Abstract

Through their combination of portability and security, smart cards are playing an increasingly important role in the rapidly developing areas of electronic commerce and online information services. After giving an overview of smart card characteristics and usage scenarios, we discuss the challenges in the development of smart card applications. Traditionally, this has been a lengthy and difficult process, requiring knowledge of inner workings of the smart card which varied from manufacturer to manufacturer. However, we focus our attention on the current mainstream technology for smart card application development, the Java Card technology. Java Card technology provides a secure, vendor-independent, ubiquitous platform for smart cards that enables rapid development of smart card applications. We present a comprehensive overview of Java Card technology, including concepts, architecture and best practices. Finally, we give a brief survey of the Java Card applet development process.

Keywords: Smart Card, Java Card, Security, Trust

1 Introduction

Smart cards represent nowadays the most portable and secure computing platform available. They are used especially in applications that require high level of security. They are capable of carrying sensitive information of the cardholder and can support security services like authentication, encryption of private data, data integrity and non-repudiation services like signing, etc. This paper serves as an introduction to the development of the on-card portion of smart card enabled applications.

In Section 2, we give an overview of smart cards. First, we look at the differences between memory cards and microprocessor cards and between contact cards and contactless cards. Then, we discuss the different types of card acceptance devices, which are necessary to establish a physical connection with the smart card. Finally, we explain the smart card communication model and the development process of software systems based on usage of smart cards. This section serves as a solid background for the material presented in the next sections.

In Section 3, we focus our attention on the Java Card technology. Java Card technology is a state of the art operating system for smart cards, that allows application developers to create and install on-card applications (Java Card applets) on their own. We will look at the current publicly available specification of the Java Card technology and discuss the contents of the different parts of it.

In Section 4, we show typical steps in the development of a real-world Java Card applet.

Section 5 concludes our introduction to smart card application development using the Java Card technology.

2 Overview of Smart Cards

A smart card (see Figure 1), or integrated circuit card (ICC), is defined as any integrated circuitry embedded into a flat, plastic body. The plastic substrate can have either the size of a credit card, 85.6 mm x 54 mm, which is used in most applications, or can be only 25 mm x 15 mm. The latter form is mostly used in mobile phones and for Security Access Modules in terminals. Depending on the type of the chip, there are two different types of ICCs.

Figure 1. Smart card.
2.1 Memory Cards and Microprocessor Cards

Depending on whether the ICC contains a microprocessor or not, we can divide smart cards into memory cards and microprocessor cards. As the name suggests, memory cards are equipped only with a memory chip and are only able to store limited amount of information, such as health insurance information of the cardholder. Its data processing is performed by a simple circuit capable of executing a few preprogrammed instructions. Depending on the security requirements of the stored data, the memory access can be protected. For example, the memory of a prepaid phone card is protected against unauthorized reloading. The advantage of memory cards lies in the simple technology. Therefore, they are favored in applications where low cost is a priority. They are typically used as prepaid cards for various services sold against prepayment, e.g., the aforementioned prepaid phone cards.

Microprocessor cards, as the name implies, contain a processor. Such a smart card is actually a small, portable, tamper-proof computer, with a CPU and memory. The CPU can be an inexpensive 8-bit processor, but high-end cards contain nowadays a 16-bit or even a 32-bit processor. There are usually three types of memory available on a microprocessor card:

ROM Read Only Memory, is used to store the cards operation system routines and applications. This content is written during production of the card and cannot be changed after the card is manufactured. No power is required to keep the content stored in ROM.

EEPROM Electrical Erasable Programmable Read Only Memory, like ROM, can preserve data content even if the smart card is unpowered. Unlike ROM, the content of this kind of memory can be modified after the card is manufactured.

RAM Random Access Memory, is a transient memory which keeps the data only as long as the card is powered.

A typical smart card can have nowadays around 32 kB of ROM, about the same size of EEPROM, and up to 4 kB of RAM, which is the most expensive and (physically) largest type of memory.

Optionally, the card can contain a cryptographic coprocessor that increases the performance of cryptographic operations.

Such smart cards offer much higher security and functionality than simple memory cards. All cryptographic operations are performed on the card, so that the sensitive information, like cryptographic keys, never leaves the card. The card can be designed for a specific application, or support multiple functionality and be used by many different applications.

For all these differences, in many publications, only microprocessor cards are referred to as being really “smart”. For the purpose of our paper, we require a programmable card supporting Java Card environment, so we will consider only microprocessor cards under smart cards in our further discussion.

