Graz University of Technology

Institute for Information Systems and Computer Media

Lund University

Department of Information Technology

Master Thesis in Telematics
Information Systems

Smart Card Security Services for an Open Application Environment used in Mobile Phones

Florian Eisl

Sony Ericsson Mobile Communications AB
Lund

Lund, June 2004

Supervisor and Assessor
Univ.-Doz., Dipl.-Ing., Dr.techn. Klaus Schmaranz

Acknowledgement

In preparing to this paper, I owe much to my supervisor Stefan Andersson and my colleges Marcus Liwell, Jonas Hörström and Tobias Karlsson at Sony Ericsson Mobile Communications AB and Sebastian Hans from Sun Microsystems.

I thank Ben Smeets and Klaus Schmaranz, my supervisors from Lund University and Graz University of Technology, for their guidance and support throughout this project.

For their help and support during my study in Graz, I must give special thanks to my friends Albert and Wolfgang.

Finally I would like to thank my family and Sofia for their love, support, and patience during the duration of my study.

I hereby certify that the work reported in this thesis is my own and that work performed by others is appropriately cited.

Signature of the author:

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten Anderer unverändert oder mit Abänderungen entnommen würde.

Abstract

A world without mobile phones is no longer imaginable. They changed the way people communicate with each other and are part of everyday life. However the functionality of mobile phones has changed a lot since the early beginning in the 80s. Back then the main purpose of these phones was to talk to other people.

Nowadays mobile phones are multimedia devices with a functionality that goes far beyond the functionality of their predecessors. Due to the functionality enhancement security has become a more and more important issue.

This paper focuses on security and trust services in mobile phones. Security and trust services are in an IT context usually divided into five fundamental components, namely authentication, authorization, integrity, non-repudiation, and reliability.

These services are implemented in a Sony Ericsson mobile phone using an open application environment. The Java technology for mobile devices is called J2ME (Java 2 Micro Edition).

The security and trust services are based on the definitions contained in the JSR-177 (Java Specification Request 177) that specifies the security and trust service APIs (Application Programming Interface) for J2ME enabled devices.

The main interest of this thesis is the communication between the mobile phone and the integrated security element, a smart card. The communication between the two elements is based on the APDU (Application Protocol Data Unit) protocol specified by the ISO/IEC 7816 standard.

Zusammenfassung

Eine Welt ohne Mobiltelefone ist nicht mehr vorstellbar. Die Art miteinander zu kommunizieren wurde durch sie entscheidend verändert und sie sind ein nicht mehr wegzudenkender Bestandteil des Alltags. Aber die Funktionalität hat sich, verglichen mit den Telefonen aus den 80igern, wesentlich verändert. Damals wurden Mobiltelefone noch hauptsächlich zum Telefonieren verwendet.

Heute sind Mobiltelefone Multimedia Geräte mit einer Funktionalität, die jene ihrer Vorgänger bei weitem übersteigt. Aufgrund der Erweiterung der Funktionalität wird das Thema Sicherheit zunehmend wichtiger.

Diese Diplomarbeit konzentriert sich auf Sicherheits- und Vertrauensdienste in Mobiltelefonen. Sicherheits- und Vertrauensdienste werden, betrachtet in einem informationstechnologischem Kontext, normalerweise in fünf grundlegende Elemente, nämlich Authentisierung, Autorisierung, Integrität, Unleugbarkeit und Zuverlässigkeit, unterteilt.

Diese Dienste sind in einem Sony Ericsson Mobiltelefon implementiert. Die Implementierung wendet das 'Open Application Environment' J2ME (Java 2 Micro Edition) an.

Die Sicherheits- und Vertrauensdienste basieren auf den Definitionen, die im JSR-177 (Java Specification Request 177) enthalten sind, welcher die Sicherheits- und Vertrauensdienst APIs (Application Programming Interface) für J2ME fähige Geräte spezifiziert.

Das Hauptinteresse ist die Kommunikation zwischen dem Mobiltelefon und dem darin enthaltenen Sicherheitselement, einer Smart Card. Die Kommunikation zwischen den beiden Elementen basiert auf dem APDU (Application Protocol Data Unit) Protokoll, das dem ISO/IEC 7816 Standard entspricht.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The services available on state-of-the-art mobile phones are becoming more and more complex. A mobile phone is not just used for voice calls any longer. In addition to sending simple SMS (Short Message Service) messages, mobile phone users now send EMS (Enhanced Message Service) and MMS (Multimedia Messaging System) messages. Recording sounds, taking pictures and playing MP3 files are the newest trends.

The phone has become a multimedia device with many different applications running on it. In order to be able to protect the phone and its user a number of security services need to be implemented in modern mobile phones.

1.1 Motivation

In today's world the number of mobile phone users is rapidly increasing. With them the number of phones which are connected to the Internet. New technologies enable operators and content providers to expand the features of a mobile phone to allow the mobile phone users to personalize their phones to an extent never seen before. Users can download new applications, like, e.g. utility software and games. The field of software development is not limited to the handset producers any longer, third parties are now able to develop software for mobile phones.

To satisfy this greater demand of personalization a more powerful application development platform was to be used. The J2ME (Java 2 Micro Edition)[1] is such a platform. It delivers the power and benefits of Java technology fitted for consumer and embedded devices. This includes a flexible user interface, robust security model, wide range of built-in network protocols, and support for

[1] `http://java.sun.com/j2me/`

networked and disconnected applications [Mic02a].

The JCP (Java Community Process)[2] created the JSR-177 (Java Specification Request 177) in order to upgrade the security model. It adds a security and trust service API (Application Programming Interface) or short SATSA (Security and Trust Services API). The purpose of these APIs is to provide the functionality that enables a device to establish a secure communication.

1.2 Structure of this Work

This master thesis is structured as follows:

Chapter 1, *Introduction:* This chapter gives an introduction to the JSR-177 and the problem definition. Furthermore, it describes the usage of a SC (Smart Card) in a mobile phone and shows the available open application environments.

Chapter 2, *Smart Card:* Since the creation of the first SC a lot has changed. SCs have become more powerful (processor, memory, cryptographic operations, ...) and there are many suitable applications where SCs are in use. SCs are famous for their physical security - the tamper-resistant packaging. Therefore SCs are commonly used to implement a SE (Security Element) in a mobile phone.

Chapter 3, *Used Java Technologies:* In order to be able to develop software for SCs and portable devices like mobile phones special programming environments need to be used. JC (Java Card) and J2ME are programming environments specially developed for this purpose. Both technologies have their own VM (Virtual Machine) especially designed to run on these limited devices. JC applets can be created and loaded onto a SC. In J2ME applications are better known as MIDlets. To guarantee that J2ME can be used for consumer and embedded devices the J2ME architecture defines configurations, profiles and optional packages.

Chapter 4, *Design of the Java Specification Request 177:* Security and trust services are usually divided into five domains: confidentiality, authentication, integrity, non-repudiation, and reliability. The JSR-177 defines a collection of APIs that provide these services for J2ME enabled devices. The focus is particularly on the SC communication API for accessing data and functions on a SC via the APDU (Application Protocol Data Unit) protocol.

[2] `http://www.jcp.org/`

Chapter 5, *Implementation:* As described in Chapter 4 the JSR-177 defines security and trust services APIs. The implementation deals with two packages of the JSR-177. The package `javax.microedtion.io` consisting of the Connector class for creating new connection objects and the `javax.microedition.apdu` package, which defines an interface for APDU connections.

Chapter 6, *Conclusion and Outlook:* A synopsis of the achieved goals of the implementations is shown. Furthermore, an overview of the evaluation of the JSR-177 is given and future work with the JSR-177 is outlined in this chapter.

1.3 Smart Cards in Mobile Phones

The beginning of the 80s was the "hour of birth" for cellular telephony in Europe and they quickly reached a large popularity. In the beginning there was no standardized system and almost every country had an own telephone system. The early systems were not safe at all, eavesdropping on the traffic was not difficult and it happened quite often that users were charged for other people's calls. This resulted in substantial losses for both parties, the telephone operators and their customers.

Adding adequate security mechanisms was therefore one of the most important issues when companies started to think about a Europe-wide mobile telephone system. The result of this developing process is the today well-known GSM (Global System for Mobile Communications). The GSM is today's most widely used telephone standard, although another standard called PCS (Personal Communications Service) is in use in North America and Japan.

To implement the above mentioned necessary security mechanisms, a special module was integrated into mobile phones, the SIM (Subscriber Identity Module).

Subscriber Identity Module

The SIM needs to fulfil the following three GSM security requirements [Hen01]:

- User Authentication;
- Protection of the integrity and reliability of calls;
- Protection of the confidentiality of calls and call-related data.

The SIM card is known by everybody who is using a mobile phone. It is a SC on which the SIM application is executed. It is possible for a subscriber to have more than one SIM (different network providers) or to use the same SIM in different phones. A special application is defined by the GSM standard to protect the data in the card and also for access control.

To enhance the functionality of the SIM further, the SAT (SIM Application Toolkit) was created.

SIM Application Toolkit

The SAT defines a standard interface between the applications running in the SIM and the telephone's display, keypad, etc. The SAT makes it possible to, remotely controlled (over the air) or by a user command, load data and/or applications to the SIM and to control the ME (Mobile Equipment).

A further step in the evolution of mobile services is the implementation of the WAP (Wireless Application Protocol).

Wireless Application Protocol

WAP 1.0 enables mobile telephones to display a subset of HTML used for displaying content on web sites which is called WML (Wireless Markup Language). WML has limited display and navigation possibilities. Nevertheless it enables mobile phone users to access data available anywhere on the Internet.

In the newer version WAP 2.0 embraces the new standards for the Internet browser markup language. This has led to the definition of the XHTMLMP (eXtensible HyperText Markup Language Mobile Profile). WAP 2.0 supports HTTP/HTML and provides backward compatibility to previous WAP versions.

1.4 Open Application Environments

In an open application environment the SC operating system allows third parties to load data and applications onto the SC without involving the producer of the SC operating system. Another term often used for open application environments is open platforms. In general open platforms specifications are public and are created by a consortium of companies.

The benefit of an open application environment is that it speeds up the development process of SC applications. And this leads to a reduction of development

costs. Furthermore, it increases the flexibility of the SC application development process.

Several open platforms for SCs are available [RE04]. All of these open platforms support application design for multi application SCs. A multi application SC is a SC that contains more then one application. Furthermore, these applications can be selected at the same time.

- Java Card
- Multos
- Basic Card
- Windows for Smart Cards
- Linux

Java Card

JC is a SC operating system which enables SCs to run programs, called applets, written in Java. Strictly speaking JC is not a true operating system because of the fact that the JC specification does not include a file management. But in practice JC is considered as the prototype of an open SC operating system.

MULTOS

As JC, MULTOS is a multi application SC operating system. Its origin goes back to the development of the Mondex system for electronic purses [RE04]. The goal was to develop an operating system which meets the requirements of electronic payment systems.

The design of MULTOS conforms the ISO/IEC 7816-4[3] standard and interprets code downloaded onto the SC. Typically the code for an application is written in C and later transformed into the MEL (Multos Executable Language) using a special compiler.

In order to enable a download of an application onto a SC this application needs to be digitally signed by a licensed MULTOS certification service.

[3] `http://www.iso.org/`

Basic Card

Besides JC and MULTOS, Basic Card is a third multi application SC operating system which allows executable program code to be downloaded by third parties. Basic Card is based on the programming language Basic and several versions with different features for different SCs with various memory sizes are available.

The program generation procedure is based on the traditional Basic interpreters. First the source code is translated by a compiler into P-code and in a second step this code is loaded into the memory of the SC microcontroller. The interpreter included in the Basic Card operating system is furthermore responsible for the execution of the downloaded code.

Windows for Smart Cards

WSC (Windows for Smart Cards) was Microsofts attempt to create a multi application SC operating system. The idea was to expand the area of application for the Microsoft operating system to SCs and to establish an alternative to Suns JC which was at that time not yet a widely used operating system.

After several different versions of WSC Microsoft cancelled the development of WSC due to a lack of acceptance by the SC industry and offered the source code of WSC to several companies.

Linux

Until now SC microcontrollers have not been powerful enough to reach the requirements of a Linux operating system. Usually these requirements can only be provided by a 32-bit processor, several kilobytes of ROM and even more kilobytes of RAM.

However, it is imaginable that newer Linux versions reduce their hardware requirements and at the same time SCs' microprocessors increase their performance with every new generation. Due to this it would be possible for Linux to be available for SCs in the near future.

1.5 Problem Definition

As mentioned above, the J2ME is the open application environment which is used to integrate the security services for a SC used in a mobile phone. Especially in mobile phones SCs are the most commonly used device to implement a SE. As

discussed in Section 1.3 in the GSM system the SE implemented in the SC is the SIM.

SCs are used to deliver a broad range of security and trust services which are defined in the JSR-177. According to JSR-177 [Gro03] such a SE provides in a J2ME enabled mobile phone the following benefits:

- Secure storage of sensitive data;
- Cryptographic operations;
- Secure execution environment for custom and enabling security features.

Two methods to communicate with a smart card are defined in the JSR-177 specification. The first is the APDU protocol and the second the RMI (Remote Method Invocation) protocol. Due to these two communication protocols is it possible for a J2ME application to utilize the security services provided by the SC.

This work focuses on the APDU communication method. The basic communication between an application being executed in the mobile phone and an application in the SC is shown in Figure 1.1.

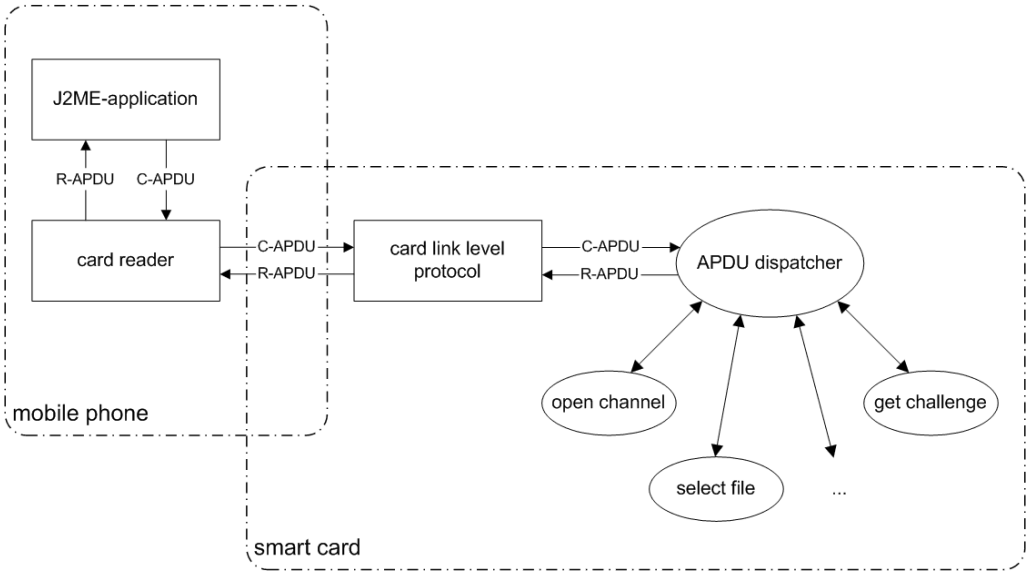

Figure 1.1: Application Communication Structure [GJ02]

The application running in the mobile phone sends step a command APDU to the card reader in the first which is integrated in the phone and the card reader

forwards it to the card. The card processes the request in the APDU dispatcher and responds to the card reader which forwards it to the application in the phone.

A closer look concentrating on a more abstract view of how the data communication between a J2ME-application running in the mobile phone and a JC application running in the SC works, is illustrated in Figure 1.2.

