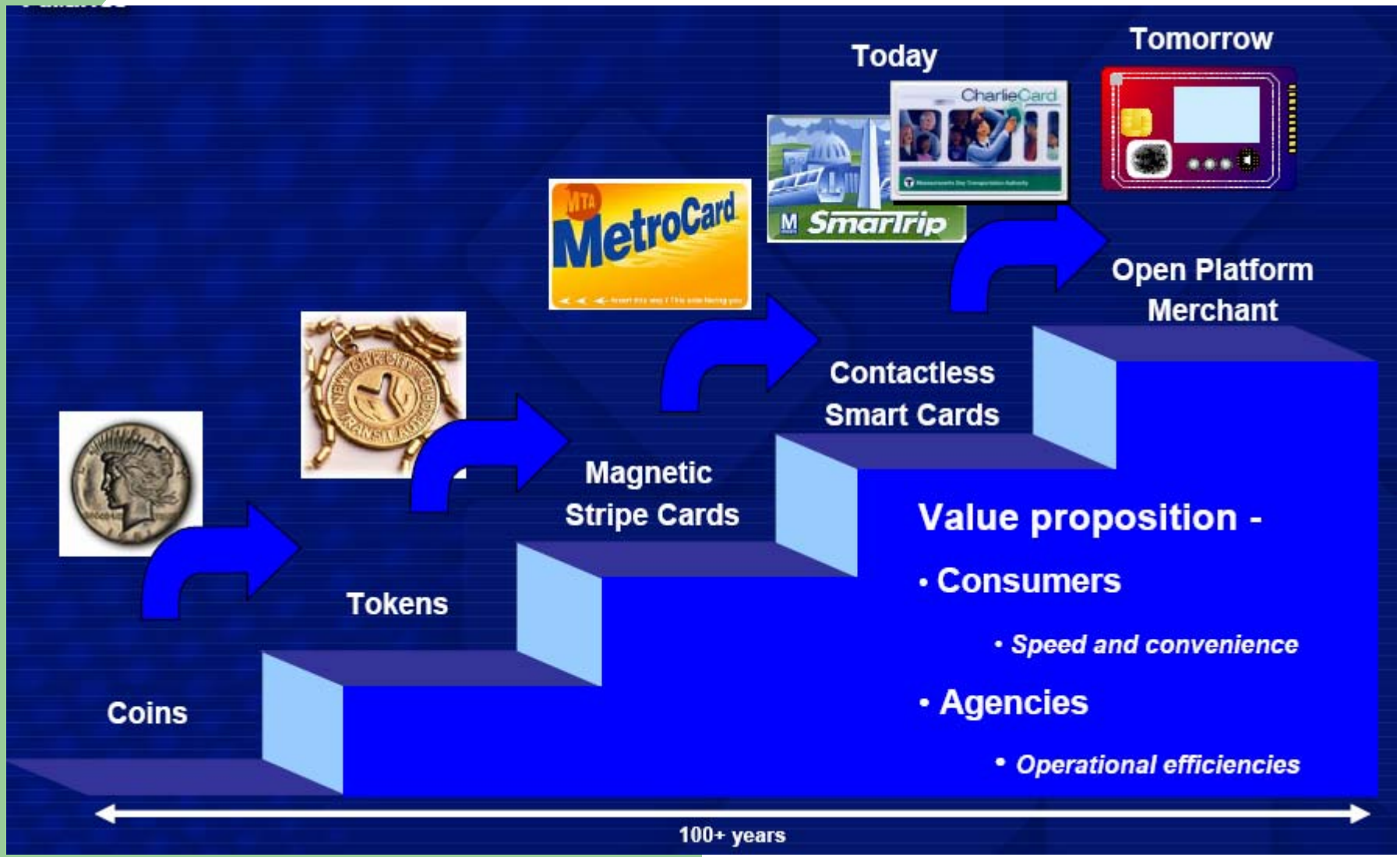
Questions ???????

INTRODUCTION

What are Smart Cards?

- A simple processor embedded in a plastic card
  - Same size as a credit card
- New technology allows multiple applications on the same card
- Useful for hundreds of applications
  - Debit, credit, cash
  - Identity, cryptography

Evolution

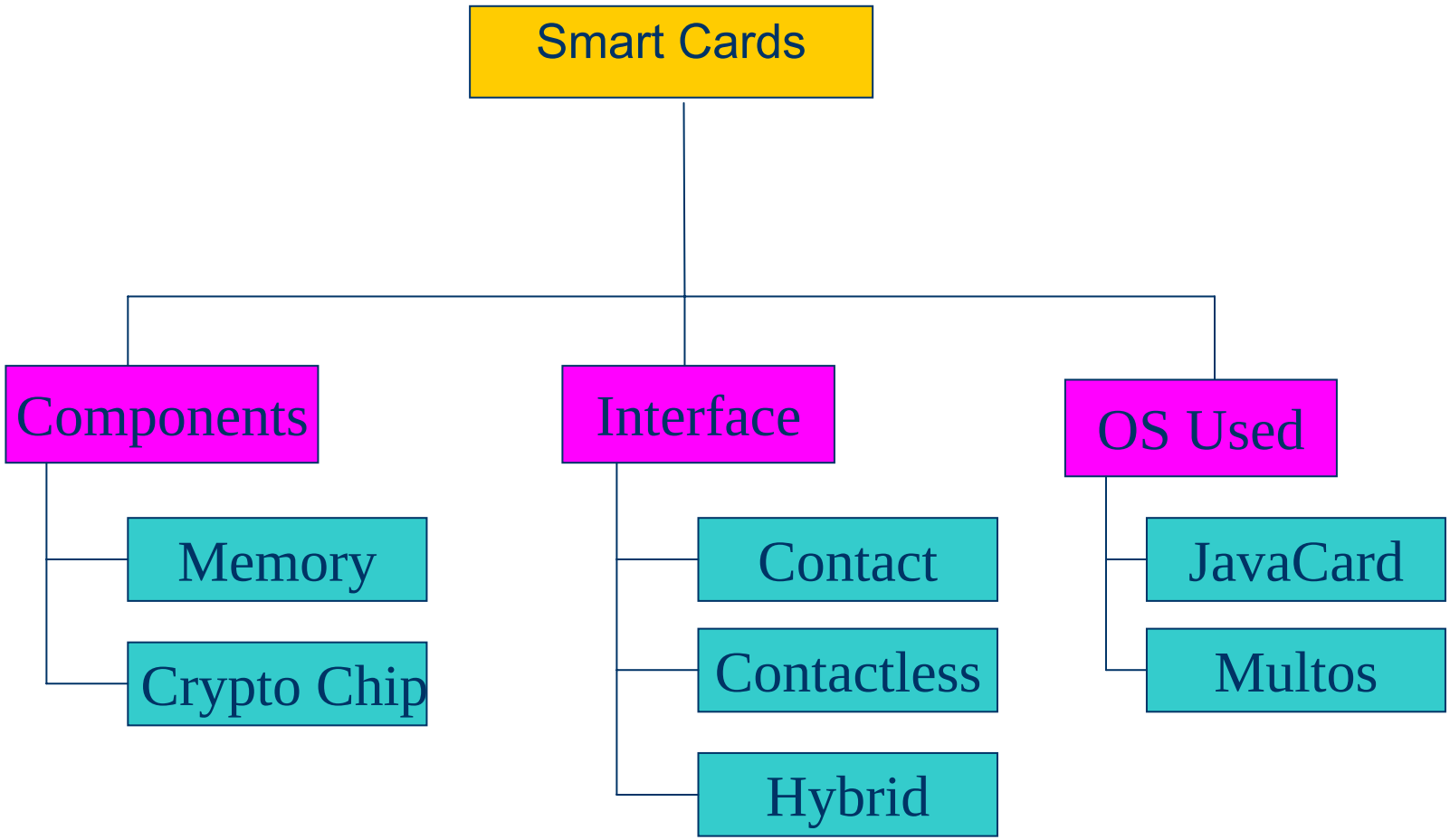

Today
Tomorrow
CharlieCard
MTA
MetroCard
SmarTrip
Open Platform
Merchant
Contactless
Smart Cards
Magnetic
Stripe Cards
Tokens
Coins
Value proposition -
• Consumers
• Speed and convenience
• Agencies
• Operational efficiencies
100+ years

Small Comparison

- ***Current Smart Chip technology***
  - 16-64 MHz CPU
  - 8-32 bit architecture
  - 4-72k EEPROM
  - 1-8k RAM
- ***Compare to: Apollo 11 Guidance Computer***
  - 2 MHz CPU
  - 16 bits word length
  - 4KB RAM
  - 72KB ROM

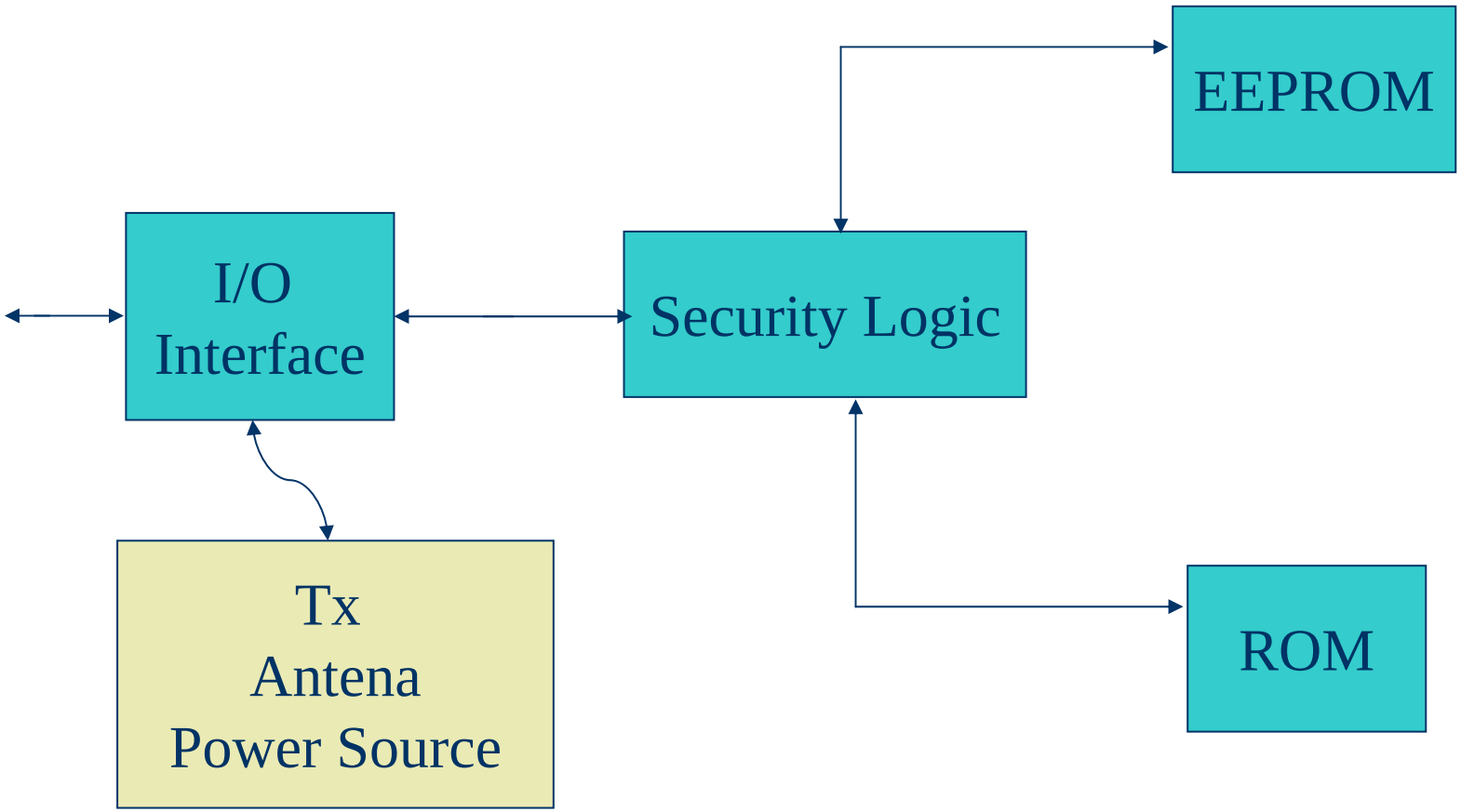
- *www.apollosaturn.com/gnc.htm*
- *www.digitalmist.com/plethorama/apollogc.htm*