2.2 Contact Cards and Contactless Cards

To communicate with the smart card (see Section 2.4 for more information on the smart card communication model), the host computer has to establish a connection to it. This can be done through the contacts on the card (see Figure 2) or via wireless (“contactless”) transmission.

Contactless smart cards are preferable in situations requiring fast transactions. Examples are public transport systems and access control for buildings.

Contact smart cards are the more common type of smart cards. They have to be inserted into the card reader in the right way and with the right orientation. Unfortunately, the contacts are occasionally the cause of failures if they are damaged, worn from excessive use, or just not sufficiently clean.

A contact smart card has eight contact points; their functional assignments are shown in Figure 2. Part 2 of ISO standard 7816 [6] specifies the position, the minimal size, and the usage of the contacts:

![Figure 2. Mechanical contacts of a smart card.](image)

Vcc is used to supply the power to the chip. In general, the voltage is between 4.5V and 5.5V. In mobile phones, smart cards have typically around 3V Vcc voltage.

RST is used for the reset signal to reset the address counter and the microprocessor. This is called a warm reset. A cold reset occurs when the power supply voltage is turned off and on again, e.g., by taking the card out of the reader and reinserting it again.

CLK is used to supply the external clock signal, from which the internal clock is derived.

GND is used as reference voltage; its value is considered to be zero voltage.
Vpp is optional and is used only in older cards to supply voltage necessary to program the EEPROM memory in these cards.

I/O is used to transfer data between the smart card and the reader in a half-duplex mode, i.e., data can be transmitted in only one direction at any particular time.

RFU points are reserved for future use.

2.3 Card Acceptance Device

Before the communication between the computer and the smart card can take place, it is necessary to establish a physical connection with the smart card. In case of contact cards, the smart card has to be inserted into the card acceptance device. There are two types of card acceptance devices:

- Smart card readers
- Smart card terminals

A smart card reader is essentially a connector between the computer communicating with the card and the smart card. It can be attached to the serial, parallel, or USB port of a computer. Other reader types are integrated with the keyboard or fit into the PCMCIA slot. A smart card reader has usually one slot for a smart card. However, there are readers with more than one slot; they are used for a special type of applications, where one smart card is required for authorization before reading the data from another card. In addition to the card slot and the computer interface, a smart card reader can have also a display and a PIN-pad.

In contrast, a smart card terminal is a computer on its own. It integrates a smart card reader as one of its components. Smart card terminals are used nowadays mostly in stores for payments and credit card transactions. Another form of terminal is a bank ATM. An ATM, if it accepts smart cards, can be used for example to load money to an electronic purse, to check the actual balance, to change a password, etc.

For our further discussion, we will assume a normal smart card reader connected to a computer. However, most of the discussion, especially about the communication model in the next section, applies to applications running inside a smart card terminal as well.

2.4 Smart Card Communication Model

The communication between the host computer and the smart card is half-duplex; i.e., the data can either be sent from the host to the card or vice versa, but not in both directions at the same time.

The protocol stack used for communication has two layers, which we discuss in the following subsections.

2.4.1 APDU Protocol

The top layer is the APDU (application protocol data units) protocol, which is specified in ISO 7816-4. This protocol defines two types of messages, the command APDU and the response APDU. The first one is used by the host application to send commands to the card, the other is used by the card to send responses back to the host application. In this way, a command APDU is always paired with a response APDU. Their structures are depicted in Figure 3 and Figure 4, respectively.

![Figure 3. Command APDU structure.](image-url)

<table>
<thead>
<tr>
<th>Header(required)</th>
<th>Body(optional)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLA</td>
<td>INS</td>
</tr>
</tbody>
</table>

Each command APDU contains [2]:

- A class byte (CLA). It identifies the class of instruction of the command.
- An instruction byte (INS). It determines the specific command.
- Two parameter bytes P1 and P2. These are used to pass command specific parameters with the command.
- A length byte Lc (“length command”). It specifies the length of the optional data sent to the card with this APDU.
- Optional data.
- A length byte Le (“length expected”). It specifies the expected length of the data returned in the subsequent response APDU.

CLA, INS, P1, P2 form the header of the command APDU. This header is mandatory. Lc, optional data, and Le form the optional body of the command APDU. This body can have several variations, which we won’t discuss here in detail. For more information, please refer to [1].