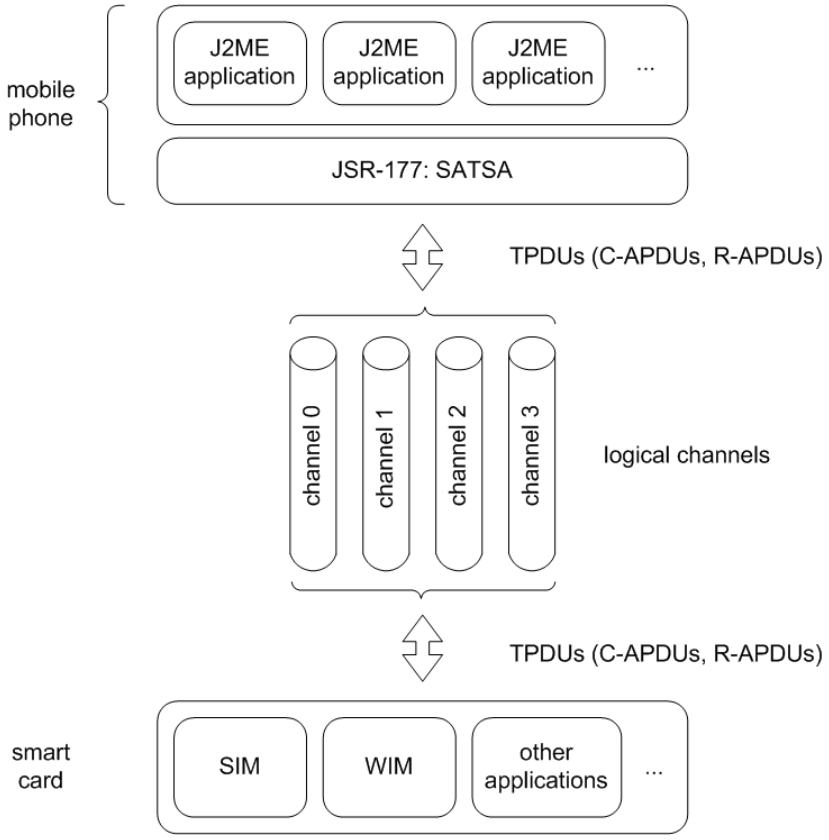

Figure 1.2: Communication Model

In order to be able to communicate with an application on the SC, a connection to the application needs to be established. This is done by opening a logical channel and selecting an adequate application. The number of available logical channels is limited by the SC. Single application SCs have one logical channel (channel 0), multi application SCs have four logical channels (channel 0, 1, 2, and 3).

After opening the connection an APDU can be exchanged with the SC. APDUs can have two different types depending on the direction of the dataflow. APDUs from the application in the phone to the SC are called command APDUs or short C-APDUs. APDUs in the opposite direction are called response APDUs

or short R-APDU.

At the end of every communication the application executed in the phone needs to close the logical channel and after the channel is closed it is again available for other applications.

The goal of the JSR-177 is to add security and trust services to the existing packages. The JSR-177 specifies APIs which enable devices to establish a secure communication.

Therefore the JSR-177 is structured into the following four packages:

- An APDU package which supports the communication with the SC using the APDU protocol;
- A package which defines the JC RMI client;
- A package which supports digital signatures and user credential management;
- A package which provides basic cryptographic operations.

Chapter 2

Smart Card

An ICC (Integrated Circuit Card) or better known under the name SC is a portable, tamper-resistant computer containing a programmable data store [RE04]. The IC embedded in the card has features through which data can be transferred, stored and processed.

2.1 Introduction

SCs and magnetic-stripe cards are the two most popular types of cards and can be found in every wallet. The fact that SCs are more powerful then magnetic-stripe cards is not the only advantage. One of the most important qualities of a SC is the possibility to protect data stored on the card against unauthorized access and manipulation. A SC operating system controls the interface for transferring data between the SC and a connected reader. Due to the usage of cryptographic algorithms and security protocols confidential data can be stored on the card in a way that prevents it from being read from the outside.

2.2 Smart Card Types

SCs can be subdivided into three categories which differ in both functionality and price:

- Cards with surface contacts leading to a memory-only integrated circuit chip - memory cards;
- Cards with surface contacts leading to a microprocessor-integrated circuit chip - microprocessor cards;

- Cards with an electromagnetic connection to a microprocessor-integrated circuit chip - contactless cards.

2.2.1 Memory Cards

Memory cards usually include an EEPROM where applications can store their data. Access control is handled by a security logic which is in the simplest case just a write and erase protection for the memory or special memory areas. More advanced memory cards implement a more sophisticated security logic which is able to perform simple encryption.

As a matter of fact most memory cards have a functionality which is usually optimized for a specific application. On one hand this may seem a restriction of the flexibility of such cards but on the other hand it makes them quite inexpensive. A typical application area for memory cards are prepaid telephone cards.

2.2.2 Microprocessor Cards

Even though microprocessor cards are very flexible, the simplest and a very common case for their usage is to optimize the card for a single application. This means that the card contains just one application.

Another use case integrates a SC operating system onto the card which allows several applications to be loaded onto a single card. The basic functionality of the operating system is stored in the ROM of the SC and the application specific part of the operating system is loaded into the EEPROM after the card has been produced. Modern SCs allow the cardholder to load application programs onto the card as described in Section 2.5 even after the card has been personalized. Recent manufactured SCs use microprocessor chips with high processing and large memory capacities which enable the cards to run complex cryptographic algorithms.

2.2.3 Contactless Cards

Error statistics show that contacts are one of the most frequent sources of failures in electromechanical systems. Contamination, contact wear, and electrostatic discharge rank among the most common problems caused by contacts.

Contactless cards solve the above mentioned failure causes in a very elegant way. Additionally to this advantage contactless cards offer several new attractive applications. Due to the fact that cards do not necessarily need to be inserted into a reader, the coverage of a standard card is about 1 meter, access control

system significantly gain comfort. The usage of the cards is simplified because orientation restrictions are eliminated. Contactless cards are becoming more and more accepted and mass production helps to decrease the price for such cards. To relieve the transition from contact to contactless cards a fourth type of card has been created, so called dual-interface or combicards. These cards have both contact and contactless coupling elements. A popular use case for this card type are public transportation systems where the card is used for identification and authorization purposes.

2.3 Elements of a Smart Card

If we look at SCs an entire computer system consisting of the CPU, the memory, and the Input/Output system is assembled in a single integrated circuit chip. The computer system is illustrated in Figure 2.1.

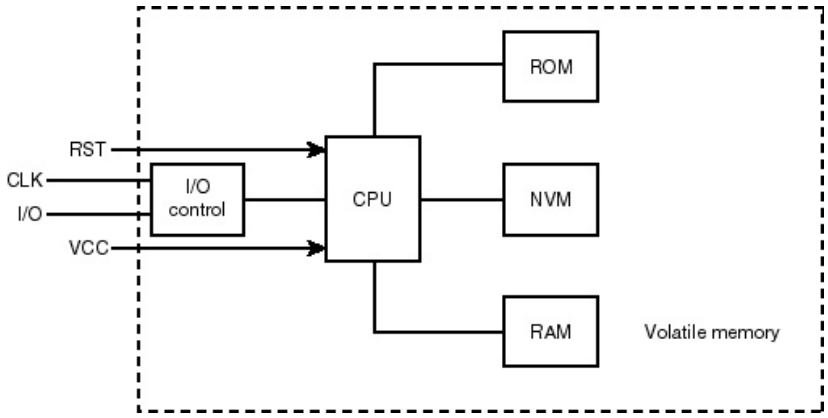

Figure 2.1: Elements of a SC computer system [GJ02]

Because of the simple packaging the system can be integrated in an ICC. Thus the interconnections between the components of the system are hidden and therefore is it difficult to intercept the communication. This enhances the security of the system.

2.3.1 The CPU

The processors used in SCs are usually devices that have been in use in other areas for some time. Typically SCs use a 8-bit processor and the instruction set is based on the Motorola 6805 or the Intel 8051 architecture. These processors have been proven in practice and fulfil the need of high reliability.

However, the demand of processing power needed by the SC operating systems is growing and the development of SC tends to integrate 16-bit or 32-bit processor types.

2.3.2 The Memory System

Memory available on a SC is limited and therefore it must be handled very well. The Figure 2.1 shows the three types of memory on a SC:

- ROM (Read Only Memory)
- NVM (Non-Volatile Memory)
- RAM (Random Access Memory)

ROM

ROM is the memory area where the SC operating system is stored. Common SCs have between 8kB and 96bK of ROM. Code and data are put into ROM during the production process of the SC and are 'hardwired' into the card.

NVM

The name non-volatile memory arises from the fact that once the power is removed the data stays there for about 10 years. The NVM does not exactly act like RAM. Writing the data takes more time and consumes more power than reading. And the NVM wears out after about 100.000 writing operations. A common application is to store data that can be changed by an application (eg. PIN codes, account numbers, account balance of the wallet, ...) in the NVM.

RAM

SCs possess some RAM but usually not very much. Therefore the RAM resources must be handled with much care. A SC software developer needs to be aware of the need to economize on the use of RAM.

2.3.3 The Input/Output System

Data is transferred byte oriented between a SC reader and a card on an unidirectional serial channel at speeds up to 115.200bps. Even if there are several logical channels available just one of them can be active at a time.

The protocol used for communication is based on a master (card reader) and slave (SC) relationship. The card reader sends commands to the SC and waits for a response. SCs never initiate data transfer.

2.3.4 Smart Card File System

Actually the file system in a SC is applied to the non-volatile memory. The files are stored hierarchically and the structure consists of three basic elements which can be seen in Figure 2.2. The SC file system shown in the figure assumes already some kind of application, such as a SC operating system, that takes care of the file management.

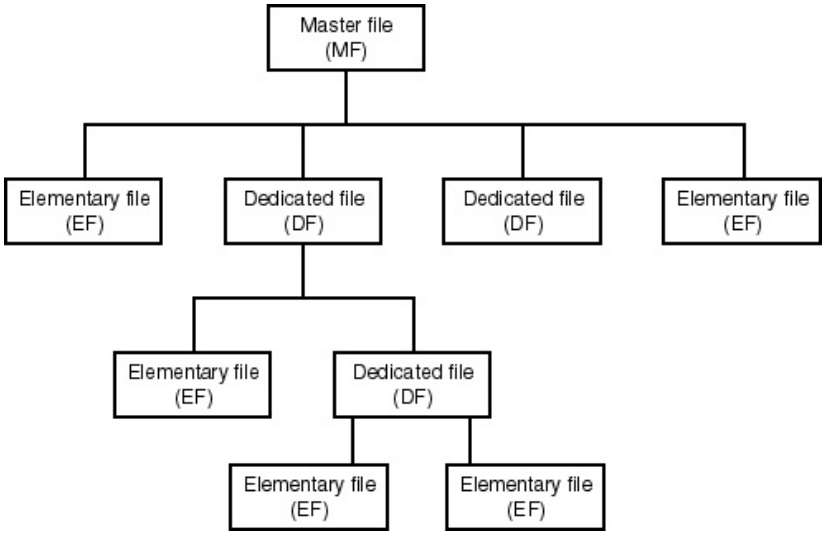

Figure 2.2: The SC file system [GJ02]

Master File Characteristics

Only one MF (Master File) exists on a SC and it is the root of the file architecture. The MF may contain any number of DF (Dedicated File) and EF (Elementary File).

A two byte long file identifier is used to identify each file. The MF can be found under the reserved file identifier $'3F00'_{16}$.

Dedicated File Characteristics

A DF usually forms a subdirectory in the file hierarchy which is rooted by the MF. DFs are also identified by a unique file identifier within the DF or MF that

contains it. This means that the path to the place where the DF is stored is simply the concatenation of the file identifiers.

Elementary File Characteristics

The EFs form the leaves of the file hierarchy and they are the files where the data is actually stored. Basically the two following types of EFs exist: internal EF and working EF. Internal EF are the files used by applications running on the card. Working EF are used by off-card applications.

EFs are again identified by file identifiers with the exception that within a specific DF EFs may be identified by a 5-bit identifier.

Files can be addressed using the *Select File* command which establishes a logical pointer to a specific file on the SC. Through the logical pointer is it possible to manipulate (read, write, delete, etc.) the selected file.

2.4 Multi Application Smart Card

The current standard SC is a SC which contains one application. Such a card is therefore called single application SC. Usually they are created for a special purpose. In case that another application is needed on the SC the entire SC needs to be changed. That means that the SC manufacturer has to produce a SC with the desired application.

The above mentioned disadvantage is probably the main advantage of a multi application SC. A multi application SC allows to load applications onto the card even when the card is in the hands of a card holder. Downloaded applications are stored in the EEPROM of the SC which means that an application can be deleted or replaced every time the card is in use. A further description of how such applications are created and stored on a SC is given in Section 2.5.

2.4.1 Reasons for Multi Application Smart Cards

So far many of the use cases were based on single application SCs. But cards started to be used for smaller and quicker evolving applications and the turnaround time for conventionally manufactured cards and the costs where simply too high. Therefore a number of reasons for the existence of multi application SC exist [GJ02]:

Faster Transaction Times

Off-the-shelf cards implement low level commands. This means that for a complex application many of these low level commands are needed to obtain the wanted functionality. Because of the low data transfer speed between the card reader and the card that can be very time consuming.

This problem can be solved by implementing application specific APDUs which implement the complex functionality wanted and the data can be gained with one data exchange.

Rapid Application Deployment

Nowadays SCs have to compete with other trusted devices such as PDAs, pagers, mobile phones, etc. No matter how secure SCs are, if the turnaround time for a new application is simply to long SCs will loose this competition.

Chip Independence

Multi application SCs have to solve the problem that applications are able to be executed on different chip hardware. This leads to the benefit that card issuers and their applications become more independent from the underlying hardware and its suppliers.

2.4.2 Application Identifier

The AID (Application Identifier) is specified in the ISO 7816-5 standard, which defines a universal name space for SC applications. An AID consists of two parts.

The first element of the AID is the RID (Registered Application Provider Identifier), which has a fixed length of 5 bytes. RIDs are assigned by a registration authority and include a country code, an application category, and an application specific number which refers to the application provider. RIDs are unique and can be used world wide to identify a certain application.

The second part of the AID is a variable length field of up to 11 bytes called the PIX (Proprietary Application Identifier Extension). The PIX is used by application developers to identify specific applications. The PIX name space is handled by the application developer and the application developer has to make sure that the concatenation of RID and PIX uniquely identify an application.

2.5 Open Platforms

Generally the term 'open platform' refers to SC operating systems that allow third parties to load applications onto SCs independent of the card manufacturer.

Visa International[1] was a pioneer at working with multi application SCs. Visa defined an interface inside SC operating systems for managing SC applications. Later on a Global Platform Committee[2] was formed whose function is to standardize technologies for multi application SCs.

The 'Open Platform' specification is the most important specification for application management in multi application SCs. The specification is independent from any SC operating system, but in practice it is the standard which is used for loading and managing Java based applications with the JC operating system.

2.5.1 Java Card

For a detailed discussion of Java Card[3] see Section 3.2.

2.5.2 MULTOS

MULTOS[4] is a SC operating system that provides the underlying communications, memory management, and an AAM (Application Abstract Machine) for multi application SCs. MULTOS applications are executed in MEL. MEL is a byte code language normally written using assembly language notation, but applications can also be written in higher level languages like C or Java and being compiled into MEL byte code. The AAM provides every application stored on the card with its own memory space to hold code and data [MUL03].

One public memory area exists in every card, otherwise applications are not able to access the memory of other applications. Figure 2.3 illustrates the MULTOS architecture.

While an application is loaded onto the SC, MULTOS checks its validity and allocates a memory area protected by a firewall. Each application is strictly stored separated from the other applications and it is not possible for them to interfere with each other.

MULTOS allows applications being loaded onto the SC even when it is already in the card holders' hands. That means a SC with MULTOS can change features,

[1] `http://www.visa.com/`
[2] `http://www.globalplatform.org/`
[3] `http://java.sun.com/products/javacard/` and `http://www.javacardforum.org/`
[4] `http://www.multos.com/`

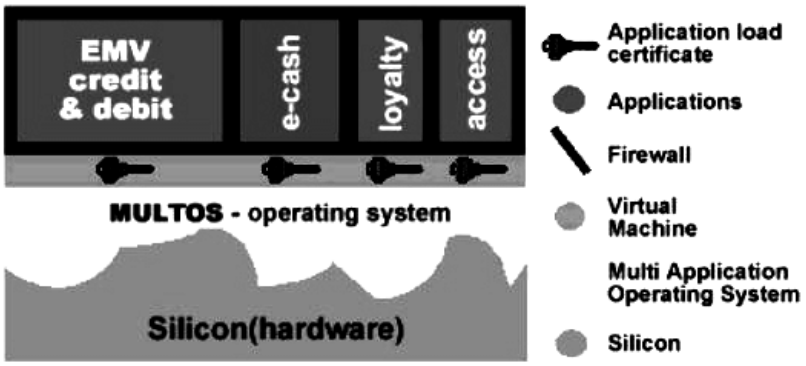

Figure 2.3: MULTOS system architecture [Tec04]

the contained applications, during its lifetime. Furthermore, applications can be loaded onto the card through encryption and signing. This provides a complete end to end security solution for application loading.