Smart Card Classification



ARCHITECTURE

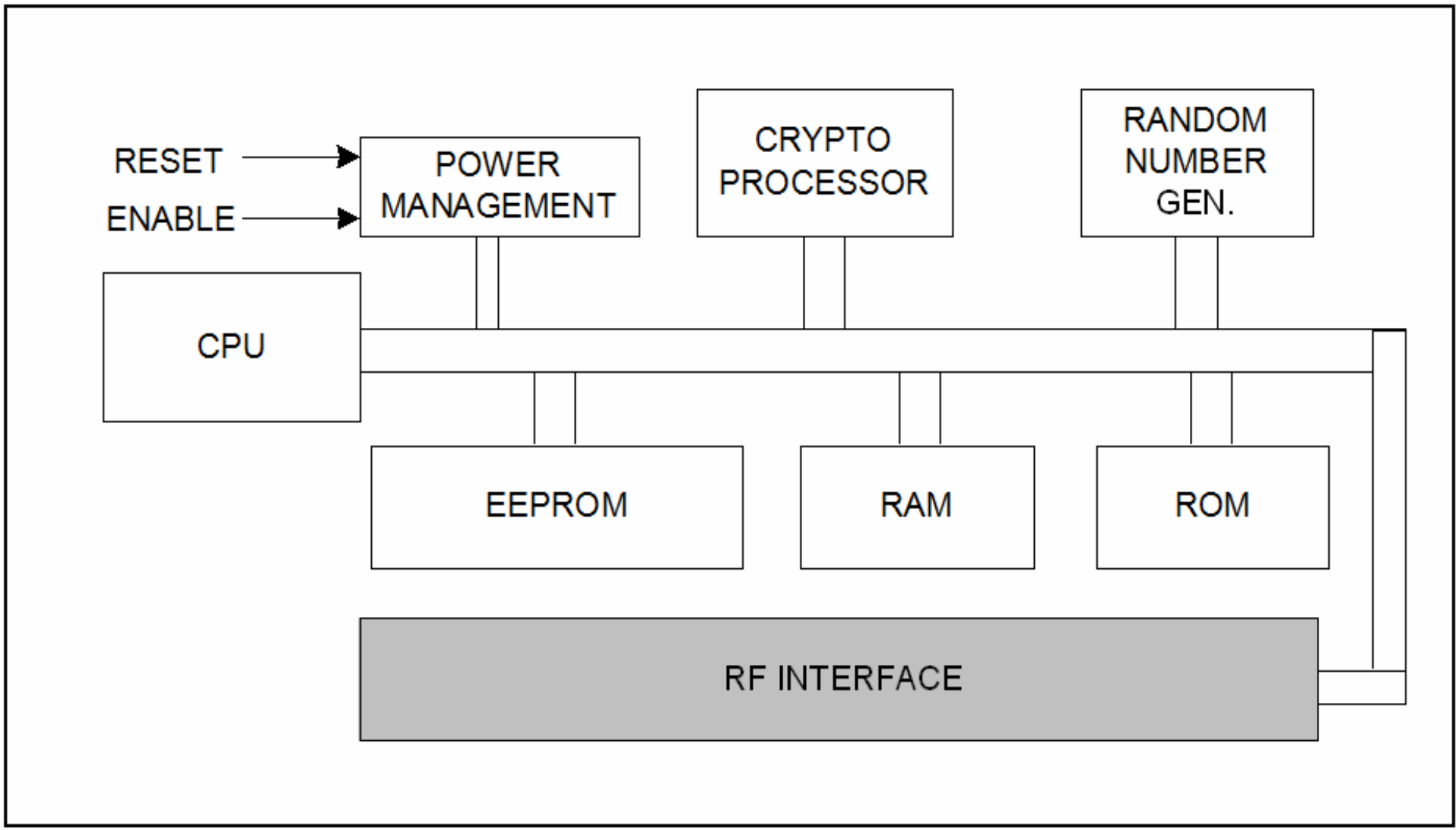
Memory Card Architecture



Memory Cards

- Most common and the cheapest.
- Contain EEPROM and ROM.
  - ROM holds card number, card holder name.
  - EEPROM holds data that changes with time, usually application data. E.g. in pre-paid phone card, it holds talk time left.
  - EEPROM can be locked with a PIN.
- Cost Rs 50
- Areas where used: Pre paid telephone cards, parking schemes, ticketing, vending machines

CPU Smart Card Architecture



CPU Smart Card

- Contains microprocessor.
- Various parts:
  - **ROM**: Also called the Mask of the card. Holds the Operating System.
  - **EEPROM**: Holds the application programs and their data and various keys used
  - **PROM**: Holds the card number.
  - **RAM**: Used as temporary storage space for variables.
  - **Crypto Processor**: 8-32 bit processor based on CISC architecture. Crypto processor in a FPGA device.
  - **Random No Generator**: DES
  - I/O Interface for data transfer to and from the card.

CPU Smart Card

- Cost: Rs 100/-
- Can house multiple applications.
- Provide robust security. **Very difficult** to crack.
- Areas used: Financial Cards, Electronic Purses, Access Control, Travel and Ticketing.

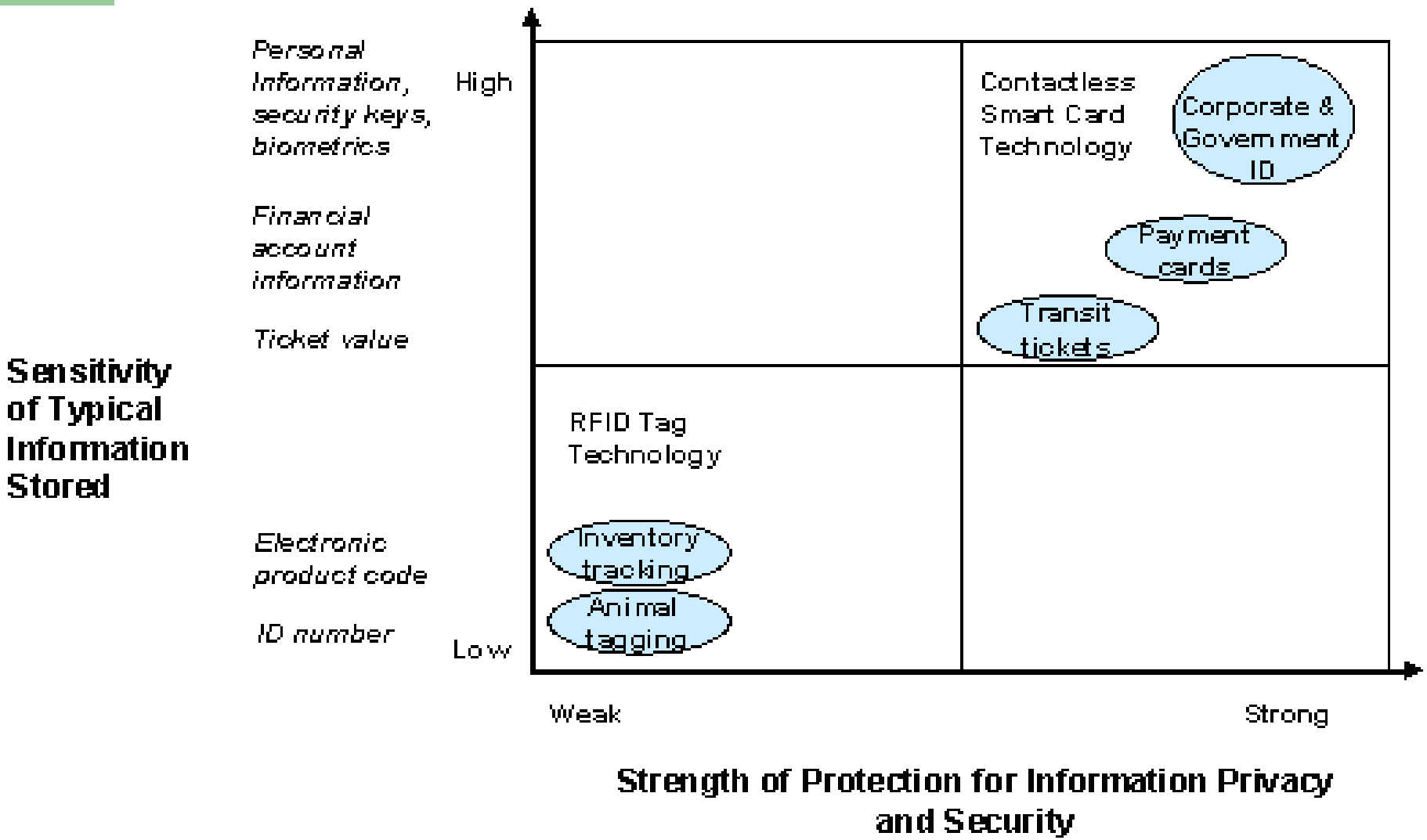
RFID Vs Contactless Smart Card

Comparison

- **Low cost, high volume**
- **Minimal security**: Tags read by any compatible reader
- **Disposable** or one-time use
- **Minimal data storage** comparable to bar code, usually a fixed format written once when the tag is manufactured
- **Read range optimized** to increase speed and utility