![Figure 4. Response APDU structure.](image-url)

<table>
<thead>
<tr>
<th>Body(optional)</th>
<th>Trailer(required)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>SW1</td>
</tr>
</tbody>
</table>

A response APDU contains:

- Optional data.
- Two status word bytes SW1 and SW2. They contain the status information as defined in ISO 7816-4. In case of successful execution, they contain $0x9000$. 

G-3
The length of the optional data in the response APDU is determined by the Le byte in the corresponding command APDU. Should an error occur, no optional data might be returned despite of the specified length.

2.4.2 TPDU Protocol

The next lower layer is the transport protocol, specified in ISO 7816-3. APDUs are transmitted by this protocol in data structures called transmission protocol data units, or TPDUs. The protocols T=0 and T=1 are the two most-used variations of the transport protocol. The T=0 protocol is byte oriented, which means that each byte is transmitted separately, whereas the T=1 protocol is block-oriented; i.e. a single unit of transfer between a card and a host is a block, consisting of a sequence of bytes. Most modern smart card readers support both T=0 and T=1 protocols. The transport protocol supported by the card is discovered from the ATR, which is a sequence of bytes that the smart card returns to the reader immediately after it is powered up. In addition to the protocol information, the ATR contains data identifying the type of the card. The ATR is defined in ISO 7816-3.

2.5 Smart Card System Development Process

A smart card system is a distributed system that consists of the following two parts:

- Host (off-card) application
- On-card application

The host application is the part that resides on the computer connected to the smart card through a smart card reader device. Such application typically recognizes a specific card and handles the communication with the card by exchanging a predefined set of APDUs with the card. Host application is usually written in a high-level programming language, such as Java or C++. For Java development, the de facto standard framework, that supports host application development, is the OpenCard Framework (OCF). Unfortunately, the development of host applications is out of scope of this paper, so we refer readers interested in Java development using OCF to [2].

The development of the on-card application has been traditionally a lengthy and difficult process, requiring knowledge of inner workings of the smart card and usage of proprietary toolkits and low-level assembly languages. Such application development has been done by the card operating system developers. The code was then integrated into the mask for the ROM of the smart card before the card was manufactured.

Current smart card operating systems enable application developers to create and install on-card applications on their own. The mainstream technology in this area is the Java Card technology, which we cover in the next section.

3 Introduction to Java Card Technology

In 1997, Sun Microsystems realized the potential of smart cards, and defined a set of specifications for a subset of Java technology to create applications for them, Java Card applets. This brought the main advantages of Java to smart card application development. Additional security mechanisms allowed multiple applications from different vendors to run on the same card without compromising each other’s security.

The Java Card technology specification, currently in version 2.2.1, consists of three parts:

- The Java Card Virtual Machine (JCVM) specification, which defines a subset of the Java programming language and a virtual machine for smart cards.
- The Java Card Runtime Environment (JCRE) specification, which further defines the runtime behavior for Java enabled smart cards.
- The Java Card Application Programming Interface (API) specification, which defines the core and extensional Java packages and classes for smart card application development.

3.1 Java Card Language Subset

The Java Card Virtual Machine supports only a restricted subset of the Java programming language, yet it preserves many of the familiar features including objects, inheritance, packages, dynamic object creation, virtual methods, interfaces, and exceptions. The JCVM specification drops the support for a number of language elements that would use too much of a smart card’s limited memory [4]:

- Dynamic class loading, security manager, threads, object cloning, and certain aspects of package access control are not supported.
- Keywords native, synchronized, transient, volatile, strictfp are not supported.
- There is no support for char, double, float, and long, or for multidimensional arrays. Support for int is optional.
- The Java core API classes and interfaces (java.io, java.lang, java.util) are unsupported except for Object and Throwable, and most methods of Object and Throwable are not available.

---

1Sun Microsystems has recently initiated JSR 268, whose objective is to add a smart card I/O API to Java Platform, Standard Edition (Java SE).
• The Exception and Error subclasses that cannot arise in the Java Card platform are omitted.

3.2 Java Card Virtual Machine

The virtual machine for the Java Card platform is implemented in two separate pieces. One portion resides on the card itself, while the other one, which is rather a development tool, is external to the card. The purpose of this external part, which is often referred to as the Java Card Converter tool, is to load, verify, and further prepare the Java classes in a card applet for on-card execution. The output of the converter tool is a Converted Applet (CAP) file, a file that contains all the classes in a Java package in a loadable, executable binary representation, and an export file representing the public API of the package. The on-card part of JCVM interprets bytecode instructions, manages classes and objects, and enforces the runtime security model.