2.5.3 Basic Card

The BasicCard[5] is a programmable microprocessor card with a P-code interpreter written in Basic. The P-code generated by the Basic Card compiler differs from the one used in Java. The Basic Card design is based on four main criteria [Gui03]:

- An inexpensive solution for SC software developers (the development software is free of charge);
- Easy to program due to the fact that Basic is an easy and well-known programming language;
- State-of-the-art cryptographic algorithms provide a secure developing environment (RSA, AES, SHA-1);
- Basic Card is designed according to the ISO/IEC 7816 specifications.

In comparison with other SC operating systems with an integrated interpreter, Basic Card has very compact program code and relatively high execution speed. Programs can be quickly and easily developed and Basic Card can certainly represent an alternative to other SC operating systems.

[5] `http://www.basiccard.com/`

2.6 Data Transmission

Data transfer (defined in the ISO/IEC 7816 specification) between the card reader and the card takes place on a single line, a so called half-duplex connection. The card reader and the card have a master and slave relationship. This means that data exchange is always initiated by the host (e.g. a card reader) and never by the client, a SC. The card receives the command (C-APDU) from the reader, executes it and responds to the card reader by sending a R-APDU.

2.6.1 Answer to Reset

Once the SC is connected to the power supply, it automatically executes a power-on reset and sends an ATR (Answer to Reset) to the card reader. This data string can have a maximum length of 33 bytes and can contain the data elements listed in Table 2.1 [RE04].

Data element	Designation
TS	Initial character
T0	Format character
TA1, TB1, TC1, TD1, ...	Interface characters
T1, T2, ..., Tk	Historical characters
TCK	Check character

Table 2.1: The data elements of the ATR and their designitions

Initial Character

The TS character is the first byte of the ATR and it is mandatory. Only two codes are allowed for this byte: $'3B'_{16}$ for direct convention and $'3F'_{16}$ for inverse convention.

Format Character

T0 is the name of the second ATR byte. The upper nibble indicates which interface character follows T0 and the lower nibble declares the number of historical characters included in the ATR. As the TS the format character is mandatory.

Interface Character

The interface characters are responsible for all the transmission parameters of the current protocol. They are optional bytes in the ATR and it is often not necessary to transmit them, since default values are defined for all of the parameters of the transmission protocol.

Historical Character

Historical characters contain a wide variety of information. SC operating system producers use them to identify the operating system and further hardware specific data. A maximum number of 15 characters is possible. Like interface characters historical characters are not required in the ATR.

Check Character

The final character in the ATR is the TCK which contains the XOR checksum of the bytes from T0 to the last byte before the TCK. It can be used in addition to parity testing to verify the correctness of the ATR transmission.

2.6.2 Data Transmission Protocols

The ISO/IEC 7816-3 standard specifies a total number of 16 protocols. The protocols are named 'T=' plus a sequential number (0-15). In practice four protocols are of interest. The T=0 and the T=1 are predominate in international use. T=2 is still in the design process and a standard will be available in a few years. T=14 is a protocol for national use. In Germany the 'Deutsche Telekom' uses such an internal specification of the T=14 protocol.

The T=0 protocol is the only protocol specified in the GSM standard and hence it is the relevant protocol for the ongoing discussion.

T=0 Transmission Protocol

The T=0 was the first internationally standardized SC transmission protocol and the design rules were minimum memory usage and maximum simplicity. It is the protocol used in GSM cards and therefore the most widely used protocol.

The protocol is byte orientated and each TPDU (Transmission Protocol Data Unit) consists of two parts, a header and a data part. The header consists of the 5 bytes listed in Table 2.2.

Name	Description
CLA	specifies a collection of instructions
INS	specifies a specific instruction
P1	specifies the addressing
P2	also used to specify the addressing
P3	specifies the number of data bytes transferred to or from the card

Table 2.2: T=0 header structure

The error detection mechanism is simple and consists only of a parity check at the end of each byte. On detection of an error it is demanded to resend the byte.

The card responds to the header with so called procedure bytes which include three or four bytes shown in Table 2.3.

Name	Description
ACK	indicates the reception of the [CLA, INS] command
NULL	used to do flow control on the I/O channel
SW1	status information regarding the current command
SW2	status information regarding the current command (optional)

Table 2.3: T=0 procedure bytes

2.6.3 Application Protocol Data Units

The internationally standardized data unit for the data exchange between the card reader and the smart card is called APDU. A distinction is made for different purposes of APDUs. An APDU used in the transmission protocol layer is called TPDU. TPDUs are subdivided at the application protocol layer into two types of APDUs, namely command APDUs (C-APDUs) and response APDUs (R-APDUs). APDUs can be understood as boxes which either contain a command sent from the card reader to the card or a response from the card to the card reader.

Command APDU

Every C-APDU has two elements, a header and a body. The length of the header is fixed to 4 bytes, the length of the body varies, depending on the amount of the

included data. Table 2.4 shows all the bytes included in the C-APDU [Ins03].

Code	Length (byte)	Description	Grouping
CLA	1	Class of insturction	Header
INS	1	Instruction code	Header
P1	1	Instruction parameter 1	Header
P2	1	Instruction parameter 2	Header
Lc	0 or 1	Number of bytes in the command data field	Body
Data	Lc	Command data string	Body
Le	0 or 1	Maximum number of data bytes expected in response to the command	Body

Table 2.4: Contents of a command APDU

Lc and Le are the abbreviations for 'length command' and 'length expected'. For further details about the coding of the bytes in the C-APDU see [Ins03].

Generally four C-APDUs with different contents are possible. Table 2.5 illustrates the cases of C-APDUs.

Case	Structure	Length (byte)
1	CLA - INS - P1 - P2	4
2	CLA - INS - P1 - P2 - Le	5
3	CLA - INS - P1 - P2 - Lc - Data	variable
4	CLA - INS - P1 - P2 - Lc - Data - Le	variable

Table 2.5: Cases of C-APDUs

Response APDU

A R-APDU is composed of a body and a trailer. The body is optional and the trailer mandatory. Two R-APDU types are available consisting either out of a body and a trailer or just a body. Table 2.6 shows the elements of a R-APDU.

The length of the response data is specified in the preceding C-APDU, but regardless of the number of bytes indicated by the Le byte the length can be 0 if the SC terminates the process caused by an error or invalid parameters in the C-APDU. The status words SW1 and SW2 provide the processing result of the C-APDU execution. Figure 2.4 shows the basic classification scheme for the

status words and further details can be found in [Ins03].

Code	Length (byte)	Description	Grouping
Data	Lr[6]	Response data string	Body
SW1	1	Status byte 1	Trailer
SW2	1	Status byte 2	Trailer

Table 2.6: Contents of a response APDU

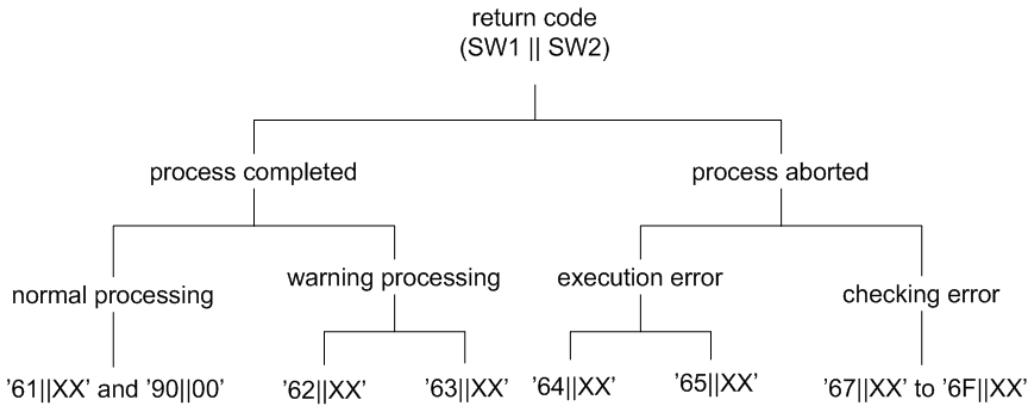

Figure 2.4: Return code classification scheme (ISO/IEC 7816-4)

Status bytes containing $'63||XX'_{16}$ or $'65||XX'_{16}$ indicate that the non-volatile memory of the SC has been altered. Other status words starting with $'6X'_{16}$ indicate a premature termination of command execution without altering the non-volatile memory. $'90||00'_{16}$ are the status bytes for successful processing.

Even though a standard for return codes exists, many applications define their own non-standard codes.

2.6.4 Logical Channels

In a multi application SC several applications can be stored on the card. Logical channels make it possible to address up to four applications at the same time. Physically there is still the single serial interface, but on a logical level it is possible to have four connections. The class byte (CLA) in the C-APDU specifies in the two least significant bits (Bit 1 and Bit 2) the logical channel for which the C-APDU is dedicated. One major limitation for the communication on the logical channel exits. The external process and the application on the SC must

[6]Length of the response data field.

be mutually synchronized and it is not allowed to interfere a communication in progress, since the R-APDU does not include any information about the logical channel. Therefore it is not possible to figure out to which C-APDU the received R-APDU belongs.

One possible application for logical channels could be the following scenario. A business man is using the GSM application in his multi application SC which is inside his mobile phone. While he is talking, he wants to confirm an appointment. Therefore he executes the calendar application likewise stored on his SC. Using a second logical channel he searches in his calendar for the desired day and is now able to provide the information about confirming the appointment.

Logical channels produce several management requirements for the SC operating system. Every logical channel can be seen as a separate SC and a lot of memory is necessary to store all the information and conditions.

2.7 Security in Smart Cards

SCs compared with other data storage media have one major advantage. Data is stored, protected, and kept secret. What makes a SC secure? Four components are responsible for the security of a SC. The card body which holds the microcontroller, the chip hardware, the operating system and the applications. The task of the three last mentioned components is to protect programs and data in the SC microcontroller.

The SC operating system and the applications on the SC need to be able to handle a variety of cryptographic elements. These elements are the elements of the following discussion.

2.7.1 Physical Security

As mentioned in the introduction of this chapter a SC is a tamper resistant computer. Central for the concept of tamper-resistance is to guarantee the physical security by packaging the ICC and its connections into a module that is enchased in an epoxy resin. Figure 2.5 shows the elements and the physical structure of a SC.

The term tamper resistance refers to the characteristic, that it is a nontrivial task to get access to the ICC and even more complicated to extract information from the chip. Detailed knowledge of the systems architecture and the software is needed. Furthermore, the necessary equipment is reasonably expensive and a

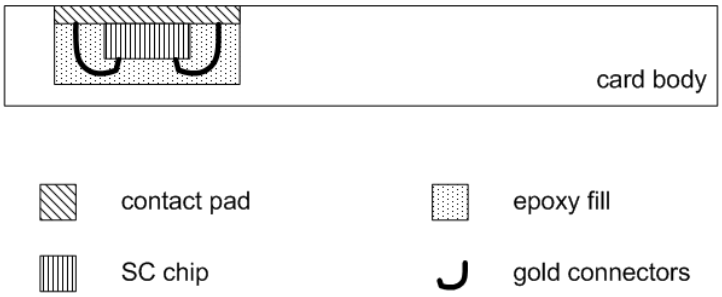

Figure 2.5: The physical structure of a SC

physical attack typically leaves an obvious track.

Additionally it is difficult to attack a SC without the cardholder knowing that the card is no longer in his/her possession.

2.7.2 Cryptographic Algorithms

Cryptology combines the two disciplines cryptography and cryptoanalysis. Cryptoanalysis goes hand in hand with encryption/decryption. Only few algorithms fulfil the requirements of cryptoanalysis and the limitation of processing power and memory in the SC environment. Therefore the number of cryptographic algorithms used in SCs is very small.

Symmetric Cryptographic Algorithms

The basic principle for symmetric cryptographic algorithms is to use the same secret key for data encryption and decryption. The most common symmetric algorithm is the DEA (Data Encryption Algorithm) and it is defined by the DES (Data Encryption Standard). The DEA encrypts and decrypts 64-bit blocks with a single key, typically 56 bits long. The computation is quite simple and can be performed on slow processors like the ones available in SCs.

Several newer algorithms have been suggested like Twofish[7], IDEA (International Data Encryption Algorithm), and AES (Advanced Encryption Standard). Only the last mentioned algorithm is generally available in SC microcontrollers and has evolved to a rival of the DEA.

[7] An algorithm that was especially designed for the usage in low power processors.

Asymmetric Cryptographic Algorithms

In contrast to the symmetric algorithms, asymmetric cryptographic algorithms use different keys for encryption and decryption. The systems are based on two keys, a public and a private key pair. The public key is publicly available. The private key needs to be kept secret by its owner. Even though there are several other algorithms available the most common asymmetric algorithm is RSA (Rivest Shamir Adelman). The execution of the RSA algorithm is slow on byte orientated processors, like the 8-bit processors used in many SCs.

2.7.3 Key Management

The key management main goal is to minimize the consequences for the system and the SC application if one or more secret keys get compromised by a unauthorized person. Not all the security principles need to be implemented, this would take up too much memory. In fact just the necessary methods should be added to the system [RE04].

Master Keys and Derived Keys

SCs can easily be taken away and it is therefore more likely that they are exposed to the most severe attacks. Even if somebody breaks the security mechanisms and reads the contents of the card the found keys are just those derived from a master key.

Derived keys are unique and they are usually created out of card specific features[8] and a master key using a cryptographic algorithm (DES, AES).

Key Diversification and Key Versions

Usually different keys are used for every cryptographic operation in the SC to reduce the damage in the case that somebody breaks a key. For each type of key a separate master key must exist to generate the needed derived keys.

In practice more than one key generation is stored in the SC in order to change the version of the keys in case that one or more keys get compromised. But a change of a key generation is not always due to an intrusion of an unauthorized person. Normally a version change takes place at fixed or variable intervals.

[8] Usually the card number is the card specific feature.

Dynamic Keys

Dynamic keys are another common security practice used in the area of data transmission. So called 'session keys' or 'temporary keys' are generated and used for data exchange. These keys are generally created from a random number which is sent to the other party.

Key Parameters

To be able to use the keys stored in the SC they need a special key number. The operating systems' task is to ensure that a specific key can only be used for the purpose is was created for. To distinguish between different versions of a key, a version number is additionally needed. Table 2.7 lists typical key parameters stored on a SC [RE04].

Data Object	Remarks
Key number	Key reference number, unique within the key file
Version number	Version number of the key
Application purpose	Identifies the cryptographic algorithms and the procedures with which the key may be used
Disable	Allows the key to be temporarily or permanently disabled
Retry counter	This counter keeps track of non successful attempts to use the key with a cryptographic procdure
Maximum retry counter	If the retry count reaches a maximum count, the key is blocked
Key length	Length of the key
Key	The actual key

Table 2.7: Typical key parameters stored in a SC

2.7.4 Authentication

Authentication is used to ensure that the communication partner is a genuine SC reader or a genuine SC. For this purpose they need to share a common secret that can be verified. By using an authentication procedure it is not possible to discover the secret by eavesdropping the communication because it does not need to be sent openly over the communication channel.

Looking at a mobile phone the mobile phone user needs to enter the PIN (Personal Identificaion Number) during the start-up of the phone. The SC verifies the PIN and permits access to the phone if the verification was successful.

2.7.5 Digital Signatures

Digital signatures serve to establish the authenticity of transferred data and verification of the signature determines whether the data has been altered or not.

The main feature of digital signatures is that only one device can produce it correctly, but it can be verified by any device which receives the signature.

2.7.6 Certificates

In order to be able to verify a digital signature the corresponding public key is needed. The public key cannot be sent without any protection because it is necessary that the receiver of the public key can verify the authenticity of the key. Therefore the public key must be signed by a trustworthy third party called a CA (Certification Authority).