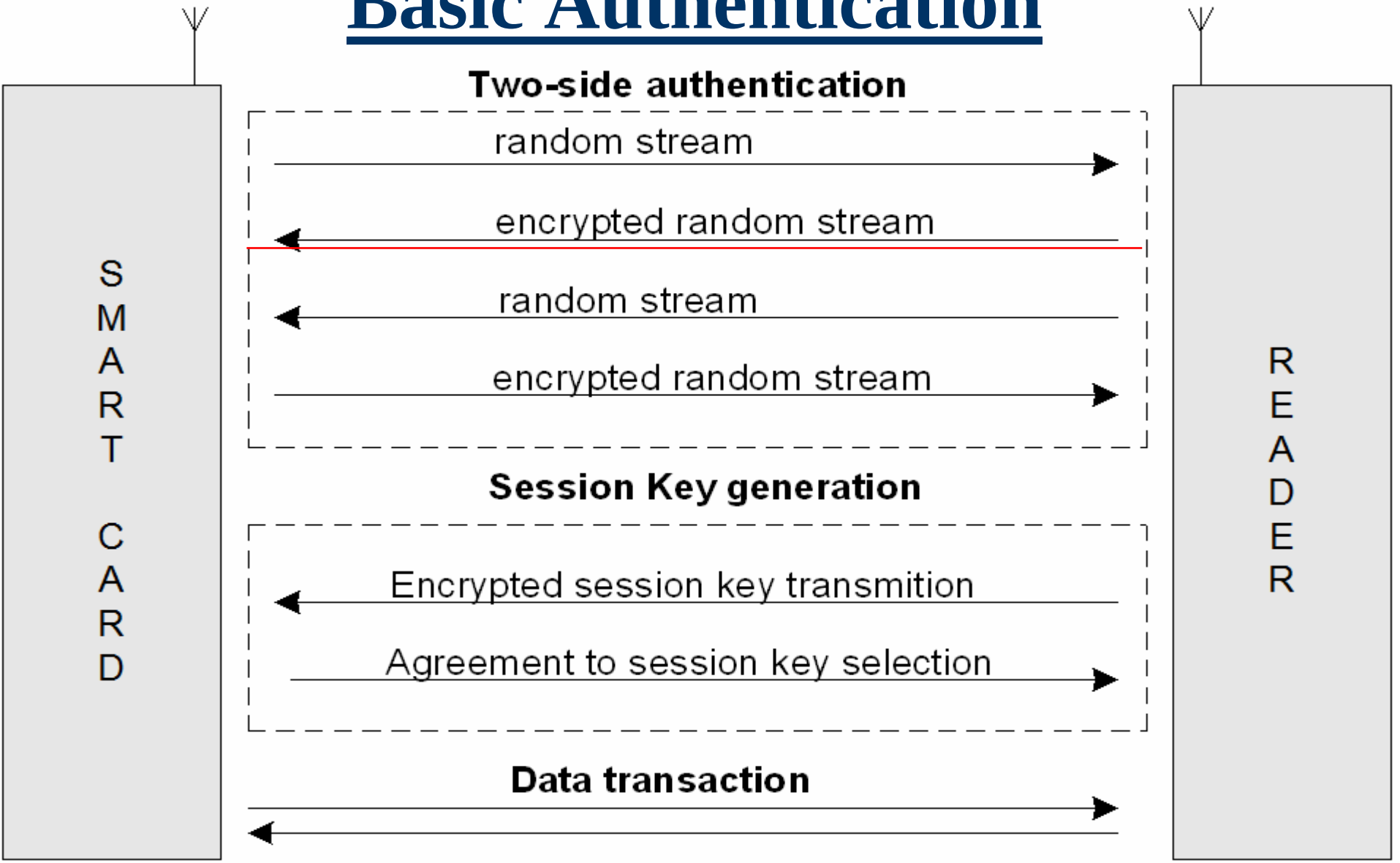
- **Mutual authentication.**
- **Strong information security.** Encrypted communication
- **Strong contactless device security.** Difficult to duplicate or forge, has built-in tamper-resistance.
- **Authenticated and authorized information access.** Personal identification number (PIN) or biometrics
- **Strong support for information privacy**. Personal firewall for an individual, releasing only info required.

Comparison RFID Tag Vs Contactless Smart Card Technologies



FUNCTIONING

Basic Authentication



Java Card

How Java and smart cards mix

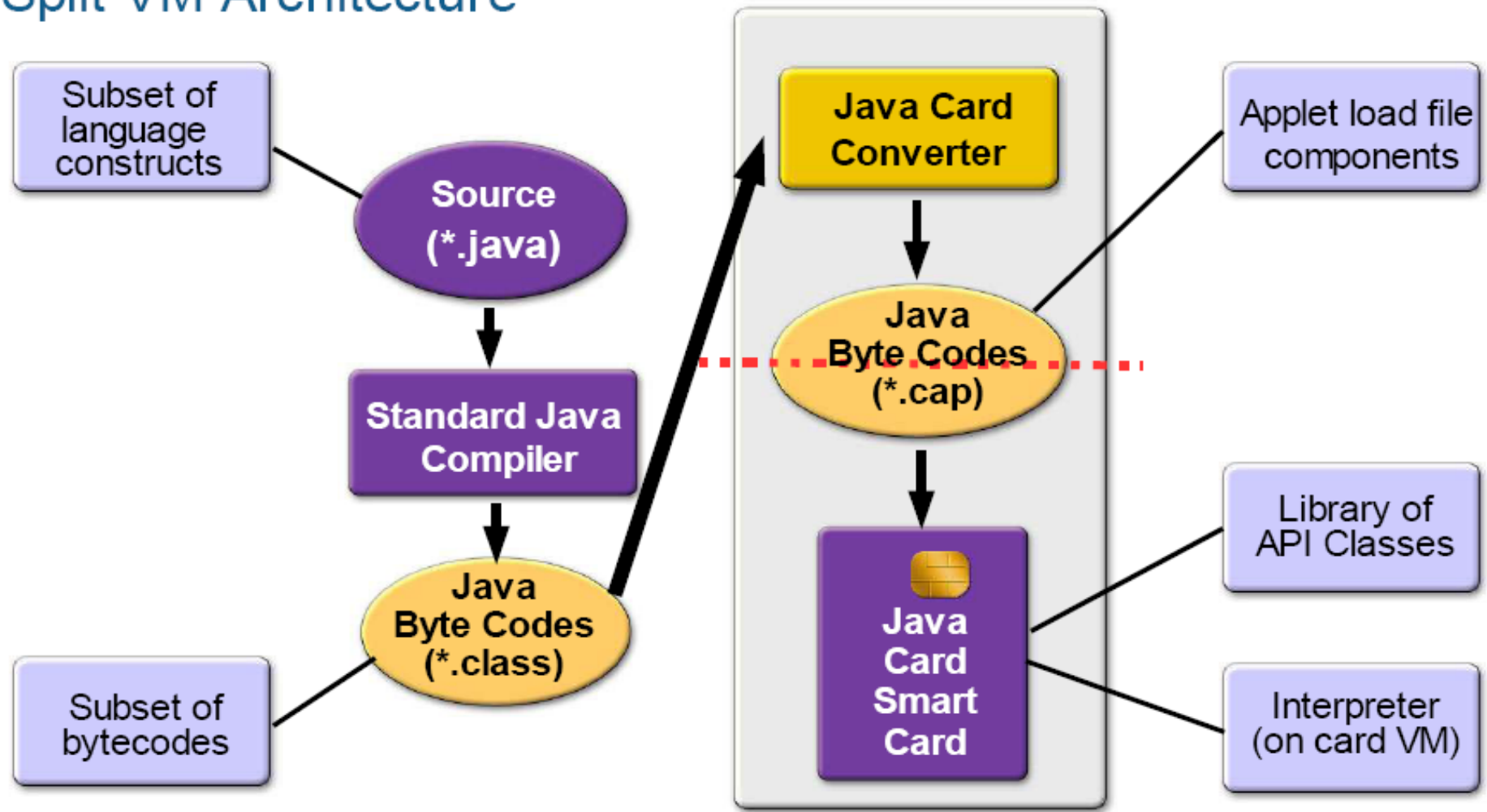
- Java Card is a stripped down version of Java for smart cards (split VM model)
- Well-designed object-oriented language
- Cross platform compatibility
- Tight integration with other Java systems
- Industry and tools support.
- Java Card makes multi-application cards based on a common platform possible
  - open up smart card development
  - use a real language

Java Card Architecture

- Java card virtual machine (JCVM)
- Java card runtime environment (JCRE)
- Java card application programming interface (API)

