ROYAL INSTITUTE OF TECHNOLOGY

KTH

Master Thesis
Report
2002-11-22

SMARTTRUST

Evaluation of architectures for the development of secure mobile applications

By

Paul Velenik d98-pve@d.kth.se

Department of Microelectronics and Information Technology

Examiner & advisor: Associate Professor Vladimir Vlassov
Department of Microelectronics and Information Technology
Royal Institute of Technology.

Industrial advisor: Jesper Söderlund, Smarttrust

Abstract

This is one of two theses that were made in close collaboration at Smarttrust, which objective were to evaluate different architectures for the development of secure mobile applications. SmartTrust wants to see how their WIB is compared to other techniques such as Java in cellular phones and Java on the SIM card with a focus on their business area in secure mobile applications.

Smarttrust products are focused on SIM management and especially to manage SIM cards that have a WIB. This thesis is focusing on an evaluation of the WIB compared to a Java enabled SIM card even though also WIB with Java plug-in is looked upon. The main objective is to compare their difference regarding performance, development environments and some management aspects like learning threshold.

The conclusion when it comes to choosing strategy is that there are a lot of issues to consider, if ease of development is the most important aspect then the WIB or Java on the phone is ideal. One benefit from using the Java Card compared to the WIB is that much more can be done locally and thus decrease the communication needed. Looking at security all SIM cards evaluated have inherent support to offer a totally secure end-to-end communication, which isn't the case for J2ME applets.

Development environments available for the different technologies vary a lot. The tools available for developing WIB services and J2ME midlets are the best as it comes to documentation and stability while the tools available for Java Card and WIB with Java plug-ins lack a lot in those areas.

The most interesting parts to look further into in the future are the threats from WAP2.0, the convergence between J2ME in the phone and a Java enabled SIM Card and the fact that both mobile operators and phone manufacturers are moving the focus more and more from the SIM to the phone, e.g. phone numbers are stored in the phone instead of on the SIM.

Table of contents

List of Figures

List of Tables

Acknowledgements

First and for most I would like to thank Smarttrust for giving me the opportunity to do the thesis work there, they have also been very supportive at times when problems have occurred. Special thanks also goes out to the supervisor Vladimir Vlassov who has come with very good and interesting opinions regarding the work that has been conducted. Finally a special thanks goes to my family and friends (also known as those "Då äventyren aldrig var längre bort än 17-berget") who have been very supportive throughout my years at KTH.

1 Introduction

This is one of two theses that have been done at Sonera Smarttrust during the year of 2002 the other related thesis was conducted by Peter Lindström. Both these theses were done in tight collaboration and supervised by Vladimir Vlassov at the Department of Microelectronics and Information Technology at the Royal Institute of Technology, Stockholm, Sweden.

SmartTrust is a wholly owned subsidiary of the Finnish mobile operator Sonera. During the year of 2000 SmartTrust acquired both Acrosswireless and ID2, which had spawned from the consulting company AU Systems. SmartTrust has kept the software development in Liljeholmen where ID2 and Acrosswireless previously resided. The number of employees today is around 300.

SmartTrust is today a leading provider of infrastructure software, enabling secure e-services to wireless and Internet users and more than 80 operators are using their Delivery Platform to launch enhanced SMS services and to manage mobile end-user applications.

To be able to give as many cell phone users as possible the ability to use mobile e-services SmartTrust developed the WIB (Wireless Internet Browser), which specification is public so that the SIM card manufacturers can incorporate the WIB into their products. The WIB was launched in 1999 and has now been integrated into more than 25 million SIM cards. In that time it has become an essential tool for operators looking to broaden their mobile e-service capabilities.

For an operator to be able to supply their customers with a WIB the only requirement is that the cell phone is able to handle SIM toolkit, which almost all phones do since the past 2 and half year. This means that the operator can reach the majority of his subscribers and not just the percentage of those with WAP-enabled handsets

But today with GPRS and 3G around the corner one question that SmartTrust has is how secure mobile transactions will be developed in the future. Even though 25 million SIM cards are enabled with the WIB it is far from a majority of the SIM cards out there. The question is whether to keep focus on the WIB/WIG solution in the future or start moving to some other platform like Java enabled phones. If a change of platform were to be done then the question is what benefits/drawbacks that can come of moving to the platform.