A certificate is the combination of a public key signed by a CA, the attached digital signature of the CA and certain additional parameters. It identifies the owner of a public key.

2.8 Telecommunications and Smart Cards

The SCs used in GSM mobile phones are better known under the name SIM. The SIM was and still is a pioneer when it comes to functionality and memory capacity among SCs.

In 1992 the successful story of GSM started and within a few years it became the standard for mobile telecommunication systems.

2.8.1 Subscriber Identity Module

A mandatory element of a GSM mobile phone is the SIM. The following definition of the SIM can be found in the GSM 02.17 specification [Ins99].

> *The SIM is the entity that contains the identity of the subscriber. When placed in a* ME*, together they become a* MS *(Mobile Station) which may then register onto a GSM network. The primary function of the SIM in conjunction with a GSM network is to authenticate the*

validity of a MS when accessing the network. In addition it provides a means to authenticate the user and may also store other subscriber-related information or applications.

The SIM has two main purposes. First it ensures access to a particular GSM network and second it associates the use of the network with a payment account. The basic functionality is illustrated in Figure 2.6.

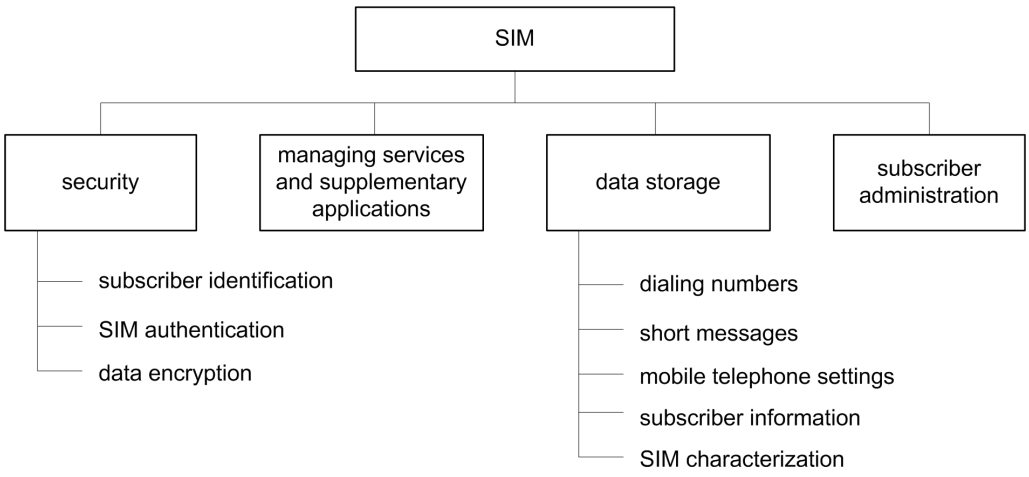

Figure 2.6: Basic functionality of the SIM in the GSM system [RE04]

If the SIM is incorporated into a multi application SC the SIM is permitted to contain non-GSM functionality.

2.8.2 SIM Application Protocol Data Units

The GSM 11.11 specification defines 22 APDU commands which can be executed on the SIM card. The commands are split up into 4 different command subgroups, namly security commands, file operation commands, SAT commands, and miscellaneous commands. Table 2.8 summarizes the commands and gives a brief description of their functionality.

2.8.3 The SIM Application Toolkit

As a consequence of the wide acceptance of the GSM standard it became necessary to expand the SIM functionality and add supplementary services (e.g. checking the balance of the bank acount, news, sport results, ...). The answer to these new requirements is the SAT [rGPP04].

Command	Brief description
Security commands	
Change CHV[9]	Change the PIN
Unblock CHV	Reset the PIN retry counter
Verify CHV	Verify the PIN
Run GSM Algorithm	Execute the GSM specific authentication algorithm
File operation commands	
Select	Select a file
Status	Read various data form the currently selected file
Read Binary	Read from a file with a transparent structure
Read Record	Read from a file with a record-orientated structure
SAT commands	
Envelope	Pass data to a value added service of the SIM
Fetch	Retrieve a SAT command from the SIM in the ME
Terminal Profile	List all functions of the ME with respect to the SAT
Terminal Response	Convey the response of the ME to a previous SAT command
Miscellaneous commands	
Get Response	Command specific to T=0 protocol to request data from the SC
Sleep	Obsolete command for putting the SC into a low power state

Table 2.8: Summary of SIM Application Protocol Data Units specified by GSM 11.11

> *The SIM Application Toolkit (SAT) provides mechanisms which allow applications, existing in the SIM, to interact and operate with any ME which supports the specific mechanism(s) required by the application.*

To enable the SIM to start a communication with the ME a simple mechanism is used. The ME sends in a certain time interval a 'Status' command to the SIM and if the SIM responds with a special status word $'91||XX'_{16}$ the ME knows that there is a command ready to be fetched from the SIM. The commands which make it possible to inverse the master slave relationship between the ME and the SIM are listed in Table 2.8 in the 'SAT commands' group.

[9] CHV (Card Holder Verification)

2.8.4 Wireless Identity Module

The WIM (Wireless Identity Module or WAP Identity Module) is a security module on the SC which can be used to establish a secure communication using the WAP. It is not necessary for server authentication as it is allowed to store public keys on the mobile phone. But a client authentication without the WIM is not possible because the necessary private keys are stored on the SC. Further it is used to store and process information needed for user identification and authentication.

Sensitive data can be stored in the WIM (e.g. keys) and all operations where this data is involved can be performed in the WIM (compute digital signature, verify signature, encipher, decipher, derive key, ...) [For01].

Chapter 3

Used Java Technologies

The new mobile phone generations enlarged their functionality and they are becoming more and more open to the outside world. This means that developers can design and create software which can be loaded onto mobile phones. The J2ME[1] technology for mobile phones is based on the Java[2] technology from Sun Microsystems.

J2ME combined with its related technology suitable for SCs, Java Card, makes a complete open development environment available for the development of applications for both devices, the mobile phone and the SC.

3.1 Java 2 Micro Edition

The J2ME platform is a set of standard Java APIs defined through the JCP program by expert groups that include leading device manufacturers, software vendors, and service providers.

J2ME delivers the power and benefits of Java technology tailored for consumer and embedded devices including a flexible user interface, a robust security model, a broad range of built-in network protocols and support for networked and disconnected applications. With J2ME, applications are written once for a wide range of devices, are downloaded dynamically and leverage each device's native capabilities [Mic02a].

[1] `http://java.sun.com/j2me/`
[2] `http://java.sun.com/`

3.1.1 Introduction

The J2ME was first announced in June 1999 at the JavaOne Conference. Not just the announcement of the J2ME was of interest but also the introduction of a completely new VM which makes it possible to run Java applications on small computing devices, the KVM (K Virtual Machine).

The graphic in Figure 3.1 shows the different Java technologies and marked by a brace the J2ME architecture, which is divided into three main parts:

- Configurations
- Profiles
- Optional Packages

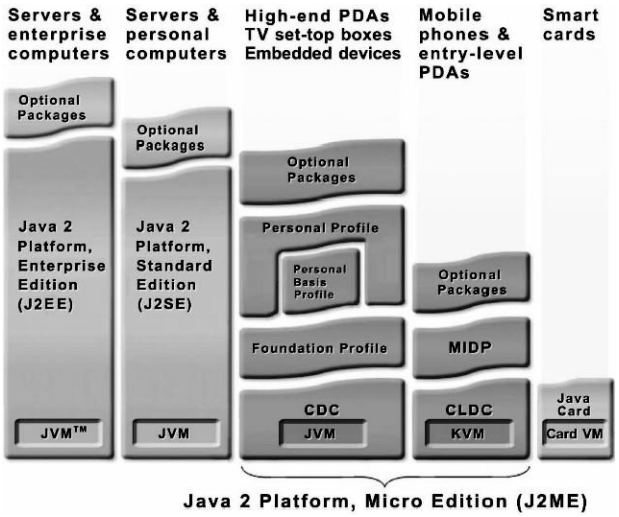

Figure 3.1: J2ME platform architecture [Mic02a]

One of the main problems which needed to be solved is to downsize the amount of runtime classes installed with the runtime environment. This goal is reached by removing all unnecessary classes from the set of core classes. The J2ME runtime environment consists of a true subset of the J2SE (Java 2 Standard Edition).

To only reduce the number of runtime classes is not particularly useful, especially when it comes to the interaction with other devices or with the user. Therefore new classes have been defined, which partly replaced similar classes from the J2SE.

3.1.2 Configuration

Configurations are used to define a minimum of JVM features and a minimal set of class libraries. The idea is to provide the base functionality for a particular range of devices that share similar characteristics (network budget, hardware capabilities, ...). In more detail a configuration specifies which Java programming language features, which JVM features, and which basic Java libraries and APIs are supported.

The J2ME specification defines two configurations [Mic02a]:

- CLDC (Connected Limited Device Configuration)
- CDC (Connected Device Configuration)

The CLDC is designed for devices with intermittent network connection, slow processors, and limited memory. Mobile phones are such devices and they typically have a either 16- or 32-bit CPU and a minimum of 128kB to 512kB of memory.

The CDC is designed for devices with faster processors, more memory, and greater network bandwidth.

The graphic in Figure 3.2 illustrates the relationship between the two J2ME configurations and the J2SE.

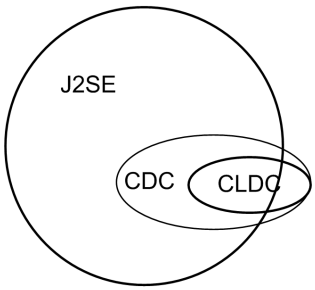

Figure 3.2: Relationship between the J2ME configurations and the J2SE [Mic00]

Due to the fact that mobile phones are part of the discussed subject the CLDC is the configuration of interest and further information about the CDC can be found at `http://java.sun.com/products/cdc/`.

CLDC is available in two different versions, CLDC Version 1.0 (May 2000, JSR-30) and CLDC Version 1.1 (March 2003, JSR-139). The ongoing discussion

focuses on the last mentioned which by the way provides backward compatibility to version 1.0.

The Connected Limited Device Configuration

The CLDC device family aims to keep the footprint as small as possible, to focus on application programming rather than on system programming and to enable dynamic downloading of applications and encourage third parties to develop applications. Generally CLDC devices have the following characteristics in common [Mic03a].

- At least 192kB (160kB non-volatile and 32kB volatile) of total memory budget available for the Java platform.
- A 16-bit or 32-bit microprocessor.
- Low power consumption, often battery powered.
- Connectivity to some kind of network, often wireless with limited bandwidth.

Architecture The architecture of a typical J2ME device can be divided into three main elements which are laying on top of each other.

The lowest element is the host operating system which is integrated by the manufacturer in the CLDC device. On top of the host operating system runs the KVM which holds all native functionality.

The element placed next are the Java class libraries which are besides the KVM, the second part of the CLDC. Some of the libraries are defined by the CLDC itself, but the J2ME profiles may define additional libraries and features which are situated on top of the CLDC.

Security Generally the J2SE provides developers with a powerful and flexible security framework. Unfortunately it is not possible to make this security framework available on the J2ME platform due to memory restrictions. Therefore the CLDC security model needs to be simplified and is defined at the three following different levels [Mic03a].

Low-level Security Executed applications, which are running in the KVM are not allowed to harm or crash the CLDC device on which the KVM is running. This constraint is guaranteed by the bytecode verifier, which controls the bytecode

and other items stored in the class files in order to make sure that they do not contain illegal instructions, that they cannot be executed in the wrong order and that they do not contain references to invalid memory areas.

Application-level Security Although the bytecode verifier is important it is not enough for security. There are several other security threats that need to be taken care of. The application-level security pays attention to limit the access to libraries which are accessible in the Java runtime environment.

This requirement is fulfilled by the so called "sandbox". An application must run in a closed environment which limits the accessible libraries by defining them in the configuration and the profile.

Dynamic downloading is central for the functionality of the CLDC. Therefore it is not allowed for downloaded applications to override or extend the system classes of the KVM.

Furthermore, the CLDC ensures that the applications can only load application classes which are included in their own JAR (Java Archive) file. This mechanism makes it impossible for applications to interfere or steal data from each other.

End-to-end Security CLDC devices usually belong to an end-to-end solution such as a mobile phone network. Normally special security solutions are necessary to ensure a safe transfer of data between devices connected to the network. The CLDC is not responsible for these security solutions.

3.1.3 K Virtual Machine

In the beginning KVM stood for Kuaui Virtual Machine but was later changed to KiloByte Virtual Machine. Today, however, it is more popularly known as the K virtual machine. KVM is a completely new implementation of a JVM (Java Virtual Machine), an implementation optimized for the use on small devices.

Generally the J2ME supports two VMs, the standard VM JVM and the KVM. But for the ongoing discussion only the KVM is of importance. The KVM was designed according to the following criteria [Mic00]:

- Small, with a static memory footprint of the virtual machine core in the range of 40kB to 80kB (depending on compilation options and the target platform;
- Clean, well-commented and highly portable;

- Modular and customizable;
- As "complete" and "fast" as possible without sacrificing the other design goals.

Restrictions caused by these design criteria are the fact that it is not possible to add native methods at runtime. All the native functions are built into the KVM. Further the KVM does only include a subset of the standard bytecode verifier, which makes it impossible to verify the code in the device. The verification process needs to be split between the CLDC device and the external connected device.

3.1.4 Profile

For a working runtime environment, profiles need to be added to the configurations. A profile defines a set of classes and APIs available for a specific device. Applications are written for profiles and are portable to every device which supports this profile.

Four different profiles are available for the J2ME platform:

- MIDP (Mobile Information Device Profile);
- Foundation Profile;
- Personal Profile;
- Personal Basis Profile.

As illustrated in Figure 3.1 the MIDP belongs to the CLDC and the foundation, the personal, and the personal basis profile to the CDC.

The MIDP is designed for mobile phones and offers the basic application functionality, including the user interface, network connectivity, local data storage and application management. The combination of the CLDC and the MIDP provides a complete Java runtime environment for a mobile phone.

MIDP is available in two different versions, MIDP Version 1.0a (December 2000, JSR-37) and MIDP Version 2.0 (November 2002, JSR-118). The further discussion will focus on the last mentioned version which provides backward compatibility with version 1.0a.

The Mobile Information Device Profile

The general design goal for the MIDP was to define an open development environment to enable third party application development for a MID (Mobile Information Device).

MIDs are small hand-held devices with very limited capabilities and with the user interaction as a key focus. Therefore the following minimum characteristics need to be satisfied [OG01].

- Monochrome or color display with a minimum size of 96x54 pixels and a 1:1 aspect ratio.
- One of the following input mechanisms: "one-handed" keyboard, "two-handed" keyboard or touch screen.
- A two-way wireless network.
- At least 128kB of non-volatile memory for the MIDP components, 8kB for the application-defined storage and 32kB of volatile memory for the Java runtime environment.

Architecture The architecture designed by the MIDPEG (Mobile Information Device Profile Expert Group) is illustrated in Figure 3.3. Note that not every device which implements the MIDP specification has all the elements presented.

The base is formed by the MID block which represents the hardware of the mobile information devices. Placed on top of the hardware is the native system software. This part includes the operating system and the libraries used by the MID.

The CLDC is the next element and it is another layer of software, which consists of the KVM and the libraries defined by the CLDC specification. This block is responsible for the underlying Java functionality on which higher-level Java applications may be built on.

Finally the top level of the architecture is formed by three categories of APIs, the MIDP, the OEM (Original Equipment Manufacturer) specific and the native APIs.

Security The security design basically distinguishes between a trusted and an untrusted application, a so called MIDlet (Mobile Information Device (app)let).