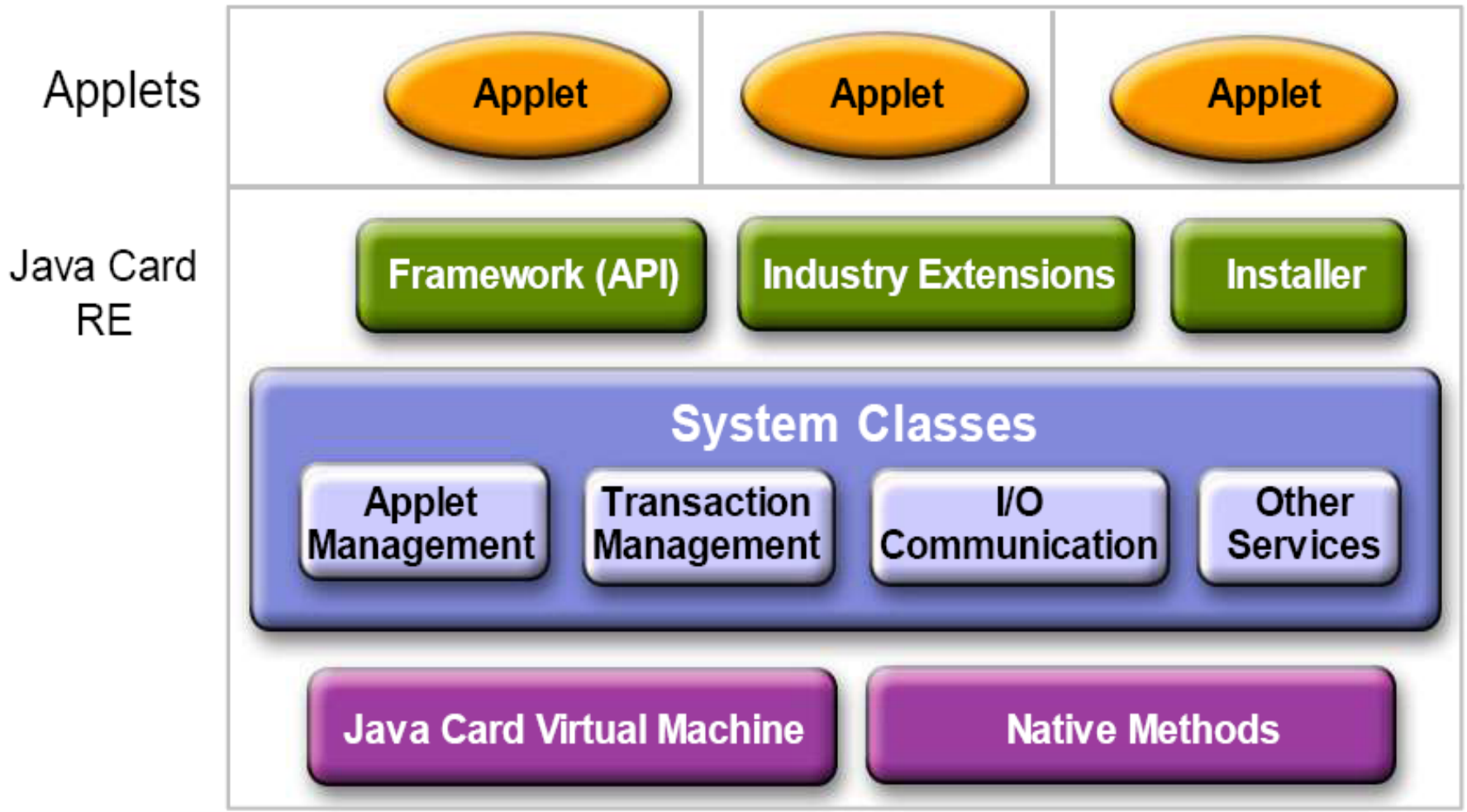
Java Card Technology

Split VM Architecture



Java Card Technology

Java Card API Stack

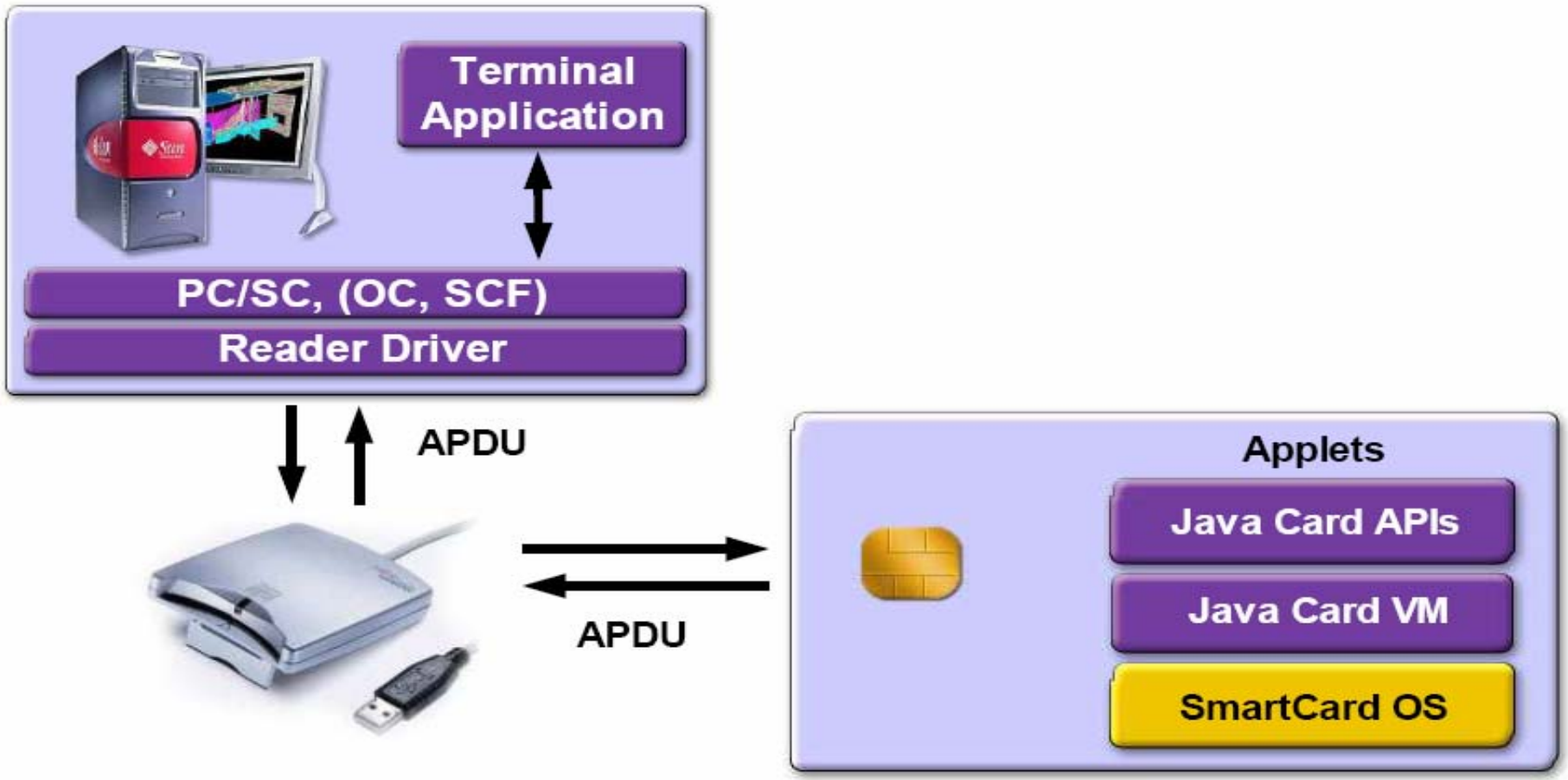


Java Card Technology

Typical Desktop Infrastructure

- Card reader
  - Connected to desktop via USB
- Card reader driver layer
  - Customized for card reader
- PC/SC card access framework
  - OS infrastructure
- Secure client application
  - Depends on Java Card technology for security

Java Card Client Infrastructure

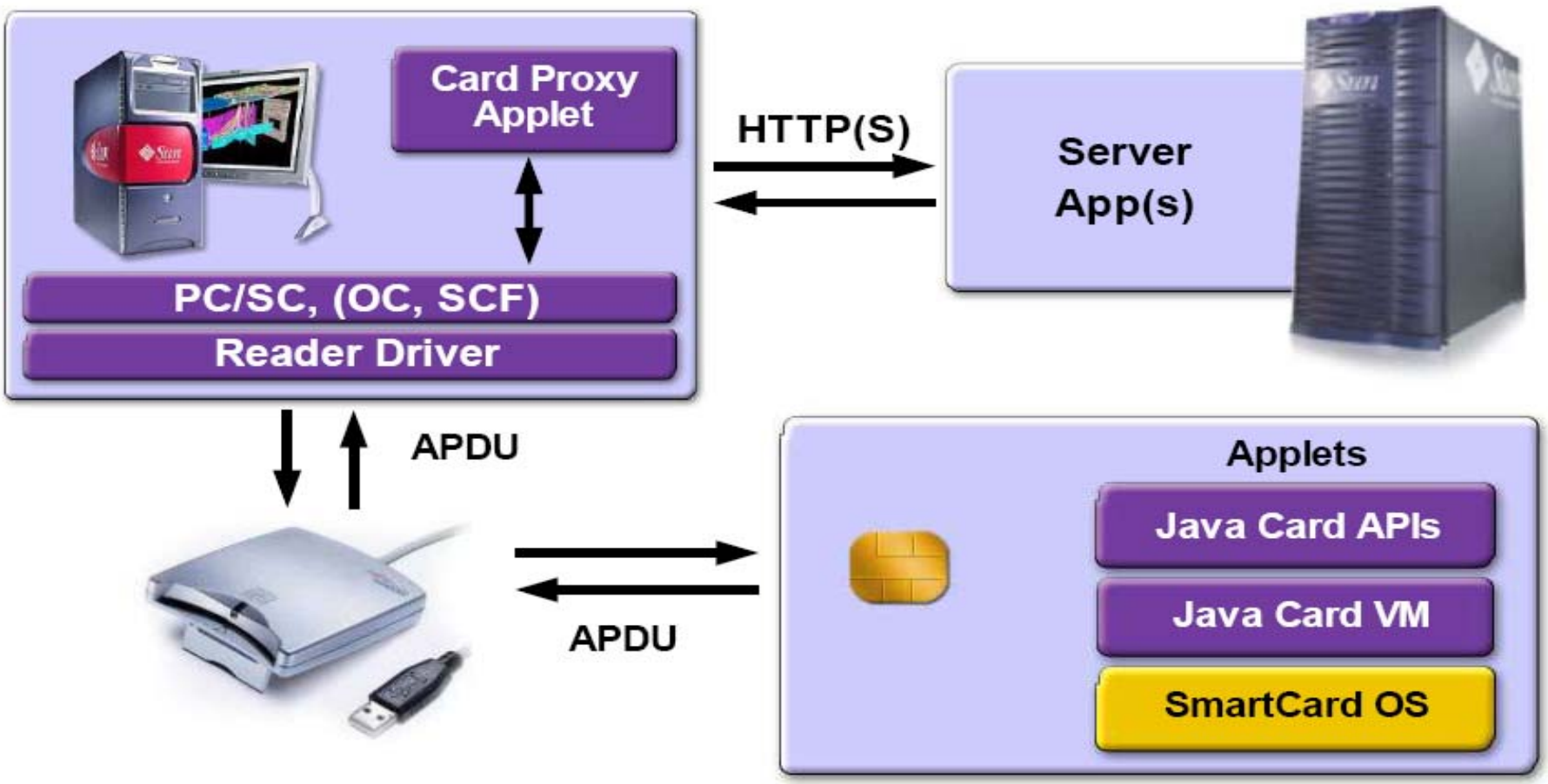


Server Components

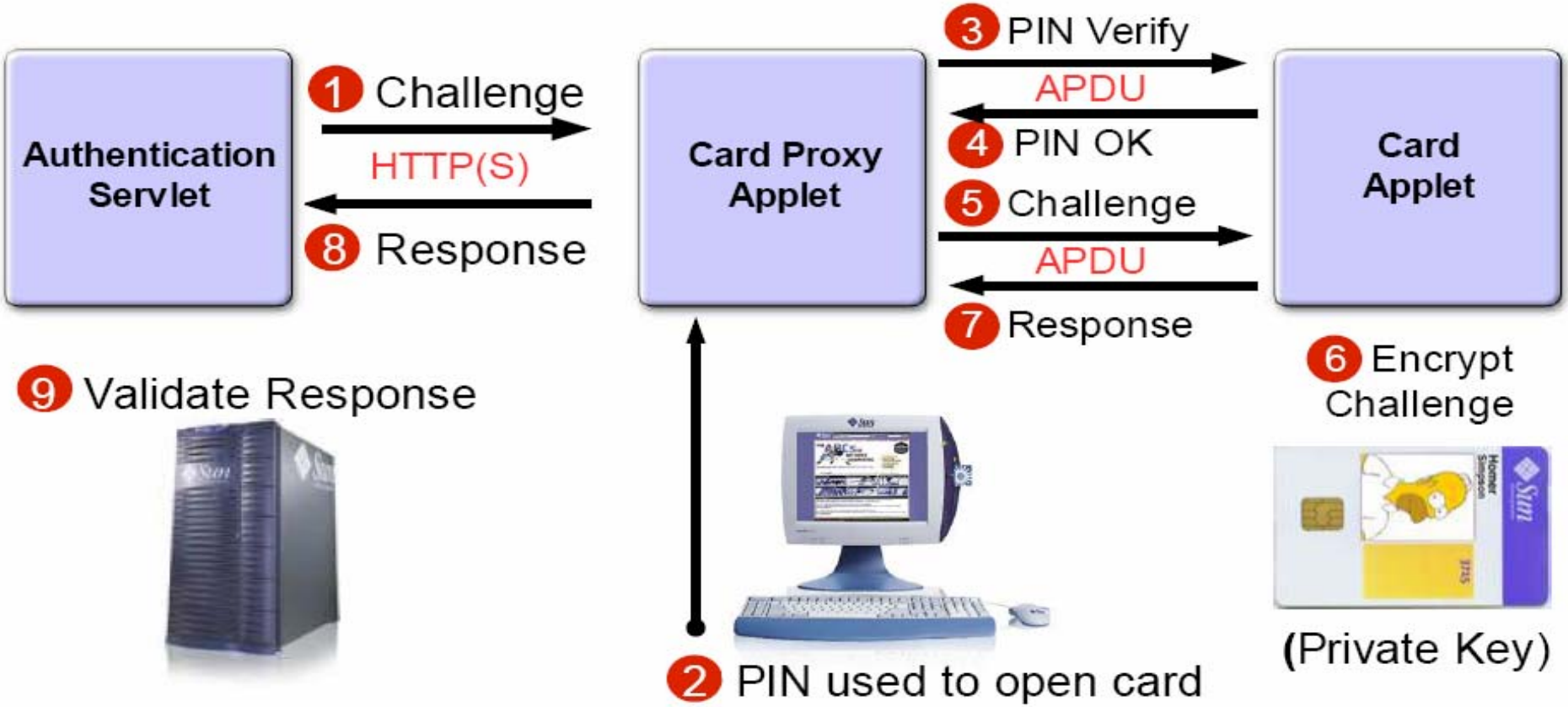
Additional Infrastructure Layers

- Card proxy middle-ware
  - Bridge between server application and Java Card platform
- Secure Server Application
  - Depends on Java Card technology for security

Java Card Client Infrastructure



Server Authentication Data Flow



Security

Java Card security != Java security

Good

- no dynamic class loading
  - type safety issues
- only one active applet
- no multithreading
- objects include rudimentary access control

Bad

- applets added post issuance
- no sandbox
  - trusted code required
- native method calls
- no garbage collection
- object sharing complexity

Other security risks in Java Card 2.x.x

- protocol interactions
  - sharing secrets between protocols introduces new problems
- security is hard
  - linking, export, CAP files
  - native methods
  - verification
  - object sharing
- multi-application risks
  - applets MUST behave
- the usual suspects apply
  - physical attacks
  - side-channel monitoring (DPA)
  - the terminal problem

Some multi-application issues

Secure Features

- no dynamic class loading
  - reduces threat of malicious applets
- no multi-threading
  - non-interference
- applet firewalls
  - prevents referencing another applet's objects

Risks and Assumptions

- trust-based applet model
  - assume applets are non-malicious
  - security testing
- JCRE must be perfect
  - prevents collusion
- more developers?!