Before the on-card interpreter can execute the code found in the CAP file, it has to be loaded onto the card. The process of downloading and installing a CAP file is performed by a unit called the installer.

The Java Card installer resides within the card. It cooperates with an off-card installation program, that transmits the CAP file to the installer running on the card via the card reader device. The installer then writes the binary into the smart card memory, links it with the other classes that have already been placed on the card, and creates and initializes any data structures that are used internally by the Java Card runtime environment [3].

3.3 Java Card Runtime Environment

The Java Card runtime environment (JCRE) consists of the Java Card virtual machine, the Java Card API classes, and industry specific extensions. The JCRE is basically the smart card’s operating system; its responsibilities include the card resource management, I/O communication and applet life-cycle management, and Java Card security model enforcement.

The JCRE is initialized at card initialization time. During this process, which is performed only once during the card lifetime, the JCRE initializes the virtual machine and creates objects for providing the JCRE services and managing applets. Thus, these objects live for the whole lifetime of the JCVM. As applets are installed, the JCRE takes care of creating applet instances, and managing their life-cycles.

During the period from the time the card is inserted into the card reader and is powered up until the time the card is removed from the card reader, the JCRE operates like a typical smart card—it supports APDU I/O communication with a host application.

After a JCRE reset, the JCRE enters a loop, waiting for APDU commands to arrive from the host. The host sends APDU commands to the Java Card platform using the serial communication interface via the card I/O contact point. When a command arrives, the JCRE either selects an applet to run as instructed in the command or forwards the command to the currently selected applet. The selected applet then takes over control and processes the APDU command. When finished, the applet sends a response back to the host application and surrenders control to the JCRE. This process repeats when the next command arrives. For more detailed description of JCRE and applet life-cycle, please refer to [3].

3.3.1 Java Card Runtime Features

Besides supporting the Java language runtime model, the JCRE supports three additional runtime features [3]:

• **Persistent and transient objects**—Java Card objects are persistent by default. They are created in EEPROM and retain their state even when the card is not powered. For security or performance reasons, applets can create objects in RAM. Such objects are called transient objects and they keep their state only while the card is powered. For more information on persistent and transient objects please refer to [3, Chapter 4].

• **Atomic operations and transactions**—The Java Card virtual machine ensures that each write operation to a single field of a persistent object is atomic. It guarantees that the updated field either gets the new value or is restored to its original value before the write attempt. Additionally, the JCRE provides transaction APIs, that let the applet developer include several write operations in a transaction. Then, either all updates in this transaction complete successfully, or the state of all variables affected by this transaction is restored to the previous state. [3, Chapter 5] gives a good introduction to the topic of atomic operations and transactions.

• **Applet firewall and the sharing mechanisms**—On a Java Card enabled smart card, multiple applications from different vendors can coexist securely. JCRE provides the applet firewall mechanism, that isolates applets from each other. Hence, the existence and operation of one applet has no effect on the other applets on the card. In situations where applets need to share data, the virtual machine permits such functionality through a well defined secure sharing mechanisms. See [3, Chapter 9] for more information on applet firewall and the sharing mechanisms.
3.4 Java Card API

The Java Card API specification defines a small subset of the traditional Java programming language API. Due to the smart card’s strict memory restrictions, many fundamental classes like String, Thread, System or Class are not supported. There is no support for wrapper classes like Boolean and Integer, and the whole collections framework is missing as well. In addition to its small subset of the traditional Java classes, the Java Card API specification defines its own set of core classes specifically to support Java Card applet development. These are contained in the following packages:[4]:

- java.lang defines Object and Throwable classes that lack many of the methods of their Java SE counterparts. It also defines a number of exception classes: the Exception base class, various runtime exceptions, and CardException. None of the other traditional java.lang classes are included.
- java.rmi defines the Remote interface and the RemoteException class. None of the traditional java.rmi classes are included. Support for Remote Method Invocation (RMI) is included to simplify migration to, and integration with, devices that use Java Card technology.
- java.io defines one exception class, the base IOException class, to complete the RMI exception hierarchy. None of the other traditional java.io classes are included.
- javacard.framework defines the interfaces, classes, and exceptions that compose the core Java Card Framework. It defines important concepts such as the Personal Identification Number (PIN), the Application Protocol Data Unit (APDU), the Java Card applet (Applet), the Java Card System (JCS), and a utility class. It also defines various ISO 7816 constants and various Java Card-specific exceptions.
- javacard.framework.service defines the interfaces, classes, and exceptions for services. A service processes incoming commands in the form of an APDU.
- javacard.security defines the classes and interfaces for the Java Card security framework. The Java Card specification defines a robust security API that includes various types of private and public keys and algorithms, methods to compute cyclic redundancy checks (CRCs), message digests, and signatures.
- javacardx.crypto is an extension package that defines the interface KeyEncryption and the class Cypher, each in its own package for easier export control. KeyEncryption is used to decrypt an input key used by encryption algorithms. Cypher is the base abstract class that all ciphers must implement.
- javacardx.rmi is an extension package that defines the Java Card RMI classes. It defines two classes, CardRemoteObject and RMIService. CardRemoteObject defines two methods, export() and unexport(), to enable and disable remote access to an object from outside the card. RMIService extends BasicService and implements RemoteService to process RMI requests.