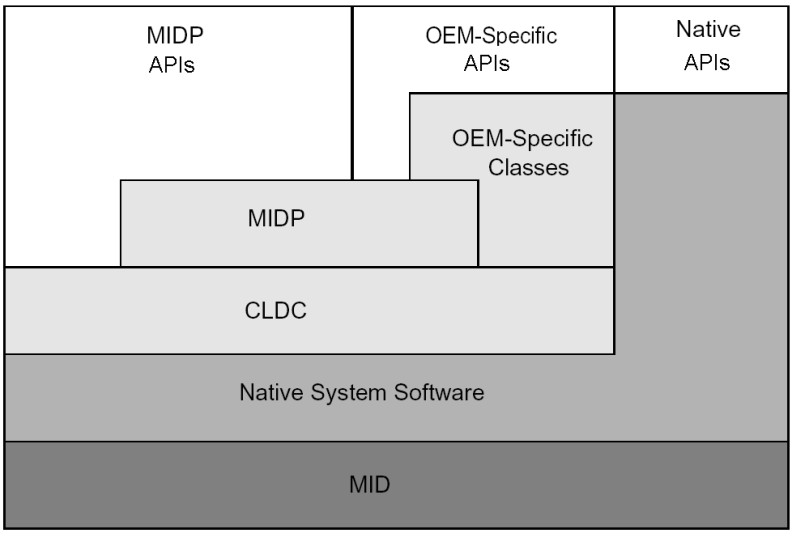

Figure 3.3: MIDP architecture [Mic02b]

Several MIDlets can be put together and form a MIDlet suite (Mobile Information Device (app)let suite).

In order to guarantee a secure execution of a MIDlet, depending on if it is trusted or not, two different concepts exist. The concept for untrusted MIDlet suites uses the standard Java "sandbox" to prevent access to sensitive APIs or functions of the MID. The "sandbox" model of a J2ME enabled device like a mobile phone is shown in Figure 3.4.

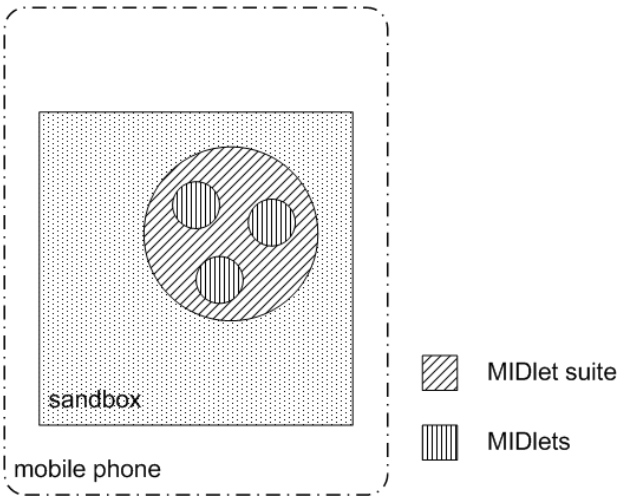

Figure 3.4: J2ME "sandbox"

Trusted MIDlet suites gain access to APIs that are considered sensitive and restricted. The definition to which APIs the MIDlet suite gets access to is made via domains.

Domains specify groups of APIs, one for APIs which are allowed to be used by untrusted MIDlet suites and another for APIs which usage is limited to trusted MIDlet suites.

The untrusted domain must allow a MIDlet suite access to their APIs without an explicit confirmation of the user. Whereas the trusted domain needs an explicit confirmation of the user to permit access.

The security for trusted domains is based on protection domains. A protection domain is responsible for the permissions that may be granted to a MIDlet suite in that domain. The owner of the protection domain defines the mechanism for identification and verification of a MIDlet suite. The way the MID identifies and trusts a MIDlet suite can be selected for the appropriate case.

Protection domains are usually related to certificates. The mobile phone network defines three groups of root certificates. The certificates are used for signing the MIDlet suites and for authentication and authorization. One root certificate is for the handset producer and one for the network operator. The last group are third parties and this group can have more than just one root certificate.

3.1.5 Optional Packages

Shortly after the launch of the CLDC and the MIDP it was emphasized that there is a need for additional general purpose libraries that are not included in one of the configurations or profiles. Optional Packages are an attempt to close this gap and they are used to extend configurations and their corresponding profiles. They are created to address specific requirements and offer standard APIs for both existing and emerging technologies.

In general an optional package is an API or a collection of APIs that can be used to extend a profile. Due to the fact that optional packages are modular, MID manufacturers can include the APIs needed to fully leverage the features of each device.

A more detailed description of an optional package can be found in Chapter 4.

3.2 Java Card

JC[3] is a software development environment for SCs and other devices with limited memory and it enables these devices to securely run small applications, so called JC applets. JC applets are written in Java.

JC offers a secure and interoperable execution platform able to store and update multiple applications on a single SC.

3.2.1 Language Subset

Due to the need to downsize the Java development environment to make it small enough to fit it on a SC with its limited memory only a carefully chosen, customized subset of the Java features is available on the Java Card platform.

The supported features focus on the aspect of writing applications for SCs and small devices preserving the object oriented capabilities of Java.

3.2.2 Architecture

To solve the problem of using too much space on the SC for the JC environment the JCVM (Java Card Virtual Machine) is separated into two parts. The first part contains all the features which are necessary to execute an applet on the SC. This part runs on the card. The second part runs off the card and takes care of all the processing tasks which are not required at runtime. Class loading, byte code verification, resolution and linking, and optimization are such tasks.

Generally the JC technology is formed by the following three elements [Che00].

- The Java Card Virtual Machine
- The Java Card Runtime Environment
- The Java Card Application Programming Interface

3.2.3 Virtual Machine

The JCVM consists of two elements. The first element is running off-card on a workstation and the second element on the SC.

The part placed on the SC implements the byte code interpreter and is therefore responsible for the execution of an applet. The off-card part contains the

[3] `http://java.sun.com/products/javacard/` and `http://www.javacardforum.org/`

converter which processes the class and export files, and converts them into a CAP (Converted Applet).

After converting the file the produced CAP file can be loaded onto a SC and is ready to be executed by the interpreter.

3.2.4 Runtime Environment

The JCRE (Java Card Runtime Environment) is built out of four elements. The first element is the on-card part of the JCVM, the byte code interpreter. The JCRE system classes are the second element followed by JC application framework classes and industry specific extensions as the elements three and four. Figure 3.5 illustrates the JCRE and its elements.

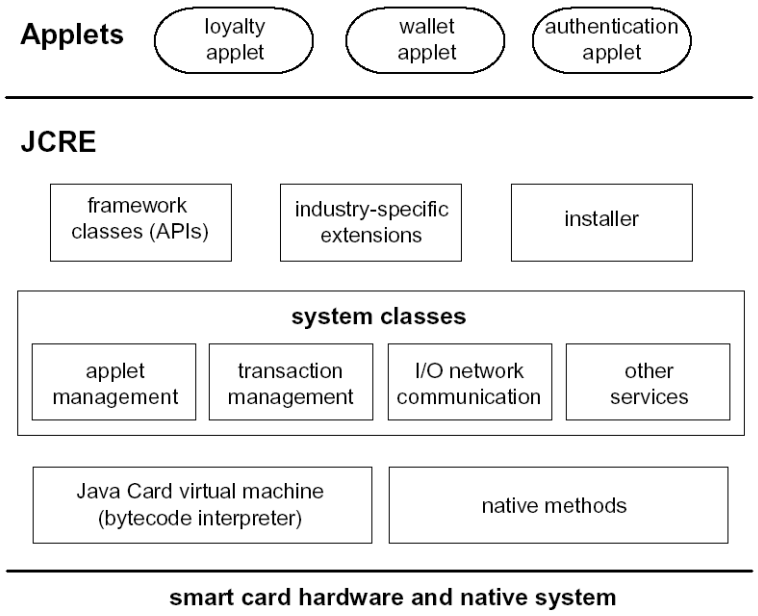

Figure 3.5: Elements of the JCRE [Che00]

The combination of the mentioned elements enable the JC to control resource management, I/O network communication, applet execution and the on-card and applet security.

Furthermore, the JCRE defines a clear separation between the SC system and the applets. Each applet is stored completely isolated in a special area protected by firewalls.

3.2.5 Application Programming Interface

The APIs for the JC platform include set of customized and adapted classes from the Java platform in order to provide Java language support and cryptographic services. Additionally special created classes for programming SC applications according to the ISO/IEC 7816 standard are included.

3.2.6 Security

Another JC design goal was to specifically enhance the security of a SC. Taking profit of the general security features in the Java platform the JC platform has integrated three special security enhancements, namely transaction atomicity, cryptographic classes, and the applet firewall [Mic03b].

Transaction atomicity solves the problem of an interrupted transaction and possible changes in the non-volatile memory. If a transaction is completed normally the memory will be updated, otherwise the card will perform no updating and returns to its prior state.

The idea of the firewall is to provide a separate private partitioning of the memory for each applet loaded onto the SC. This means that every applet is stored isolated from other applets on the card. Thus it is impossible for a malfunctioning applet to affect the card or other applets.

The cryptographic classes provide symmetric and asymmetric encryption and decryption algorithms, signature generation and verification, PIN management and several other features. The cryptography and security classes can be used to sign and authenticate CAP files and provide a secure installation mechanism.

Chapter 4

Design of the Java Specification Request 177

The JSR-177 is an optional package which adds security and trust services to the existing configurations and profiles. The SATSA defined in the JSR-177 provide security mechanisms to support applications to have secure communication such as accessing a network and mobile commerce.

SCs are primarily used to provide security for data and processing or other customer services. This specification defines an access model that enables applications on a J2ME enabled device to have a secure communication with the inserted SC. The idea is to provide a flexible mechanism that can be extended by service and equipment providers to define their secure operations.

4.1 Security and Trust Services

The definition for security and trust services can vary depending on the area to which they are referred to. In an information technology context, security and trust services are usually divided into five domains.

- Confidentiality
- Authentication
- Integrity
- Non-repudiation
- Reliability

Confidentiality has been defined by the ISO (International Standards Organization) as *"ensuring that information is accessible only to those authorized to have access"* and is one of the cornerstones of information security.

The process of ensuring that each of the two parties which are exchanging information (data) are able to prove their identity to the other party is called authentication.

Integrity takes care of checking that the data has not been altered since its origination.

The term non-repudiation means that it is impossible for a person or object who is responsible for a security related action to deny this action later on.

Reliability is the ability of an item to perform a required action under stated conditions.

To create a connection between the above described security domains and the real world the terms can be illustrated using as an example a mobile commerce transaction scenario.

4.1.1 Confidentiality

A practical application for mobile phones is mobile commerce. During a transaction only the two parties involved need to be aware of the private details of the transaction. At two points, the server and the mobile phone, the real content of the transaction can be examined.

Confidentiality primarily uses cryptography to satisfy the security requirements. The SE integrated in the phone is the SC and that is the place where the cryptographical operations take place. The plain text is encrypted in the SC and decrypted by the server on receipt to provide privacy.

4.1.2 Authentication

The identity problem is solved by authentication. In order to be able to figure out the identity of the two parties taking part in the transaction the method of trusted identification is used.

Authentication of the server guarantees the mobile phone that it is really communicating with the entity it wants to be connected with. Client authentication verifies that the requesting mobile phone is what is claims to be. In practice this authentication mechanism is implemented using digital certificates.

4.1.3 Integrity

The term which describes the mechanism that makes sure that the content of the transaction has not been changed is called integrity.

Integrity is guaranteed by analyzing the content of the transmission on reception. An algorithm determines if the data has been altered or not. Digital signatures are a common technique for testing the integrity of data.

4.1.4 Non-repudiation

Non-repudiation is the security domain which is responsible that a party cannot deny participating in a transaction.

The mechanism that usually provides the solution to this problem includes a combination of digital signatures and certificates.

4.1.5 Reliability

The security domain which gives information about the quality of the mobile commerce service is reliability. It indicates the functionality of the system.

4.2 The Goal of the JSR-177

The reason for creating the JSR-177 is to specify a collection of APIs. These APIs (SATSA) provide security and trust services by integrating a SE. A SE adds the following features to a J2ME device [Gro03].

- Secure storage to protect sensitive data, such as the user's private keys, public key (root) certificates, service credentials, personal information, etc.
- Secure execution, such as cryptographic operations to support payment protocols, data integrity and data confidentiality.
- Custom and enabling security features that J2ME applications can rely on to handle many value-added services, such as user identification and authentication, banking, payment, ticketing, loyalty applications, digital media play and so on.

A SE can be implemented in different ways but a very common implementation of a SE is a SC. Looking at the mobile communication area SCs are widely used. The SIM (discussed in Section 2.8.1) is the SE in GSM mobile phones

and the UICC (Universal Integrated Circuit Card) in UMTS (Universal Mobile Telephone System) mobile phones.

The specification contained in the JSR-177 is not limited to SCs as a SE. A SE can also be integrated in the J2ME device itself by using a suitable security framework.

Although the implementation of a SE is not restricted the JSR incloses special packages which are optimized for SCs used as a SE.

4.3 The Scope of the JSR-177

On account of the varying implementation of the SE is the design of the APIs included in the JSR-177 based on the following three criteria. This is an attempt to create a specification for each possible type of SE.

- J2ME devices need to fulfil certain requirements (processing power, memory size, ...).
- Popularity of the SE.
- APIs are flexible and extensible.

After the discussion of possible use cases and based on the three above mentioned criteria the JSR-177 defines APIs which add the following three capabilities to the J2ME platform:

- Communication APIs for SCs;
- Digital signature and user credentials management APIs;
- Cryptography APIs.

4.3.1 Communication APIs for SCs

As mentioned in Chapter 2 SCs provide a tamper-resistant programmable environment. SCs are popular because of the ability to integrate a wide range of security and trust services. An important benefit of the SC technology is the ability to upgrade these services or to install new improved applications.

SCs are able to communicate with a connected device via two different protocols, the APDU and the Java Card RMI protocol. Both access methods allow to use the security services supplied by the SC and are specified in the JSR-177.

4.3.2 Digital Signature and User Credential Management APIs

Via digital signature and user credential management services J2ME applications can generate digital signatures and manage user credentials (certificates, cryptographic keys, ...).

Digital signatures together with public certificates are used for the authentication and authorization process in public key cryptography.

The calculation of the cryptographic operations, like generating a digital signature, take places in the SE. Furthermore, the SE provides a secure storage for user credentials and cryptographic keys.

4.3.3 Cryptography APIs

Due to the requirement to use as little memory as possible on the J2ME device only a subset of the cryptography API defined in the J2SE is specified in the JSR-177. Only the basic cryptographic operations, like encryption, decryption, message digest, and digital signature verification are supported. These cryptographic operations enable J2ME applications to provide secure data communication, data protection, and content management.

The chosen cryptographic operations have been selected according to use cases and the subset of functions supported by already existing devices. Because of a necessary limitation of the cryptography API, the subset supports only a default Cryptographic Service Provider. A Cryptographic Service Provider is a package that provides a concrete implementation of one or more cryptographic services. Furthermore, the restricted functionality ensures that the anyway limited environment on the SC makes software development possible at all.

4.4 Architectural Overview

The JSR-177 defines APIs which are divided into four optional packages listed in Table 4.1 [Gro03].

The picture in Figure 4.1 illustrates the architecture of a J2ME enabled device and shows where the optional SATSA packages are positioned.

The SATSA optional packages are placed on top of the MIDP layer and define additional APIs available in the J2ME developing environment.

Additionally to the four optional packages, two further packages are mentioned in the JSR-177. The first package contains exception classes from `java.lang` which are required by the SATSA-APDU, SATSA-JCRMI, and the SATSA-

Package name	Brief description
SATSA-APDU	Defines an API to support communication with smart card applications using the APDU protocol.
SATSA-JCRMI	Defines a Java Card RMI client API that allows a J2ME application to invoke a method of a remote Java Card object.
SATSA-PKI	Defines an API to support application level digital signature and basic user credential management. To provide broader reuse, this API is independent of the type of SE that are utilized by a J2ME device.
SATSA-CRYPTO	Defines a subset of the J2SE cryptography API. It provides basic cryptographic operations to support signature verification, encryption, and decryption.

Table 4.1: JSR-177 optional packages

CRYPTO optional packages. The second package is the `javax.microedtion.io` which provides a generic connection framework interface.

Succeeding a description of the features of the optional packages included in the JSR-177 accroding to [Gro03] is given.

4.4.1 SATSA-APDU optional package

The functionality of the SATSA-APDU is determined in the `javax.micoroedition.adpu` package, which basically defines the APDU protocol handler for the communication with a SC according to the ISO/IEC 7816-4 standard.

If a J2ME application (MIDlet) wants to communicate with a SC application it needs to create an `APDUConnection`. The `APDUConnection` is responsible for exchanging APDUs (C-APDUs and R-APDUs) encoded in the ISO/IEC 7816-4 standard conforming format.