Security is harder than it sounds

- Java Card is not truly "cross platform"
  - byte code → CAP
  - export files
- linking problems
  - no strings, thus ?
- code verification?
  - before conversion
- exception handling
- native methods not verified
- Applet testing and debugging issues
- sharing methods among applets (difficult)
- ISO 7816 APDU problems
- hostile applets
  - denial of service

Security Issues (by malicious applet)

- Leaking confidential information outside (eg: PINS and secret cryptographic keys)
- Modifying sensitive information (eg: balance of an electonic purse)
- Interfering with other honest application on the card (Trojan Attack)

Solution Sandboxing

- Execute the applets in a so called “Sandbox”
- Its an insulation layer preventing direct access to the hardware resources
- Also implements a suitable access control policy

Sandbox Model- 3 components

- Component 1: Applets executed by JVM
- Component 2: Java Card runtime environment provides access control through its “firewall”
- Component 3: Bytecode Verifier **missing!**
  - **complex and expensive process**
  - **requires large amounts of working memory**

Bytecode Verification- Purpose

- To check ->code is well typed
- Doesn’t bypass protections by performing ill typed operations at runtime:
  - Forging object references from integers
  - Illegal casting of obj ref from one class to another
  - Calling private methods of API
  - Jumping in middle of API method or jumping to data as if it were code

Approach

- Relying on off-card tools
- Attesting a cryptographic signature on well-typedness of the applet
- Oncard downloading restricted to signed applets
- Drawbacks:

  - to extend the trusted computing base to off card components

  - practical issues: (how to deploy the signature keys?)

  - legal issues: (who takes liability for a buggy applet?)

Defensive VM Approach

- To type-check dynamically during applet execution
- VM computes bytecode instructions
- Keeps track of all data it manipulates
- Arguments- correct types?stack overflow or underflow?
- Are class member accesses allowed?
- Drawback:

  - **Dynamic Type-checks expensive in terms of execution speed and memory**

Physical attacks still apply

- Physical attacks attempt to reverse engineer card or monitor a running card to obtain card secrets
  - Differential power analysis (Kocher)
  - No card is tamper proof (Anderson & Kuhn)
- Some secrets could be used to add functionality and/or add value
  - Cost of hacking the card must be greater than return on investment

The terminal problem

- No trusted interface for interacting with users
- A common solution is to use PCs
  - but PCs are easily hacked
  - windows inherently insecure

Protocol interaction risks

- Unintended protocol interactions pose risks:
  - secure protocols do not necessarily compose
  - different protocols share same key material
  - observation of protocol *P* can be used against *Q*
- Shared key material is motivated by:
  - digital certificates for multi-applications
  - small memory for public/private key pairs
  - crypto APIs

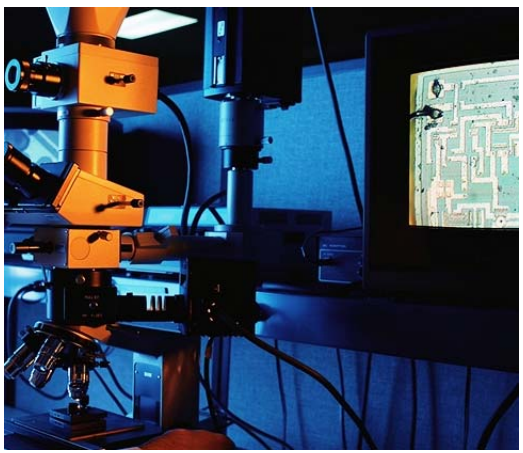
ATTACKS

Is Smart Card Secure?

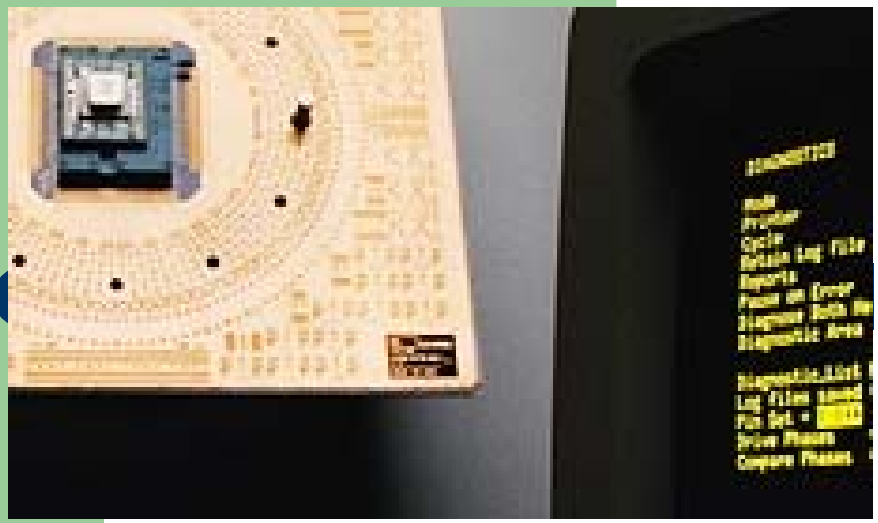
- There is no 100% secured/perfect system available
- System design and built for minimal attack risk can be treated as secure
- Secure system are evaluated/classified in different levels using international standards such as TCSEC/DoD (Orange Book-USA), ITSEC (Europe) and CCITSE (ISO15408)

Possible Attacks on Smart Card

- ***UV or X-ray inspection:*** use high efficiency UV or X-ray to inspect the memory areas to extract important information like PIN, secret key and public key

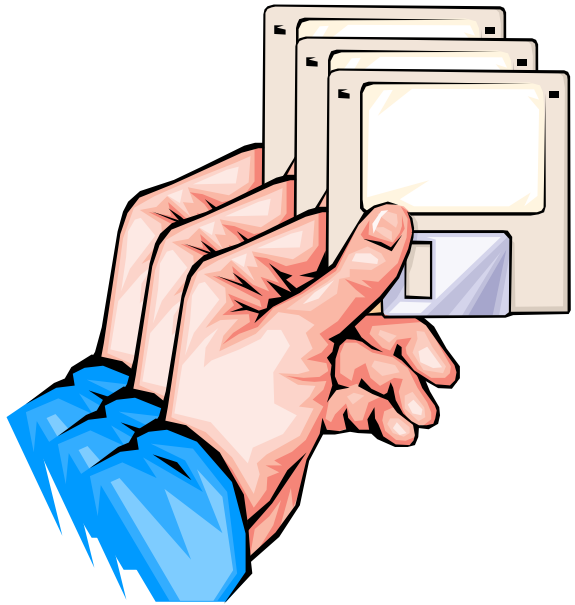


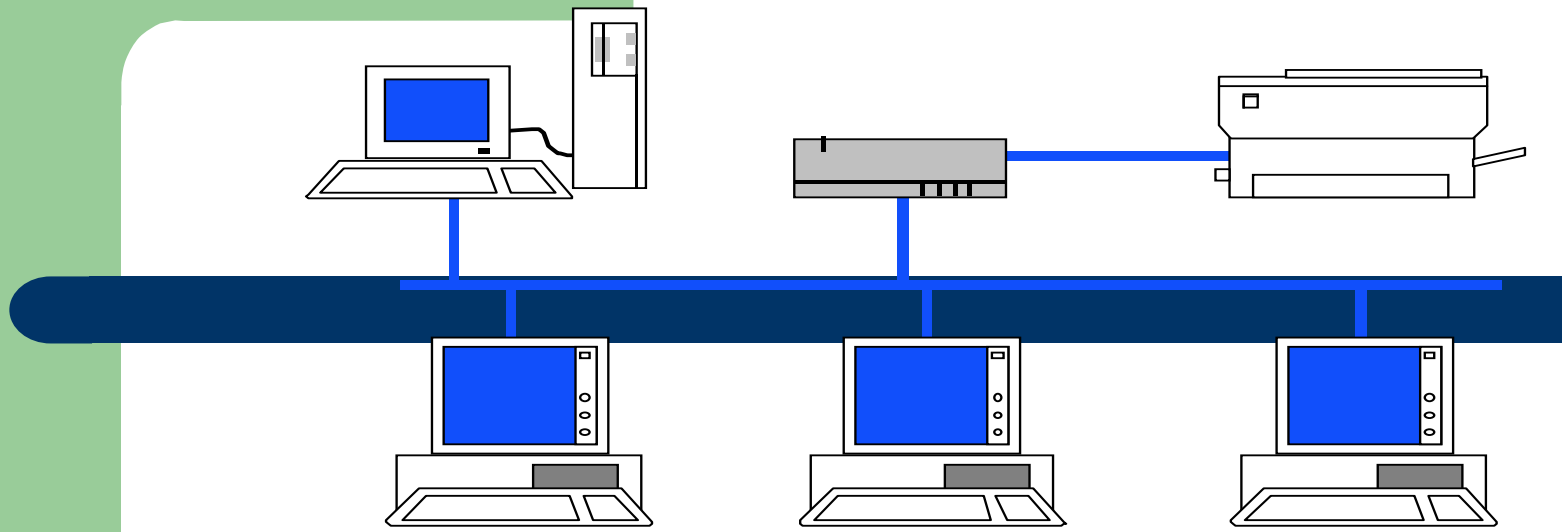
- ***EM analysis:*** use electron microscope to inspect the internal structure of the mask



- ***confusion:*** disturb the power supply/frequency during PIN verification to confuse the accurate enter of PIN and allow access to the protected memory

- ***duplication:*** illegal copying of card content from one to another





- ***tracking*:** based on the protocol exchange between the terminal and the card to track the sequence of commands

Other possible attracts:

- attack on DES like differentiate methods
- attack on RSA using cyclic properties

Security Features

- *Against UV or X-ray inspection:*
  - Using implementation to avoid visiblity of ROM Code
- *EM analysis:*
  - Address Scrambling of memories
- *Against confusion:*
  - Low/High voltage sensors
  - Low/High Frequencies sensors
  - High Frequency Protection

- *Against duplication*:
  - Security PROM Hardware Protected
  - Unique Chip Identification Number
  - Copy/Move Code Blocking
- *Against Tracking*:
  - Secure authentication and data/key encryption
- *Against DPA*:
  - Random Wait State (Advance)
  - Current Scrambling Generator (Advance)
- *Against Cyclic properties*:
  - No simple solutions

FUTURE

Next Generation Smart Card Platform

Traditional vs. High-end Smart Card Hardware (2007+)

8/16 bit CPU	**32 bit CPU**
~ 2K RAM	**16K RAM**
48–64K ROM	**>256K ROM**
8–32K EEPROM	**>128K EEPROM, or Several MB Flash**
External Clock: 1–5 MHz	**Internal Clock: 50 MHz**
Serial I/O Interface 9.6–30K Baud Half Duplex	**High Speed Interfaces** **1.5 Mb/s–12 Mb/s** **Full Duplex**

Next Generation Java Card Platform

- VM technology
  - 32 bit VM
  - .class file loading
  - Concurrent execution of apps/multithreading
  - On-card byte-code verification
- Security
  - Enhanced context-isolation
  - Policy-based security