4 Developing a Java Card Applet

There are two different programming models that can be employed for the development of a Java Card applet. The first model is the fundamental message-passing model, designed around the APDU protocol, which we will examine in this section. The second model is based on the Java Card Remote Method Invocation (JCRMI), a subset of the Java SE RMI distributed-object model. JCRMI provides a distributed-object model mechanism on top of the APDU-based messaging model. For the sake of brevity, we won’t cover development of Java Card applets using JCRMI any further.

Developing a Java Card applet using the APDU based approach is a two-step process:

1. Defining the command and response APDUs that serve as the interface between the host application and the applet.
2. Writing the Java Card applet itself.

4.1 Defining APDU Instructions

The APDU instruction set supported by an applet is always domain specific. For example, a wallet applet that stores electronic money may support functions like credit, debit, verify PIN and check balance. A health insurance applet may allow to access health insurance information, coverage limits, doctors, patient information, etc. Thus, the exact APDUs that you define depend on the functional requirements of your application.

4.2 Writing the Java Card Applet

Every applet is implemented by creating a subclass of javacard.framework.Applet class. The JCRE invokes the methods install(), select(), deselect(), or process(), which are defined in the
import javacard.framework.*;
...
public class MyApplet extends Applet {
    // Definitions of APDU-related
    // constants
    ...  
    // Constructor
    MyApplet() {...}
    // Life-cycle methods
    install() {...}
    select() {...}
    deselect() {...}
    process() {...}
    // Private methods
    ...
}

Figure 5. Typical Java Card applet structure.

4.2.1 The Constructor
The private constructor is called from the install() method; i.e., the constructor is called only once during the lifetime of the applet. Any objects that might be required during execution of an applet should be preallocated in the constructor, to ensure that the applet will never fail due to lack of memory.

4.2.2 The install() Method
The JCRE invokes install() during the applet installation process. This method must be implemented and must create the applet instance by using the new operator followed by a call to the applet’s constructor. The install() method must directly or indirectly call the register() method to complete the installation.

4.2.3 The select() Method
The JCRE invokes select() to notify the applet that it has been selected for APDU processing. The select() method must return true to indicate that it is ready to process incoming APDUs, or false to decline selection. The default implementation by the javacard.framework.Applet class returns true.

4.2.4 The deselect() Method
The JCRE invokes deselect() to notify the applet that it has been deselected. This method may be overridden to provide session cleanup. The default implementation by the javacard.framework.Applet class does nothing.

4.2.5 The process() Method
Once an applet has been selected, it is ready to receive command APDUs. Every time the JCRE receives an APDU command (from the host application via the card reader), it calls the currently selected applet’s process() method, passing it the incoming command APDU as an argument. The process() method then [5]:

1. Extracts the APDU’s CLA and INS fields.
2. Retrieves the application-specific P1, P2, and data fields.
3. Processes the APDU’s data.
4. Generates and sends a response.
5. Returns gracefully, or throws an appropriate ISO exception.

At that point, the JCRE sends the appropriate status words back to the host application, via the card reader.

This concludes our introduction to the development of a Java Card applet. For an extensive code example, please refer to [3, Chapter 12].

5 Conclusions
Smart cards represent nowadays the most portable and secure computing platform available. They are used to store sensitive information and process transactions securely. Java Card technology brings smart card application development into the mainstream while preserving smart card security.

In this paper, we have first given a detailed survey of smart cards and their applications. Thereafter we have looked closely on the current Java Card platform specification, discussing its architecture and concepts. Finally, we have demonstrated typical development steps of a real-world Java Card applet. Readers interested in a more detailed guide should refer to [3].

References