If the SC inserted in a J2ME enabled device supports multiple channels, then more than one `APDUConnection` can be created and several applications can be selected at the same time. During the creation of an `APDUConnection`, a logical channel is exclusively reserved for it. The API handles logical channel management by requesting a logical channel from the SC which allocates, if available, an unused logical channel (for more details see Section 2.6.4).

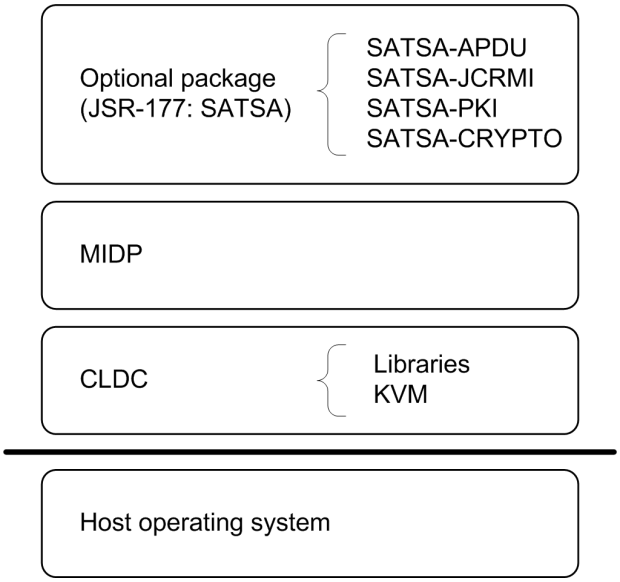

Figure 4.1: J2ME high level architecture

Otherwise no additional connection can be created and communication is only possible on channel 0 which is in GSM/UMTS networks usually reserved for the SIM application on the SC.

One exception for creating an `APDUConnection` on channel 0 exists. An `APDUConnection` can be created to communicate with the SAT/USAT (Universal SIM Application Toolkit) applications on channel 0 SIM/USIM (Universal Subscriber Identity Module). As a restriction the `APDUConnection` has limited capabilities when communicating with a SAT/USAT application.

During the communication with a SC the J2ME application is only allowed to send 'ENVELOPE' C-APDUs to trigger a SAT/USAT application on the SC. As mentioned in Section 2.8.3 the SAT allows the SC to be proactive, but proactive commands are not supported by the `APDUConnection` and the J2ME application needs to take care in order not to initiate a proactive session.

4.4.2 SATSA-JCRMI optional package

The SATSA-JCRMI optional package is a composite of a subset of the `java.rmi` package, three packages from JC, and the `javax.microedtion.jcrmi`. The `javax.microedtion.jcrmi` package provides classes and interfaces for the JC RMI connection.

Another way of communicating with an application on the SC is via the RMI protocol. For this purpose a `JavaCardRMIConnection` needs to be created which

initializes and initiates a JC RMI session with a JC applet.

If the SC supports multiple channels, the logical channel management handles the allocation of a logical channel by requesting it from the SC and a logical channel is reserved for each `JavaCardRMIConnection`. As in the case of an `APDUConnection`, several `JavaCardRMIConnections` can communicate simultaneously with JC applets through logical channels.

A `JavaCardRMIConnection` enables a J2ME application to access the initial remote reference. Via this reference the application can invoke methods of the remote object on the SC and obtain references to other remote objects.

4.4.3 SATSA-PKI optional package

The `javax.microedition.pki` and the `javax.microedition.securityservice` packages form the SATSA-PKI optional package. The first mentioned package provides classes for basic user certificate management. The second package supplies classes for the generation of application level digital signatures conform to the CMS (Cryptographic Message Syntax) format.

With the `UserCredentialManager` a J2ME application is able to add or remove a certificate or a certificate URI (Uniform Resource Identifier) to or from a certificate store. Furthermore, it enables the application to formulate a certificate enrollment request, which may be sent to a certificate registration authority.

For signing messages with a private key J2ME applications can use the `CMSMessageSignatureService` which provides a number of signature services for cryptographic messages. These cryptographic operations are frequently used to implement authorization, authentication, integrity, and non-repudiation. Depending on the security policy of the SE, the usage of such a key can make it necessary that the key is authorized (e.g. PIN entry required, no authorization required, ...).

4.4.4 SATSA-CRYPTO optional package

The last optional package of the JSR-177 contains four packages. The first package is the `java.security` package and it provides classes and interfaces for the security framework. The package supplying classes for cryptographic operations is the `javax.crypto` package. Further the `javax.security.spec` and the `javax.crypto.spec` packages provide classes and interfaces for key and algorithm parameter specifications.

The `KeyFactory` contained in this package supports the generation and storage of cryptographic private and public key pairs. Furthermore, it supplies a number of exportable cryptographic operations like a message digest (`MessageDigest`) and signature generation (`Signature`).

Encryption and decryption are done via the `Cipher` class which supports symmetric and asymmetric ciphers.

The key and algorithm parameter specifications define a transparent key representation of the key material that constitutes a key. The RSA and the DEA cryptographic algorithms are contained.

4.5 SATSA-APDU Communication Model

The software executed in the mobile phone can be separated into two parts, a Java specific and native software part. Figure 4.2 illustrates the SATSA-APDU communication model.

The software element where the communication is initiated is a J2ME application or short a MIDlet. The MIDlet creates an `APDUConnection` object which is the basic element responsible for APDU exchange between the MIDlet and the application executed in the SC.

The following element in the communication chain is the access control policy. The definition of which APIs are being allowed to be used by a certain MIDlet is made by the access control policy. This is part of the API security which is discussed in Section 4.6.

Placed directly after the access control policy block is the implementation of the KVM. The KVM is part of the runtime environment where the J2ME applications are executed.

The last element situated in the phone and being part of the native implementation is the APDU communication manager which is responsible for the communication process between the mobile phone and the SC. Furthermore, the APDU communication manager integrates a 'card reader' interface for the SC.

Once the command and the data have reached the APDU communication manager everything is put into APDUs. As mentioned in Section 2.6.3 two types, namely C-APDUs and R-APDUs exist. Depending on in which direction the APDU passes through a logical channel to the SC application, it is either a C-APDU, on the way from the APDU communication manager to the SC application, or a R-APDU in the opposite direction.

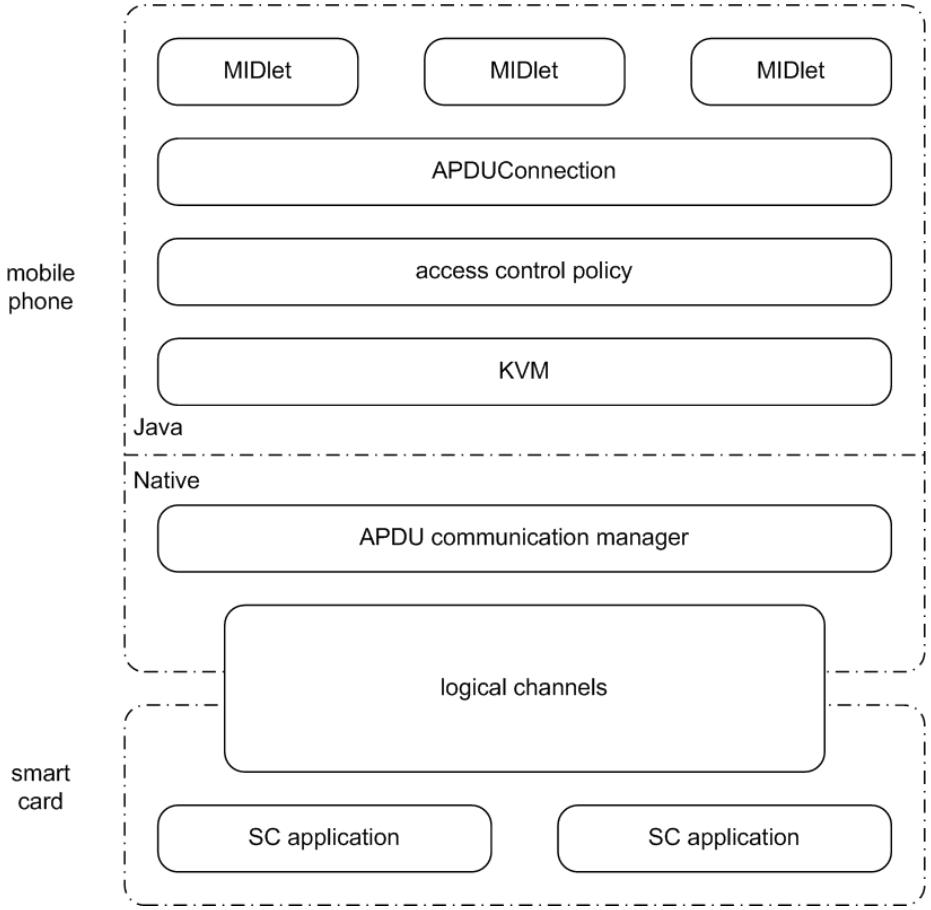

Figure 4.2: SATSA-APDU Communication Model

In case the SC does not support multiple channels, only one channel is available for communication between the phone and the SC.

4.6 SATSA Security

The four optional packages need different handling when it comes to securing the specified APIs.

The SATSA-CRYPTO optional package requires no special handling. All APIs defined in this package are generally available for use by a J2ME application. This is due to the character of this package. Only classes and interfaces for cryptographic operations are specified and there is no danger for a security threat.

A more careful treatment is needed for the APIs included in the SATSA-APDU, SATSA-JCRMI, and SATSA-PKI optional packages. If these APIs are available in a J2ME device supporting MIDP 1.0a, no usage control mechanism

exists. An application has unrestricted access to every API. The only security granted is the Java "sandbox" in which the application is executed.

Completely different is the situation if MIDP 2.0 is deployed in the J2ME device. MIDP 2.0 allows, as discussed in Section *Security*, to specify access permissions to privileged APIs in the SATSA-APDU, SATSA-JCRMI, and SATSA-PKI optional packages by using protection domains. This means that a J2ME application must have a permission to use a specific API indicated by the MIDP domain policy which can differ from device to device.

Another possibility to control access and protect private data stored on the SC and its resources, is to define a suitable access control policy in the SC. The two optional packages which are involved are the SATSA-APDU and SATSA-JCRMI optional packages which are responsible for the communication APIs. The SC can simply define in its access control policy if the `APDUConnection` or the `JavaCardRMIConnection` are allowed to access any function of the SC.

4.7 Use Cases

The APIs specified in the JSR-177 open the door to a wide variety of use cases. Due to the different possible applications it is again useful to separate the optional packages according to their main purpose.

The first group of applications is formed by the SATSA-APDU and SATSA-JCRMI optional package. A second group is the SATSA-PKI optional package and the last group of use cases is based on the functionality of the SATSA-CRYPTO optional package.

4.7.1 Use Cases for the SATSA-APDU and the SATSA-JCRMI Optional Packages

Mobile payment is an application that has already been tested and the companies Europay International, Mastercard International, and Visa International, short EMV[1], created a payment application that can be stored in a SE (e.g. a SC). A corresponding MIDlet needs to be downloaded to the MID which provides an online shopping service. After the user selected the desired products the MIDlet communicates with the EMV payment application on the card which initiates the transaction.

Operators are anxious to retain existing customers. In order to be able to

[1] `http://www.emvco.com/`

do that, operators provide customized services based on the subscriber's profile. The data necessary for creating personalized profiles is collected via an application which is running on the SC and stores the data on it. Access to the data for the customer relationship management is strictly controlled by the operator.

Another interesting use case is to measure the loyalty of customers. A simple loyalty application installed on a SC can produce relief. Every time a customer accesses a certain online service, downloads a J2ME application and so on, the loyalty application in the SE is triggered and increments the number of loyalty points. Once the loyalty point counter reaches a certain level the customer gets a reward.

A further and important use case can be discovered by looking at the functionality of the SAT and the USAT. So far the only way of triggering the SAT/USAT on the SC was to send a special type of SMS. With the implementation of the SATSA-APDU, a MIDlet is able to trigger the SAT/USAT. Triggering is not the only added functionality. Additionally the J2ME environment provides an enhanced GUI (Graphical User Interface) for the SAT/USAT.

4.7.2 Use Cases for the SATSA-PKI Optional Package

The SATSA-PKI provides powerful APIs which enable a MID to generate user certificates. The MID user can send an enrollment for a new certificate to the CA which returns the certificate to the requesting MID and stores it securely in the SE. The certificate contains a public and private key pair that can be used for authentication and authorization.

A possible application for user authentication is to control access to private data. Before allowing access to the data, the user is asked to generate a digital signature. The digital signature generated by the certificate stored in the SE is then verified to audit the users identity. By means of the verification result, access is granted or not.

A similar application is to use a digital signature for user authorization. A MID executes a banking application and downloads bank account details after a successful authentication. After having created payment instructions, the application sends the data including a digital signature to the bank server. If the bank server can verify the digital signature correctly the transactions are performed, otherwise they are ignored.

4.7.3 Use Cases for the SATSA-CRYPTO Optional Package

Data protection could be one suitable application. Nowadays MIDs are more and more used to store and reproduce pictures, music or more private data like password lists. There is a need for protecting this data and to prevent unauthorized access. This data can be encrypted with a private key. When needed an authorized user can decrypt the data with the suitable private key.

A further application could be another banking service that is executed as an J2ME application on a MID. The authentication process is handled in such a way that the user needs to enter a password to get access to the account. Therefore, the password needs to be securely transferred to the bank server by encrypting it with the bank's public key. The public key information is stored in the SE device and the SATSA-CRYPTO provides the cryptographic operations necessary for the encryption.

Chapter 5

Implementation

The goal of the practical part of the thesis work is to implement a subset of the JSR-177 in a Sony Ericsson mobile phone to create a working prototype for illustrating and testing the functionality of this newly created JSR.

The used phone does not support the MIDP 2.0 standard, it works with the previous MIDP 1.0a standard. But this limitation does not influence the functionality focused on in this work. It just implicates a loss of security. This means that the security mechanisms illustrated in Section *Security* and in Chapter 4 are not available and the J2ME applications are only secured by the Java "sandbox" architecture.

As mentioned above, the scope of the implementation is not the entire JSR-177, only the necessary classes for the functionality of the `APDUConnection` are implemented in the mobile phone prototype.

The essential classes for communication using the APDU protocol defined in the ISO/IEC 7816-4 standard are listed below.

- The `Connector` class
- The `APDUConnection` class

5.1 Connector

The `Connector` class is part of the `javax.microedition.io` package which is specified in the CLDC specification and it does not belong to the JSR-177 specification. But it is needed for the functionality of the packages defined by the JSR-177.

In order to conform to the requirements by the JSR-177 the `Connector` class specification supports the following two interfaces.

- `javax.microedition.apdu.APDUConnection`
- `javax.microedition.jcrmi.JavaCardRMIConnection`

But only the functionality of the `APDUConnection` is implemented in the mobile phone.

Furthermore, the SC communication does not support the same interaction model as other stream orientated communication methods. Therefore the following four stream orientated methods specified by the `Connector` class do not work with SCs.

- Method `openDataInputStream`
- Method `openDataOutputStream`
- Method `openInputStream`
- Method `openOutputStream`

The only important method for the SC communication via the APDU protocol is the `open` method. The number of parameters can vary from one to three. The first parameter specifies a URI string that addresses the application on the SC. A second parameter can be used to select an access mode (read, write and read/write). And, if used, a third parameter specifies a flag that tells if timeout exceptions are wanted or not.

The URI string needs to be conform to the BNF (Backus-Naur Form) syntax described in Table 5.1.

Symbol	Expression
<uri-string>	::= "apdu:"<target-address>
<target-adress>	::= [<slot>]";"<target>
<slot>	::= smart card slot number
<target>	::= "target="<AID>|"target=SAT"
<AID>	::= <5-16 bytes>

Table 5.1: URI string BNF syntax

The SC slot number is a hexadecimal number (1 byte) identifying the SC slot in which the SC is inserted. If it is left empty the default slot number is assumed. More details about the AID can be found in Section 2.4.2.

A detailed description of the `Connector` class can be found in [Gro03].

5.2 APDUConnection

Generally the package `javax.microedition.apdu` defines the protocol handler for SC communication according to the ISO/IEC 7816-4 standard using the T=0 protocol.