Next Generation Java Card Platform

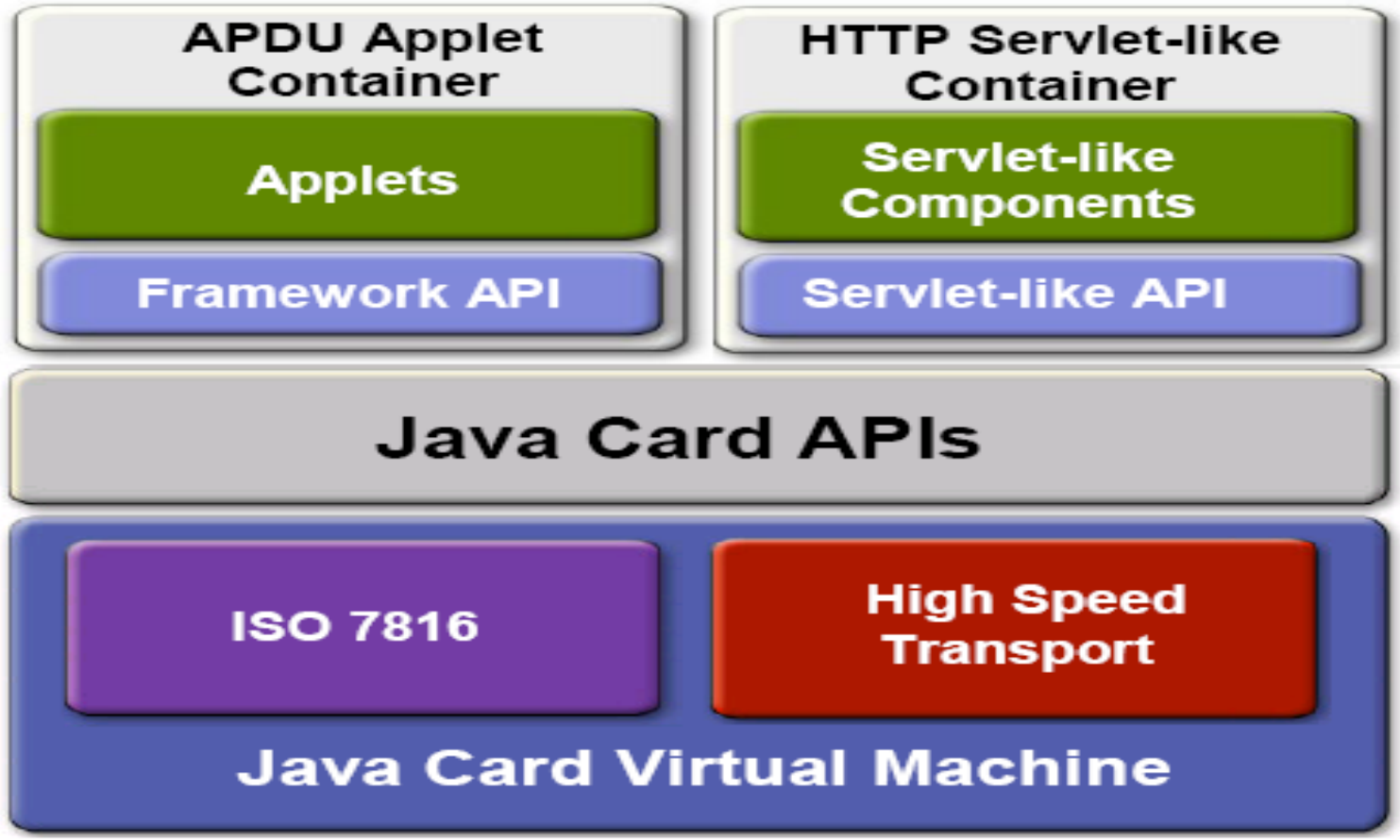
- Network-oriented communication
  - ISO 7816 and TCP/IP communication support
  - Communication over USB, MMC
  - Concurrent contact/contact-less card access
  - Embedded web server
  - Service static and dynamic content through HTTP(s)
  - Client and server communication mode
  - Generic communication API

Next Generation Java Card Platform

- Programming model
  - Fully backward compatible
  - Support additional Java language types: char, long
  - String support
  - Multi-dimensional arrays
  - Support for large data structures—Multimedia content
  - Application models
    - Classic APDU-based applet model
    - HTTP servlet-like model
  - Enhanced inter-application communication framework
  - Generic event framework
  - Evolutive cryptography framework

Next Generation Java Card Platform

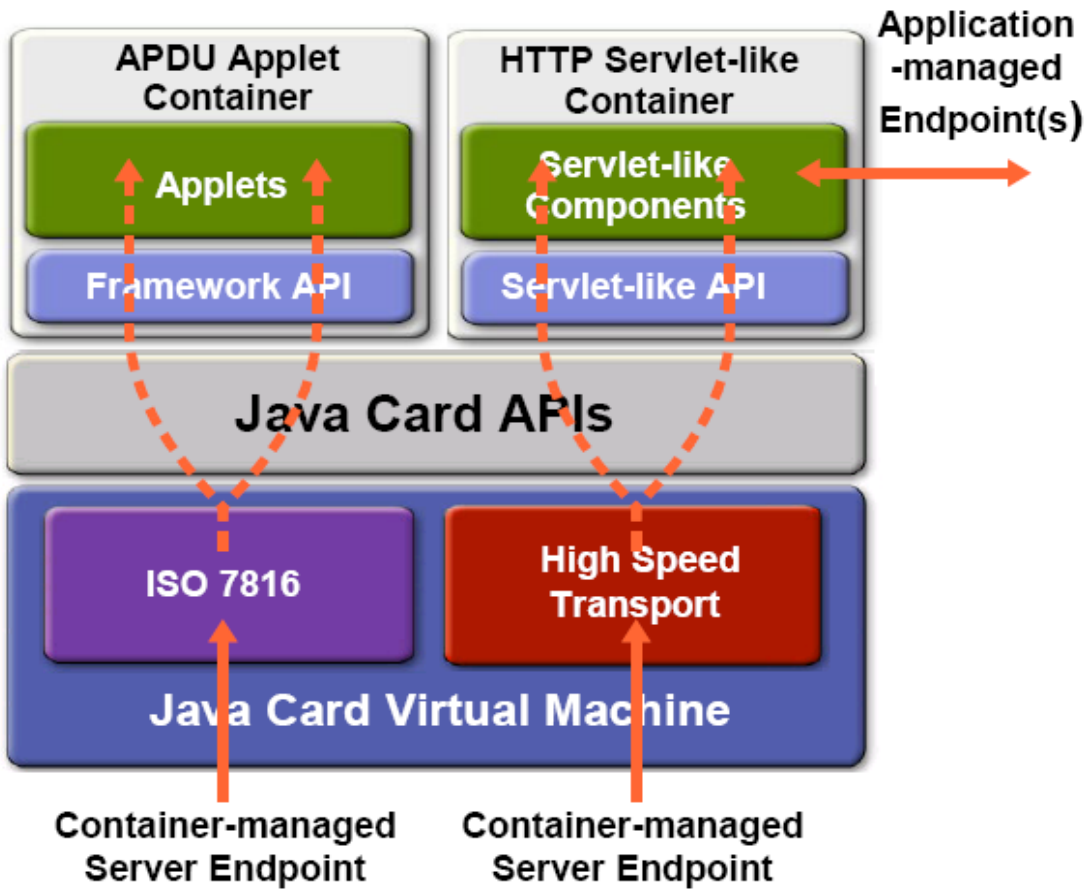
Proposed Software Stack



Next Generation Java Card Platform

Two Levels

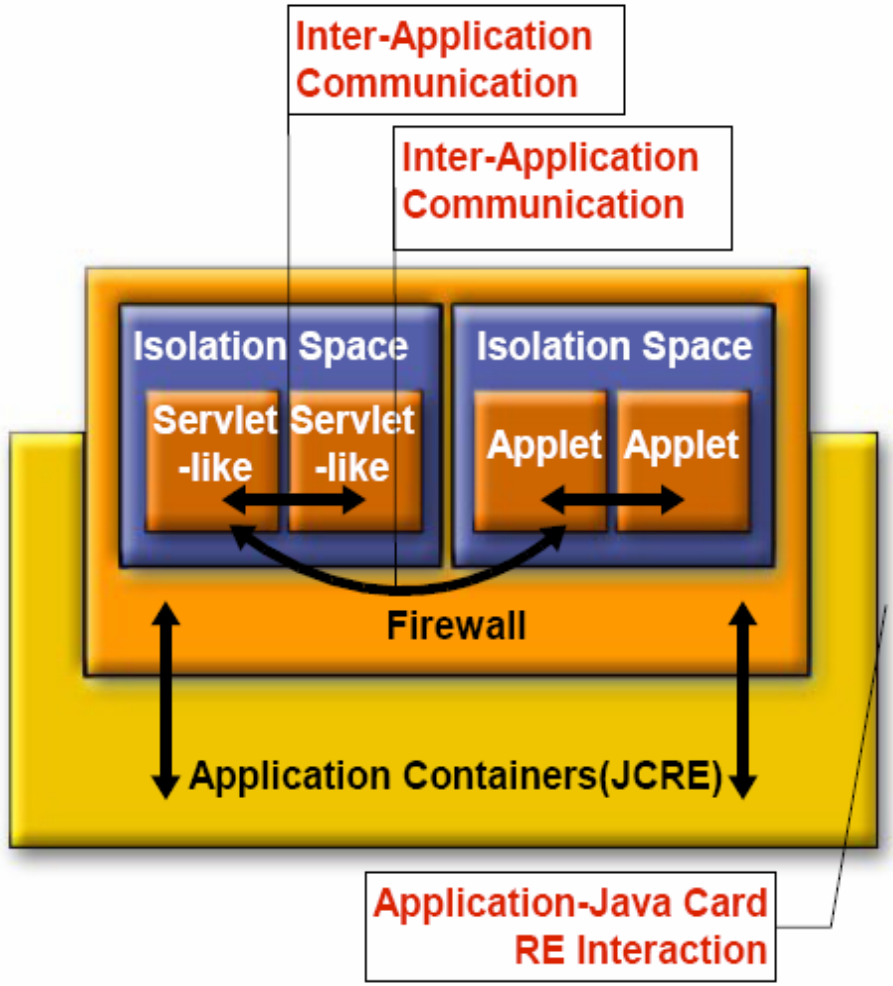
- Container-handled communications
  - Connection endpoints handled by containers
  - Request processing delegated to application components
- Application-handled communications
  - Connections initiated/handled at the application level
  - Can implement both client and server communication model