The `APDUConnection` itself is an interface for an APDU connection extending the `javax.microedition.io.Connection` interface.

How to create an `APDUConnection`, to communicate with an application selected on the SC and how to close the connection after finishing the communication process is illustrated in the following example.

```
...
APDUConnection myconnection = null;
try
{
  //creating an APDUConnection and selecting the
  //WIM application (AID=A0.00.00.00.63.50.4B.43.53.2D.31.35)
  String uri = "apdu:0;target=A0.00.00.00.63.50.4B.43.53.2D.31.35";
  myconnection = (APDUConnection)Connector.open(uri);
  //sending a command APDU to the application and receiving
  //the corresponding response APDU
  byte[] responseAPDU = myconnection.exchangeAPDU(commandAPDU);
  //further communication
  ...
}
catch (Exception e)
{
  ...
}
finally
{
  ...
  if (myconnection != null)
  {
     //close the APDUConnection
     myconnection.close();
  }
  ...
}
...
```

The `open` method from the `Connector` class is responsible for opening a logical

channel to the SC and for selecting a card application. This is done by sending a command APDU containing the 'MANAGE CHANNEL Open' (see Table 5.2) command to the SC which basically asks the SC if a logical channel is available for communication. Once the SC responded successfully, the `open` method selects the application it wants to communicate with via sending a command APDU containing the AID specified in the URI string and the 'SELECT Application' (see Table 5.3) command.

Command APDU	
Name	Value
CLA	$'00'_{16}$
INS	$'70'_{16}$
P1	$'00'_{16}$
P2	$'00'_{16}$
Lc	empty
Data	empty
Le	$'01'_{16}$

Response APDU	
Name	Value
Data	Assigned logical channel number
SW1	Status information
SW2	Status information

Table 5.2: 'MANAGE CHANNEL Open' command and response APDU

Command APDU	
Name	Value
CLA	$'0X'_{16}$
INS	$'A4'_{16}$
P1	$'04'_{16}$
P2	$'00'_{16}$
Lc	AID length
Data	AID
Le	empty

Response APDU	
Name	Value
Data	empty
SW1	Status information
SW2	Status information

Table 5.3: 'SELECT Application' command and response APDU

The CLA byte ($'0X'_{16}$) defines the logical channel for the communication in Bit 1 and Bit 2 of the lower nibble.

After communicating with the application using the `exchangeAPDU` method, the connection is being closed using the `close` method specified in the `Connection` interface. To close a logical channel a command APDU containing the 'MANAGE CHANNEL Close' (see Table 5.4) command is sent to the SC which closes

the channel and makes it available for further communication.

Command APDU	
Name	Value
CLA	$'00'_{16}$
INS	$'70'_{16}$
P1	$'80'_{16}$
P2	logical channel number
Lc	empty
Data	empty
Le	empty

Response APDU	
Name	Value
Data	empty
SW1	Status information
SW2	Status information

Table 5.4: 'MANAGE CHANNEL Close' command and response APDU

Theoretically opening, selecting and closing a logical channel could also be done by using the `exchangeAPDU` method. But due to a restriction of the JSR-177 specification 'MANAGE CHANNEL' and 'SELECT' commands are not allowed to be used in command APDUs sent with the `exchangeAPDU` method.

The `open` and the `close` method contained in these high level APIs do not return response bytes to the application. To tell the application that a request was not executed successfully a suitable exception is thrown. Moreover the application cannot figure out on which logical channel the communication takes place because the channel ID is only stored in the `APDUConnection` object.

If the SC is supporting multiple applications, several logical channels can be opened at the same time. As described in Section 2.6.4 the APDU protocol is synchronous and therefore is it not allowed that the communication is interrupted by another communication.

Aside from this, the SC can respond with status words that need to be taken care of in a special way. These status words are also known under the name procedure bytes and have the values $'61||XX'_{16}$ and $'6C||XX'_{16}$. If these bytes are returned as a response to a command APDU the application knows that command processing is not completed.

In the case of the $'61||XX'_{16}$ procedure bytes, the application knows that the SC has $'XX'_{16}$ more bytes which need to be picked up by the application. In order to do this the 'GET RESPONSE' command is sent to the SC application and the SC responds with suitable respond bytes. If the SC responds several

times with $'61||XX'_{16}$ procedure bytes the data needs to be accumulated before it is returned to the application sending the command APDU.

If the response contains the procedure bytes $'6C||XX'_{16}$, the receiving application needs to resend the command APDU after changing the value of the Le byte to the value of $'XX'_{16}$.

The `APDUConnection` interface defines a number of methods which can be arranged in the following categories.

- Exchange APDU
- PIN Management
- Answer to Reset

5.2.1 Exchange APDU

The `exchangeAPDU` method provides the basic communication interface with a SC. The main functionality of this high level API is to take a command APDU and forward it to the low level API responsible for the SC communication implemented in the MID and after processing the command in the SC respond to the J2ME application executed in the MID.

The command APDU is put together according to the rules specified by the ISO/IEC 7816-4 standard. Because the channel number associated with the communication is not visible to the outside, the `exchangeAPDU` method needs to put this number in the CLA byte as described in Section 2.6.4.

WIM APDU commands

The WIM, as described in Section 2.8.4, is a SC application which is available on almost every SIM. It has a multitude of operations and is therefore a proper application for testing.

'A0.00.00.00.63.50.4B.43.53.2D.31.35' is the standard AID value for the WIM application.

To illustrate the cryptographic operations supported by the WIM, some of them are briefly described. Before using these cryptographic operations a verification of the PIN needs to be performed.

'MANAGE SECURITY ENVIRONMENT' specifies commands that allow to select a security environment and to set attributes in a CRT (Control Reference

Template) like setting a specific public key before an encryption or selecting a certain private key before issuing a digital signature.

Several security environments can be store in a SC. Each security environment is labelled with a number. For activating a specific security environment, the 'MSE RESTORE' command containing the number of the wanted security environment needs to be executed. Via the 'MSE SET' command several parameters of the CRT can be set.

The actual cryptographic operations are implemented through the 'PERFORM SECURITY OPERATION' command. A number of operations is provided:

- 'PSO ENCIPHER'
- 'PSO DECIPHER'
- 'PSO COMPUTE DIGITAL SIGNATURE'
- 'PSO VERIFY DIGITAL SIGNATURE'
- 'PSO COMPUTE CRYPTOGRAPHIC CHECKSUM'

Chapter B shows a protocol of a communication with the WIM performing the computation of a digital signature and a proximate verification of it.

J2ME and Java Card application: MyWallet

The idea behind the `MyWallet` MIDlet is to have a wallet application on the SC which is communicating with the `MyWallet` MIDlet running in the mobile phone. The wallet application on the SC acts as a secure storage for the balance of the wallet.

The source code for the JC applet and a detailed description of the functionality is available at `http://java.sun.com/products/javacard/`. This code needs to be compiled and loaded onto the SC. During compilation the access code for the wallet application needs to be initialized and a suitable AID needs to be chosen. The wallet application specifies the commands listed in Table 5.5.

These four commands enable the `MyWallet` application to authenticate the user by verifying the access code. Furthermore, it allows to do credit and debit operations, and to check the balance of the wallet.

In case of an unsuccessful command execution the wallet applet responds with status words specifying the error reason. Successful execution is always answered with the status words $'90||00'_{16}$.

Functionality	INS code
Verify	20
Credit	30
Debit	40
Get balance	50

Table 5.5: Instruction codes (INS) for the JC applet `Wallet`

5.2.2 PIN Management

PIN management defines high level methods that enable J2ME applications to modify the PINs stored on a SC. The PINs are identified by a number, the PIN ID number. The following five possible PIN manipulations are provided by the PIN management:

- `enterPin` for verifying the PIN
- `changePin` for changing the PIN
- `disablePin` for disabling the PIN
- `enablePin` for enabling the PIN
- `unblockPin` for unblocking the PIN

Calling one of the above mentioned high level methods leads to a GUI dialog with the user of the mobile phone requesting to enter a PIN. It then asks for the current PIN, in the case of verifying, disabling and enabling the PIN. Changing the PIN results in requesting the current PIN and the value of the new PIN and unblocking the PIN results in asking for the PUK (Personal Unblock Key) and the value of the new PIN.

Once the data has been collected via the user interface it is put into a command APDU and sent to the SC which performs the selected command and responds with status words telling if the command was successfully executed or not.

5.2.3 Answer to Reset

The ATR is a data string with a maximum length of 33 bytes. Further details about the contents of this data string are described in Section 2.6.1.

The data string which contains the ATR information is not stored on the SC. There is no command to fetch the previous ATR. The ATR is saved in the mobile phone after each reset of the SC. All the method has to do is to check if a SC is inserted and to pick up the ATR data string.

5.2.4 Connection

The `Connection` class defined by the connection interface is the most general connection. The only method specified is the `close` method. As described earlier opening of a connection is done by using the `open` method of the `Connector` class.

5.3 APDUConnection Life Cycle

To sum up the functionality defined by the `APDUConnection` interface the state diagram in Figure 5.1 shows the possible states an `APDUConnection` can be in.

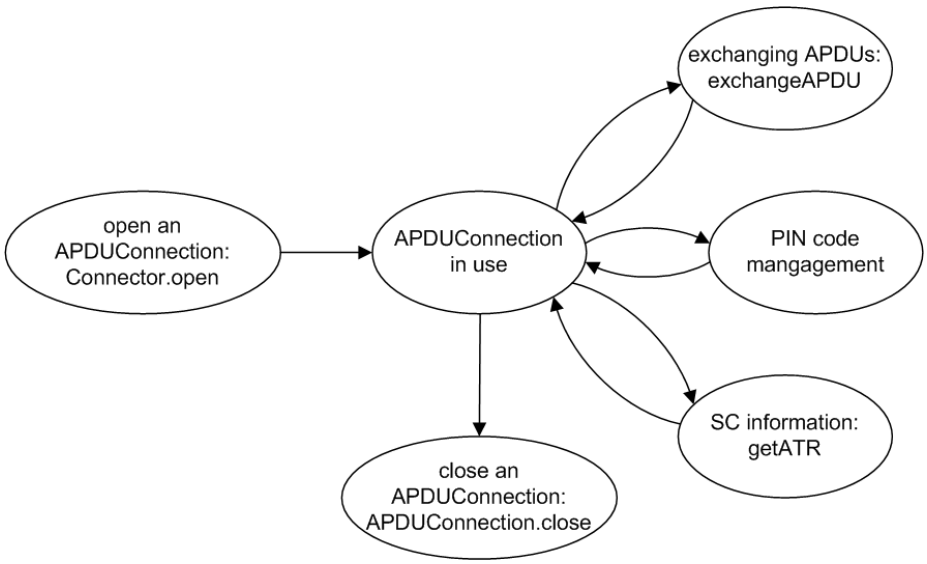

Figure 5.1: `APDUConnection` Life Cycle

At the beginning an `APDUConnection` is created with the `Connector.open` method that allocates a logical channel and selects the desired application on the SC. The `APDUConnection` is now ready for using the further defined methods.

The methods for PIN management, for exchanging APDUs and for getting the ATR information of the SC can be executed as often as needed before the `APDUConnection` is finally closed.

After executing `APDUConnection.close` the connection to the SC is terminated and the selected application as well as the allocated logical channel are free again.

Chapter 6

Conclusion and Outlook

Mobile devices are in general not a really new invention. However, the types of currently available mobile devices are rather powerful minicomputers with an incredible variety of features. These MIDs open a complete new perspective for mobile communication and mobile commerce.

Mobile phones belong to this group of MIDs that enhanced their usability and evolved from a simple phone to a highly complex multimedia device. Modern mobile phones integrate the functionality of a phone, a radio, a media player, a calendar, a WAP browser, a digital camera and a platform for video games.

This newly created functionality raises the amount of security required to guarantee a safe usage of the above mentioned applications. In order to keep step with these developments, mobile phone network operators, handset producers and Sun designed the JSR-177.

The goal of the JSR-177 is to define security and trust services for J2ME enabled devices. These services are a necessary step for a device to be able to open a secure communication. The implementation uses a so called SE for storing and protecting sensitive data, perform cryptographic operations, and create a secure execution environment to deploy custom security features. In case of mobile phones SCs are a common SE.

The discussed implementation concentrates on the communication APIs using the APDU protocol conforming to the ISO/IEC 7816 standard. These APIs enable the MID to communicate with the SE to exchange and manipulate data and to perform cryptographic operations.

Talking about security in the scope of information technology security and trust services are split into five elements. Authentication, authorization, integrity, non-repudiation, and reliability build these services. In practice they are imple-

mented by using digital certificates and cryptographic operations (encryption, decryption, digital signature, ...).

6.1 Thesis Achievements

The two main goals defined before the beginning of the master thesis, namely communication with the SC via the APDU protocol and PIN management, have been reached.

The first and most important element of the implementation is the APDU communication. The APDU communication provides the high level APIs to create and manage certificates and the low level APIs to perform cryptographic operations.

PIN management enables the J2ME environment to verify, change, enable, disable, and unblock the PIN. An application communicates with the user via a GUI which collects the data and encodes it into a valid APDU command to send it to the SC.

To make sure that the created prototype fulfils the requirements and functionality it has been tested with multi application SCs containing several applications. The WIM application described in Section *WIM APDU commands* and the wallet application described in Section *J2ME and Java Card application: MyWallet* have been of special interest.

Furthermore, the working prototype has been presented at SIM 2004 in Amsterdam (March 2004) and several prototypes are being tested by operators, SC producers and Sun.

6.2 Evaluation

Due to the fact that handset producers and mobile network operators are members of the expert group responsible for the design of the JSR-177 the specification is a compromise concerning its functionality.

Both parties are interested in gaining more access to the other parties' devices. The handset producers want to use more of the functionality and security features of the SIM that is inserted into the phone. On the other hand the operators want more freedom when it comes to install their applications on the phone because so far the only way of manipulating data on the SIM is via the SAT.

One consequence of this conflict of interests can be found in the implementation of the APDU communication, more precisely the method `exchangeAPDU`.

The communication is based on the APDU protocol which is a powerful way to communicate. But these high level APIs need to get the commands encoded in bytes and there are no other interfaces available which add another level of abstraction.

Aside from this political design decisions the integrated security model reaches the aimed goal with one limitation. MIDP 1.0a does allow access to all the APIs specify by the JSR-177 and there is no mechanism to limit access to certain applications. The only protection available is the Java "sandbox".

The design of MIDP 2.0 on the other hand provides a powerful security model that allows to define security domains for trusted and untrusted applications. Furthermore, it supplies a model to categorize the MIDlets within the trusted domain via protection domains which allow to define a fine-grained structure of permissions to specify which applications are allowed to use which APIs. MIDlets can be signed and by verifying the digital signature the MID can find out to which root certificate and to which domain the MIDlet belongs.

The number of use cases is high and ranges from simple wallet and loyalty applications to banking applications as well as to complex online shopping applications.

6.3 Future Work

The design of the JSR-177 is still in progress and the proposed final draft is available since the end of April 2004. Maybe minor modifications will be added but the general design of the JSR-177 is completed.

The JSR-177 implementation of the APDU communication in a Sony Ericsson mobile phone prototype is still in its test phase and it depends on the satisfaction and the acceptance of the network operators to make further implementation decisions.

A possible next step is the implementation of the functionality of the SATSA-JCRMI optional package. To sum up the JSR functionality a last step is the implementation of the two missing optional packages SATSA-PKI and SATSA-CRYPTO.

The JSR-177 is a first try to fulfil the security requirements needed by modern mobile phones and their applications. Acceptance by the operators and application vendors will show whether if the design reaches all needs or if additional changes are necessary and a further version of the JSR-177 needs to be created.

Appendix A

JSR-177 Package Summary

A.1 Description

The Security and Trust Services API for J2ME defines four optional packages to support smart card communication, generation of digital signatures, and low-level cryptography operations [Gro03].

A.2 Generic Connection Framework

The smart card communication API is based on the Generic Connection Framework in the `javax.microedition.io` package. Two smart card connection interfaces are defined for establishing a connection with a smart card. The `APDUConnection` interface is used to communicate with ISO7816-4 compliant smart cards. The `JavaCardRMIConnection` interface is used to initiate a JCRMI session. These connections are defined in the `javax.microedition.apdu` and `javax.microedition.jcrmi` packages respectively.