Next Generation Java Card Platform

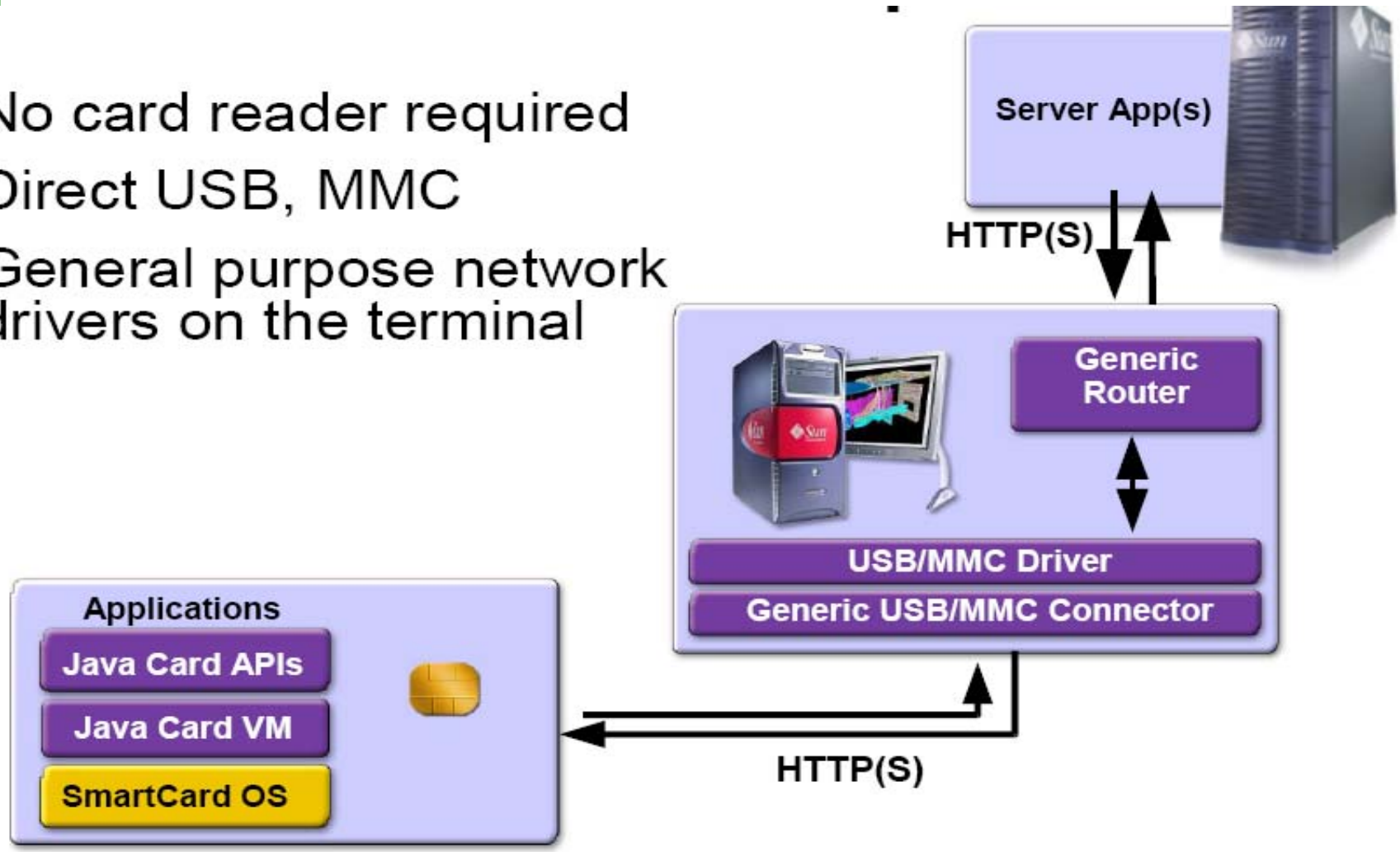
Security Containment Model

- Applications run in secure isolation spaces
  - Components from the same application can interact (share data)
- Containers run in Java Card RE isolation space
- Inter-application and Application-Java Card RE interactions restricted by Firewall
- Policy-based access control across Firewall

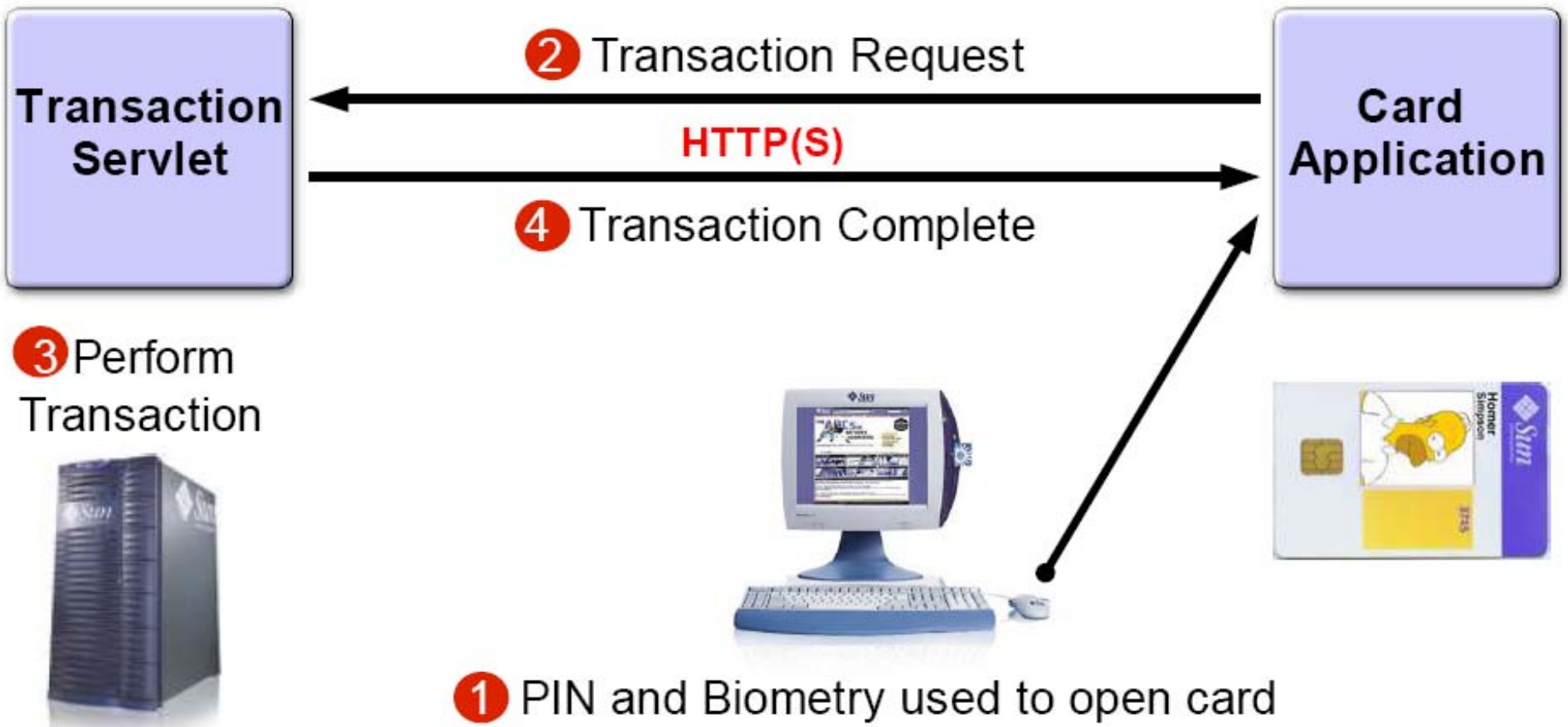


Next Generation Java Card Platform

- No card reader required
- Direct USB, MMC
- General purpose network drivers on the terminal



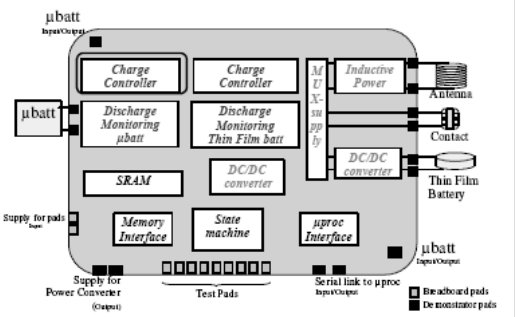
Server Authentication Data Flow

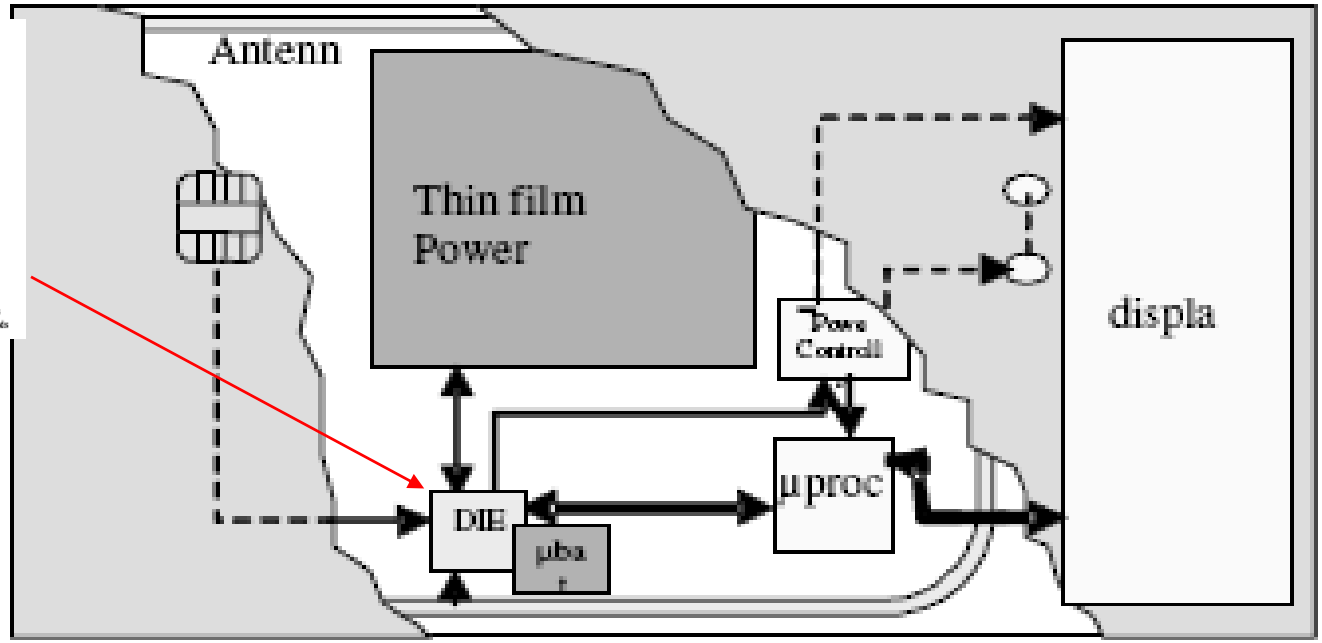


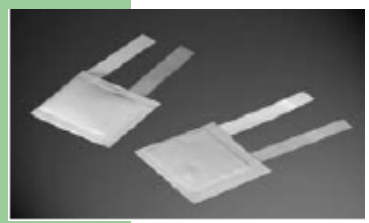
Next Generation Applications

- End-to-end secure applications
  - Servers communicating directly with cards
  - Security does not depend on trusted
- Client device
  - Secure banking transactions
  - Debit/credit
  - DRM applications
  - Browser-enabled applications
  - Private information self-management