A.3 SATSA-APDU Optional Package

The optional package SATSA-APDU includes two components to support communication with ISO7816-4 compliant smart cards using the APDU protocol.

- A subset of the `java.lang` package. It supports the exception class `UnsupportedOperationException`, which is not included in the CLDC API or the MIDP API.
- The `javax.microedition.apdu` package. It includes the interface `APDUConnection` to support APDU exchanges.

A.4 SATSA-JCRMI Optional Package

The optional package SASTA-JCRMI includes four components to provide a Java Card RMI client API:

- A subset of the `java.lang` package. It supports the exception class `UnsupportedOperationException`, which is not included in the CLDC API or the MIDP API.
- A subset of the `java.rmi` package in the Java 2 Platform, Standard Edition to provide the basic RMI client interfaces. The subset includes the Remote interface and the `RemoteException` class.
- The `javax.microedition.jcrmi` package. It includes the `JavaCardRMIConnection` interface used to initiate a JCRMI session and interfaces used by the stubs generated by the JCRMI stub compiler.
- A subset of the Java Card API. The subset includes the exception classes defined in the packages `javacard.framework`, `javacard.framework.service`, and `javacard.security` . These exceptions may be thrown on method invocation of a Java Card object because of cryptographic errors (the `javacard.security` package), card framework access errors (the `javacard.framework` package) or errors detected in accessing the card services (the `javacard.framework.service` package). Therefore, the Java Card exception classes are included to provide the same behavior when the method is invoked remotely.

A.5 SATSA-PKI Optional Package

The optional package SATSA-PKI includes two components to provide generation of digital signatures and basic user credential management (for example, an X.509 certificate is a user credential that includes a public key).

- The `javax.microedition.pki` package. It supports generation of certificate requests and local registration of the user credentials. The user credentials are used in conjunction with other parameters to compute formatted digital signatures. When deployed on the MIDP 2.0 platform, this package also includes the classes Certificate and `CertificateException`, which are defined in the MIDP 2.0 API.

- The `javax.microedition.securityservice` package. It supports generation of application-level digital signatures that conform to CMS format.

A.6 SATSA-CRYPTO Optional Package

The optional package SATSA-CRYPTO is a subset of the Java 2 platform, Standard Edition (J2SE) Cryptography API.

- The `java.security` and `java.security.spec` packages. They provide basic support for accessing public keys, computing digests, and verifying digital signatures.
- The `javax.crypto` and `javax.crypto.spec` packages. They provide basic support for encryption and decryption of data.
- A subset of the `java.lang` package. It includes the exception class `IllegalStateException`, which is not included in the CLDC API or the MIDP API.

Appendix B

WIM Communication Protocol

The printout is the protocol of a communication between a mobile phone and an inserted SC. The mobile phone establishes a connection with the WIM on the SC and performs computation of a digital signature and a proximate verification of it.

Each command of the protocol is shown separately and consists of three lines. The first line gives a brief description of the command. The following line specified by 'DATA IN' encodes the command to a valid APDU command and the last line named 'DATA OUT' shows the response of the SC. A further description of the commands printed in the protocol can be found in [For01].

B.1 Printout

```
'MANAGE CHANNEL Open'
DATA IN  (00 70 0000 01)
DATA OUT (01 9000)

'SELECT APPLICATION' -> WIM
DATA IN  (01 A4 0400 0C A000000063504B43532D3135)
DATA OUT (9000)

'MSE RESTORE' -> GENERIC RSA SECURITY ENVIRONMENT
DATA IN  (81 22 F302 00)
DATA OUT (9000)

'VERIFY' -> PIN-01
DATA IN  (81 20 0001 08 31313131FFFFFFFF )
DATA OUT (9000)
```

```
'VERIFY' -> PIN-02
DATA IN  (81 20 0002 08 32323232FFFFFFFF)
DATA OUT (9000)

'MSE SET' -> DECIPHERING, SIGNING AND VERIFYING
DATA IN  (81 22 41B6 07 8102FF07840105)
DATA OUT (9000)

'PSO COMPUTE DIGITAL SIGNATURE'
DATA IN  (81 2A 9E9A 14 7c222fb2927d828af22f592134e8932480637c0d)
DATA OUT (6180)

'GET RESPONSE'
DATA IN  (01 C0 0000 80)
DATA OUT (6C155D08048BEA641932118A708F9C0B6359DBE8DAFD80BB4946BF0FFAC0B
717160B12D669B034C708A2C2BBD0FEC318C4B2BAE6AC4AEACDD3860A84D8B965C39643
3476413B3C056C0E5A493B549B6144EC284E00CC80EAB1003669B2B9A36FCC49666B9B2
0D009FBCF8E2EEA9AEDE8EAEF2B11BDE7C0966CA6C42718C00F44 9000)

'MSE SET' -> VERIFYING AND ENCIPHERING(81), SIGNING AND VERIFYING(B6) -
PUBLIC KEY
DATA IN  (81 22 81B6 8A 8381 87 0003 010001 0080 ADB432305090A4A80E6FED
BD1639C5C096281503FD4F72AE3649AF8E370DFF00D81F05B3DA80F6E15F332CECC7A36
4B32663F88A3FD86F3FBF58662137A973440941B4DDF078D1866F7D5C784E525604900B
A78E84F21AEE94D48F390ACEC5613D738CBE0074BE3C393AB9FB4647B946C760B7413D0
92EB29AD694904FF6E8B5)
DATA OUT (9000)

'MSE SET' -> VERIFYING AND ENCIPHERING(81), SIGNING AND VERIFYING(B6) -
DATA TO BE VERIFIED
DATA IN  (81 22 81B6 16 90147C222FB2927D828AF22F592134E8932480637C0D)
DATA OUT (9000)

'PSO VERIFY DIGITAL SIGNATURE' -> DECIPHER WITH GIVEN PUBLIC KEY
DATA IN  (81 2A 00A8 83 9E8180 6C155D08048BEA641932118A708F9C0B6359DBE8
DAFD80BB4946BF0FFAC0B717160B12D669B034C708A2C2BBD0FEC318C4B2BAE6AC4AEAC
DD3860A84D8B965C396433476413B3C056C0E5A493B549B6144EC284E00CC80EAB10036
69B2B9A36FCC49666B9B20D009FBCF8E2EEA9AEDE8EAEF2B11BDE7C0966CA6C42718C00
F44)
DATA OUT (9000)
```

Appendix C

List of Abbreviations

3G ..... Third Generation

AAM ..... Application Abstract Machine

AES ..... Advanced Encryption Standard

AID ..... Application Identifier

APDU ..... Application Protocol Data Unit see also the glossary entry for ☞APDU

API ..... Application Programming Interface see also the glossary entry for ☞API

ATR ..... Answer to Reset

BNF ..... Backus-Naur Form

CA ..... Certification Authority

CAP ..... Converted Applet

CDC ..... Connected Device Configuration see also the glossary entry for ☞CDC

CHV ..... Card Holder Verification

CLDC ..... Connected Limited Device Configuration see also the glossary entry for ☞CLDC

CMS ..... Cryptographic Message Syntax

CRT ..... Control Reference Template

DEA ..... Data Encryption Algorithm

DES ..... Data Encryption Standard

DF ..... Dedicated File

EF ..... Elementary File

EMS ..... Enhanced Message Service

GSM ..... Global System for Mobile Communications see also the glossary entry for ☞GSM

GUI ..... Graphical User Interface see also the glossary entry for ☞GUI

ICC ..... Integrated Circuit Card

IDEA ..... International Data Encryption Algorithm

ISO ..... International Standards Organization

J2ME ..... Java 2 Micro Edition

J2SE ..... Java 2 Standard Edition

JAR ..... Java Archive

JC ..... Java Card

JCP ..... Java Community Process

JCRE ..... Java Card Runtime Environment

JCVM ..... Java Card Virtual Machine

JSR-177 ..... Java Specification Request 177 see also the glossary entry for ☞JSR-177

JVM ..... Java Virtual Machine

KVM ..... K Virtual Machine

ME ..... Mobile Equipment

MEL ..... Multos Executable Language

MF ..... Master File

MID ..... Mobile Information Device

MIDlet ..... Mobile Information Device (app)let see also the glossary entry for ☞MIDlet

MIDlet suite Mobile Information Device (app)let suite see also the glossary entry for ☞MIDlet suite

MIDP ..... Mobile Information Device Profile

MIDPEG ..... Mobile Information Device Profile Expert Group

MMS ..... Multimedia Messaging System

MS ..... Mobile Station

OEM ..... Original Equipment Manufacturer

PCS ..... Personal Communications Service see also the glossary entry for ☞PCS

PIN ..... Personal Identificaion Number

PIX ..... Proprietary Application Identifier Extension

PUK ..... Personal Unblock Key

RID ..... Registered Application Provider Identifier

RMI ..... Remote Method Invocation see also the glossary entry for ☞RMI

RSA ..... Rivest Shamir Adelman

SAT ..... SIM Application Toolkit see also the glossary entry for ☞SAT

SATSA ..... Security and Trust Services API

SC ..... Smart Card

SE ..... Security Element

SIM ..... Subscriber Identity Module

SMS ..... Short Message Service

TPDU ..... Transmission Protocol Data Unit

UICC ..... Universal Integrated Circuit Card

UMTS ..... Universal Mobile Telephone System see also the glossary entry for ☞UMTS

URI ..... Uniform Resource Identifier see also the glossary entry for ☞URI

USAT ..... Universal SIM Application Toolkit

USIM ..... Universal Subscriber Identity Module

VM ..... Virtual Machine see also the glossary entry for ☞VM

WAP ..... Wireless Application Protocol

WIM ..... Wireless Identity Module or WAP Identity Module

WML ..... Wireless Markup Language

WSC ..... Windows for Smart Cards

Appendix D

Glossary

APDU: *Application Protocol Data Unit* The APDU is the communication unit between a reader and a card. The structure of an APDU is defined by the ISO 7816 standards. There are two categories of APDUs: command APDUs and response APDUs. As the name implies, the former is sent by the reader to the card: it contains a mandatory 5-byte header and from 0 to up to 255 bytes of data. The latter is sent by the card to the reader: it contains a mandatory 2-byte status word and from 0 to up to 256 bytes of data.

API: *Application Programming Interface* A set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks. A programmer puts the blocks together.

CDC: *Connected Device Configuration* A J2ME configuration aimed at, for example, PDAs.

CLDC: *Connected Limited Device Configuration* A J2ME configuration aimed at, for example, mobile phones.

GUI: *Graphical User Interface* A program interface that takes advantage of the computer's graphics capabilities to make the program easier to use. Well-designed graphical user interfaces can free the user from learning complex command languages. On the other hand, many users find that they work more effectively with a command-driven interface, especially if they already know the command language.

GSM: *Global System for Mobile Communications* GSM is an open, non-

proprietary system that is constantly evolving. One of its great strengths is the international roaming capability. This gives consumers seamless and same standardised same number contactability in more than 170 countries. GSM satellite roaming has extended service access to areas where terrestrial coverage is not available.

Java: A high-level programming language developed by Sun Microsystems. It is an object-oriented language similar to C++, but simplified to eliminate language features that cause common programming errors. Java source code files are compiled into a format called bytecode, which can then be executed by a Java interpreter. Compiled Java code can run on most computers because Java interpreters and runtime environments, known as Java Virtual Machines (VMs), exist for most operating systems, including UNIX, the Macintosh OS, and Windows.

JSR-177: *Java Specification Request* The purpose of the JSR-177 is to specify a collection of API that provides security and trust services by integrating a Security Element (SE).

MIDlet: *Mobile Information Device (app)let* A MIDlet is an application written for MIDP. MIDlet applications are subclasses of the javax.microedition.midlet.MIDlet class that is defined by MIDP.

MIDlet suite: *Mobile Information Device (app)let suite* A MIDP is packaged and distributed as MIDlet suite. A MIDlet suite can contain one or more MIDlets. The MIDlet suite consists of two files, an application descriptor file with a .jad extension and an archive file with a .jar file. The descriptor lists the archive file name, the names and class names for each MIDlet in the suite, and other information. The archive file contains the MIDlet classes and resource files.

PCS: *Personal Communications Service* The PCS is the 1900 MHz radio band used for digital mobile phone services in the United States and Japan.

RMI: *Remote Method Invocation* A set of protocols being developed by Sun's JavaSoft division that enables Java objects to communicate remotely with other Java objects.

SAT: *SIM Application Toolkit* The SIM Application Toolkit is a set of commands which defines how the card should interact with the outside world

and extends the communication protocol between the card and the handset. With the SIM Application Toolkit, the card has a proactive role in the handset (this means that the SIM initiates commands independently of the handset and the network).

UMTS: *Universal Mobile Telephone System* UMTS is one of the 3G (Third Generation) mobile systems being developed within the ITU's IMT-2000 framework. It is a realisation of a new generation of broadband multimedia mobile telecommunications technology. The coverage area of service provision is to be world wide in the form of FLMTS (Future Land Mobile Telecommunications Services and now called IMT2000).

URI: *Uniform Resource Identifier* A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource. It is specified in RFC2396.

VM: *Virtual Machine* A self-contained operating environment that behaves as if it is a separate computer. For example, Java applets run in a Java Virtual Machine.

Bibliography

[Che00] Zhiqun Chen. *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*. Addison-Wesley, first edition edition, June 2000. Information available online `http://java.sun.com/developer/Books/consumerproducts/javacard/`.

[For01] Wireless Application Protocol Forum. *Wireless Identity Module*, July 2001. available online `http://www.wapforum.org/`.

[GJ02] Scott B. Guthery and Timothy M. Jurgensen. *Smart Cards: The Developer's Toolkit*. Prentice Hall PTR, first edition edition, July 2002. Information online `http://unix.be.eu.org/docs/smart-card-developer-kit/ewtoc.html`.

[Gro03] JSR 177 Expert Group. *Security and Trust Services API (SATSA) for J2ME*, October 2003. available online `http://jcp.org/en/jsr/detail?id=177`.

[Gui03] Tony Guilfoyle. *BasicCard*, April 2003. available online `http://www.basiccard.com/`.

[Hen01] Mike Hendry. *Smart Card Security and Applications*. Artech House, second edition edition, April 2001.

[Ins99] European Telecommunications Standards Institute. *GSM 02.17 V8.0.0*, September 1999. available online `http://www.3gpp.org/`.

[Ins03] European Telecommunications Standards Institute. *ETSI TS 102 221 V6.2.0*, September 2003. available online `http://www.etsi.org/`.

[Mic00] Sun Microsystems. *J2ME Building Blocks for Mobile Devices*, May 2000. available online `http://java.sun.com/products/kvm/wp/KVMwp.pdf`.

[Mic02a] Sun Microsystems. *Java 2 Platform, Micro Edition*, November 2002. available online `http://java.sun.com/j2me/`.

[Mic02b] Sun Microsystems. *Mobile Information Device Profile, Version 2.0*, November 2002. available online `http://java.sun.com/j2me/`.

[Mic03a] Sun Microsystems. *Connected Limited Device Configuration, Version 1.1*, March 2003. available online `http://java.sun.com/j2me/`.

[Mic03b] Sun Microsystems. *Java Card Platform Security*, November 2003. available online `http://java.sun.com/products/javacard/JavaCardSecurityWhitePaper.pdf`.

[MUL03] MULTOS. *Welcome to MULTOS*, March 2003. available online `http://www.multos.com/`.

[OG01] C. Enrique Ortiz and Eric Gigure. *Mobile Information Device Profile for Java 2 Micro Edition*. John Wiley and Sons, first edition edition, Januray 2001. Information available online `http://java.sun.com/products/midp/`.

[RE04] Wolfgang Rankl and Wolfgang Effing. *Smart Card Handbook*. John Wiley and Sons, third edition edition, February 2004. Information available online `http://www.wrankl.de/SCH/SCH.html`.

[rGPP04] 3rd Generation Partnership Project. *3GPP TS 11.14 V8.16.0*, March 2004. available online `http://www.3gpp.org/`.

[Tec04] CardWerk Technologies. Multos technology and architecture. Technical report, CardWerk Technologies, April 2004. available online `http://www.cardwerk.com/smartcards/MULTOS/`.

Index

K

L

M

O

P

R

S

T

W