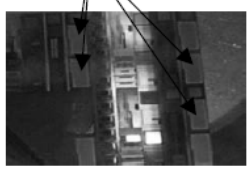
Schematic Multimedia Card

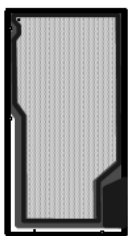


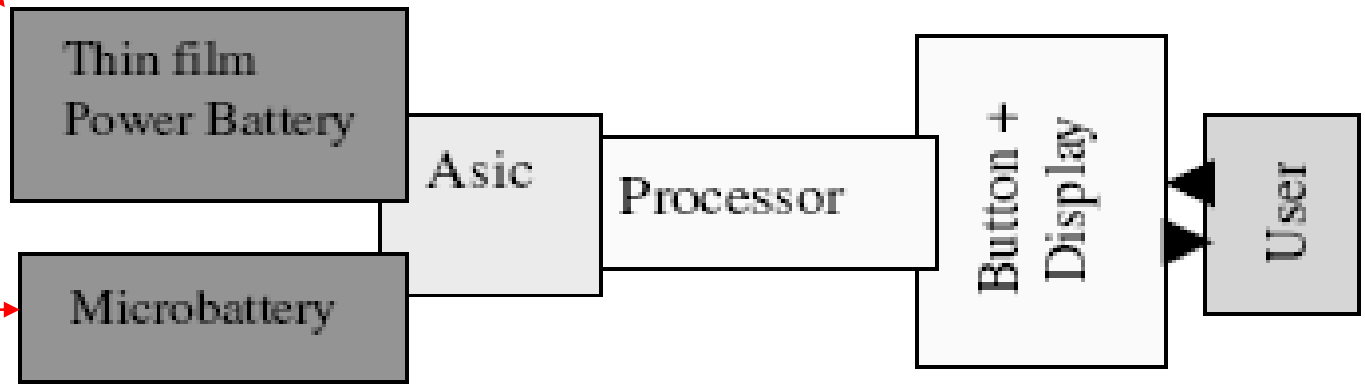




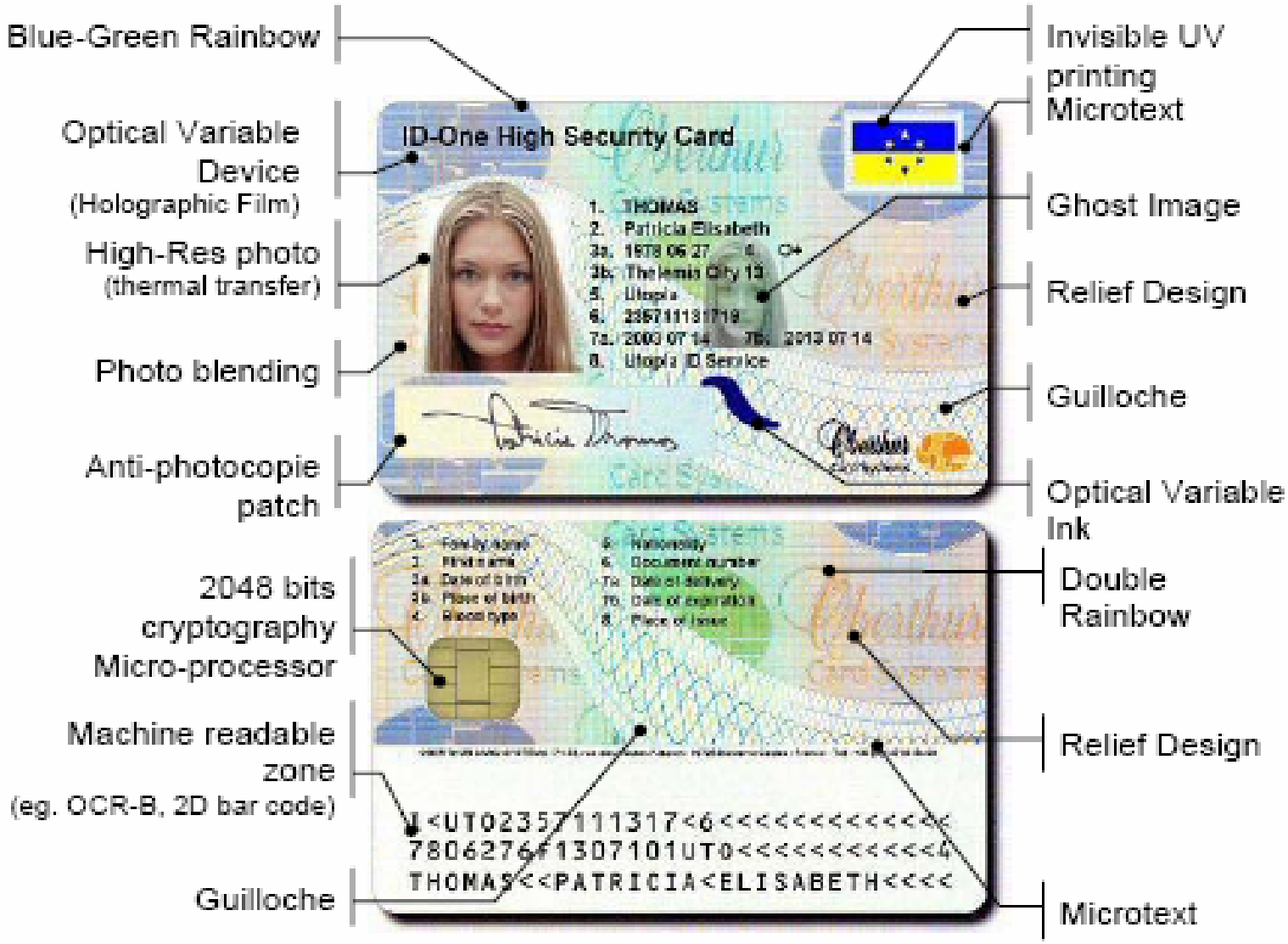
Microbatteries deposited on ASIC
4 x 8 mm overall size







Future High Security Smart card



JAVA CARD 3.0 TECHNICAL OVERVIEW

Technical features

- Communication based on ISO 7816-4 (APDU)
- Asynchronous protocol support
- Support of cap file format
- Support of class (jar) file format
- Enhanced runtime including
- String support
- Multi-threading
- New memory model
- Service oriented framework with events
- Enhanced security

Benefits

- Use in existing Smart Card environment possible
- Usage of high speed interfaces by existing standards (IP protocol based)
- Binary backward compatibility
- Standard format for Java applications
- Needed for app. layer protocols like HTTP
- Allows multiple applications simultaneously
- Speeds up handling of Java data
- Better fit to e.g. ETSI STK event model
- Extension of platform possible
- Security also in non-trusted environment
- Finer grain security

TECH. FEATURES (1/4): COMMUNICATION DETAILS

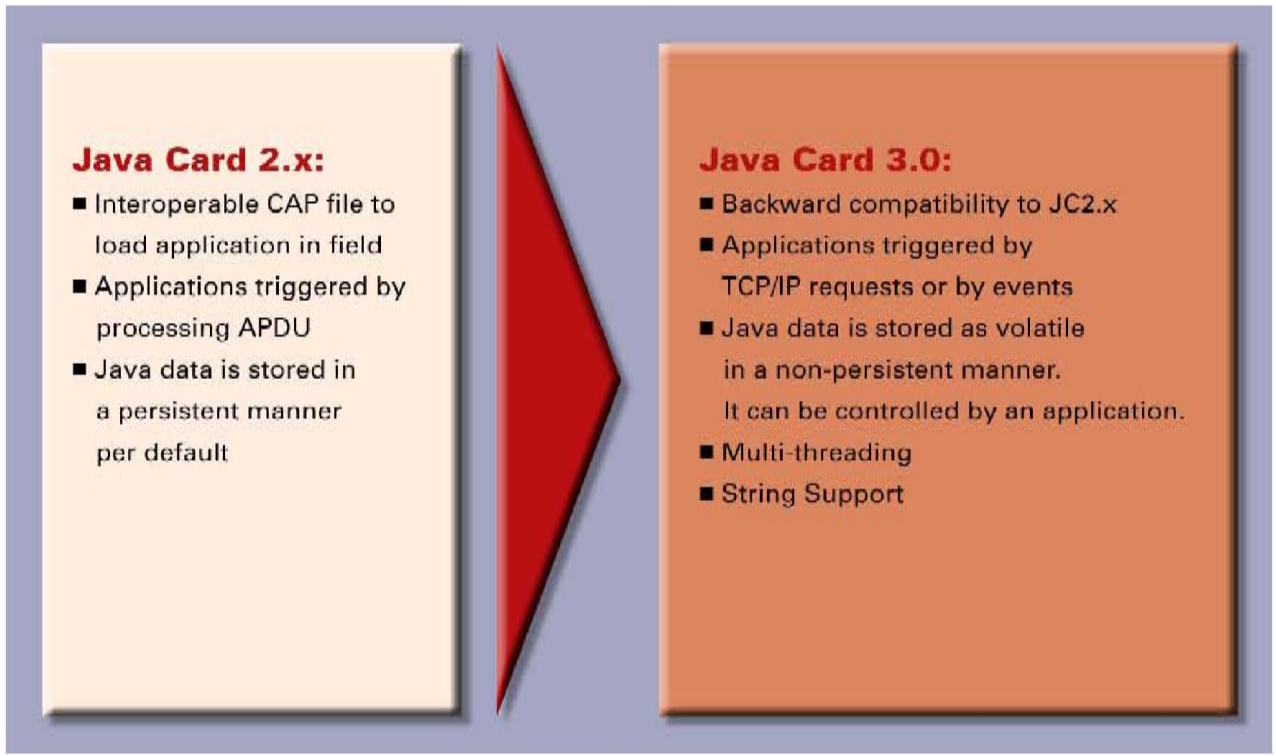
Java Card 2.x:

- ISO 7816 physical interface by default
- Opt. Contactless or USB over APDU possible
- Communication protocols T=0, T=1 (and T=CL)
- Card acts on incoming commands only
- One application active per communication channel

Java Card 3.0:

- Backward compatibility to JC2.x
- High speed protocols parallel on different physical interfaces
- Protocol independent
- Communication using TCP/IP for new applications
- Card can also initiate communications
- More than one application connected on one interface

TECH. FEATURES (2/4): RUNTIME DETAILS



TECH. FEATURES (3/4): SECURITY DETAILS

Java Card 2.x:

- Applications can access Runtime only by API
- Static Firewall (defined at compiling time) to secure application data

Java Card 3.0:

- Backward compatibility to JC2.x
- Access Controller for restrictive API access ?
- Dynamic and flexible firewall (defined at object level)?
- On-card Verification of byte code (class file)

TECH. FEATURES (4/4): APIS+FRAMEWORK DETAILS

Java Card 2.x:

- Minimum Java Card API to allow processing of APDU based applications
- Cryptographic API
- Java Card 2.2 RMI framework
- STK framework adapted to Java Card, defined by ETSI

Java Card 3.0:

- Backward compatibility to JC2.x
- Crypto. API, extendable to new algorithms
- CLDC API, which allows efficient programming of TCP/IP based applications
- Generic framework to enable download/update of application frameworks

Trusted Computer Security Evaluation Criteria – USA(DoD)

- D: Minimal protection
  - No protection
- C1: Discretionary Security Protection
  - Use control access
- C2: Controlled Access Protection
  - Use accountability/auditing
- B1: Labelled Security Protection
  - Use sensitivity (classification) labels

- B2: Structured Protection
  - Use formal security policy more resistant to penetrate
- B3: Security domain
  - Highly resistant to penetration. Use security administrator, auditing events and system recovery process
- A1: Verified protection
  - Highly assure of penetration. Use formal specification and verification approaches.

Information Technology Security Evaluation Criteria (ITSEC) and Common Criteria (CC) – Europe&Canada

- EAL1 - functional tested
- EAL2 - structurally tested
- EAL3 - methodologically tested and checked
- EAL4 - methodologically designed, tested and reviewed
- EAL5 - semiformally designed and tested
- EAL6 - semiformally verified designed and tested
- EAL7 - formally verified designed and tested

Federal Information Processing Standards (FIPS) - evaluation

- FIPS46-2 and 81 for DES
- FIPS 186 for Digital Signature
- FIPS 140-2 for Cryptographic Modules

References

- Java Card™ Technology and Tomorrow's Security Architectures
  - Tanjore Ravishankar, Aseem Sharma
  - Sun Microsystems, Inc.
  - Gemplus Research Labs TS-7136
- Introducing Research Issues for Next Generation Java-based Smart Card Platforms
  - *Gilles Grimaud ,* RD2P Labs, University of Lille, France
  - gilles.grimaud@lifl.fr
  - *Jean-Jacques Vandewalle* Gemplus Software Research Labs, France, jeanjac@research.gemplus.com
- www.javacardforum.org
- Java bytecode verification: algorithms and formalizations Xavier Leroy
  - INRIA Rocquencourt and Trusted Logic S.A., Xavier.Leroy@inria.fr
- Java Card™ Platform Security
  - *Technical White Paper, SUN Microsystems -05*

- Java Card™ Platform Evolution: Future Directions
  - Tanjore Ravishankar,
  - Florian Tournier,
  - Thierry Violleau
  - Java Card Team, Sun Microsystems, Inc. 06
- Java Card™ Technology and Tomorrow's Security Architectures
  - Tanjore Ravishankar,
  - Aseem Sharma
  - Sun Microsystems, Inc. Gemplus Research Labs -05
- New White Card Schemes and Java Card™ Technology
  - Eric Vétillard
  - CTO Trusted Labs -06
  - www.trusted-labs.com

Questions?

How can Java fit on a card?

Supported Java Features

- packages
- dynamic object creation
- virtual methods
- interfaces
- exceptions

Unsupported Java Features

- dynamic class loading
- security manager
- threading
- object cloning
- garbage collection
- large data types

Multi-application cards

- Multi-application cards are an important goal
  - getting more developers on board is essential
- Multiple applets can execute on a card
  - credit, debit, e-cash, loyalty programs
- Explicit and covert channels between applets must be eliminated
  - software risk management