The first intention was that one thesis report was to be written by Peter Lindström and Paul Velenik, but their work came out of phase with each other due to Smarttrust having problems on delivering software needed. A decision was made to split the thesis into two separate reports. Where Peter Lindström focused on WIB with Java Plug-ins and Java enabled cell phones while Paul Velenik report focused on the WIB and Java enabled SIM cards.

Smarttrust products are focused on SIM management and especially to manage SIM cards that have a WIB, this thesis is focusing on an evaluation of the WIB compared to a Java enabled SIM card. This is especially interesting for Smarttrust since there are many Java Cards coming into market today and how it might change the place that the WIB holds today. The main objective is to compare their differences regarding performance, development environments and some management aspects such as learning threshold.

Peter Lindström thesis report [PEL02] has compared WIB with Java plug-ins and the use of Java on the cell phone. The motivation to include Java in the thesis project and not WAP is that Java on the cell phone resembles applications stored on the SIM i.e. applications are stored on the phone and not as in WAP dependent on a content provider. Java on the phone also lies closer to SmartTrust business areas, which concern SIM management and soon also device management.

1.1 Structure of the report

A part of this report were written by Peter Lindström and incorporated to give the reader a better background of the different technologies evaluated on the behalf of Smarttrust. Parts written by Peter Lindström are sections *2.1 Java 2 Platform, Micro Edition (J2ME), 2.2.4 SIM WIB with Java plug-in, 3.4 Java in the phone evaluation, 3.5 WIB with Java plug-in evaluation and 4.1.2 Java Phone*. Section *3.1 Application* was written in collaboration by both Lindström and Velenik. The rest of the thesis have been written by Paul Velenik..

This report is structured as follows. First there is a background part essential for those who fully want to be able to understand the report. Then come the evaluation itself comparison of the different technologies and finally a part with discussion, analysis and conclusions.

Chapter *2 Background* contains primarily two sections one about Java and especially J2ME since that is what is used in cell phones thereafter comes a section about SIM cards. The section *2.2 SIM cards* is devoted to Java enabled smart cards, WIB (Wireless Internet Browser) enabled cards as well as WIB enabled cards that can have Java plug-ins. Finally this chapter also contains a short introduction to the security features available on the GSM layer.

Chapter *3 Development and evaluation* contains the results from the evaluation of the architectures. Section *3.1 Application* describes the example application that was developed to use as a framework for the comparison of the different architectures. The comparison looked into areas such as design changes that had to be made due to the architecture used and development environments available. Sections *3.2 WIB evaluation* and *3.3 Java Card evaluation* contains the actual results of the evaluation and in *3.4 Java in the phone evaluation* and *3.5 WIB with Java plug-in evaluation* there is a short introduction into the evaluation performed by [PEL02]. It's not complete since [PEL02] is work in progress..

Chapter *4.2 Development environment/Learning threshold* compares in detail the performance and the development environments that are available for the WIB and Java Card. WIB with Java plug-in and Java in the phone are also analysed briefly that work has been conducted by Peter Lindström and are available in detail in his report [PEL].

Chapter *5 Future Work* introduce areas in which more work should be done both directly related to this thesis as well in areas closely related. E.g. it's suggested that more work should be done on the server side of the application evaluated and also to look into new areas such as "smart phones".

Chapter *6 Conclusions* presents the conclusions that can be drawn from the thesis work conducted such as when and why to choose a specific architecture

1.2 Conventions used

When something is referred to as SMs in the thesis it's meant to be SIM Toolkit messages packed in Short Messages.

There are a lot of different GSM standards e.g. GSM 11.11, which specifies the SIM card and GSM 11.14 that specifies the SIM Toolkit commands. These and other GSM standards like GSM 03.48, which specifies the Over The Air interface may sometimes just be referred to as GSM. This also applies to GPRS which is a packet service used in GSM networks and is just referred to as GPRS while an "ordinary" dialled up connection is referred to as GSM in the text.

2 Background

2.1 Java 2 Platform, Micro Edition (J2ME)

2.1.1 Overview of Java

Java is an object-oriented programming language. An object is a software bundle of related variables and methods. Software objects are often used to model real-world objects you find in everyday life [CON02]. Java is syntactically similar to C++, but differs in some fundamental ways. One of the biggest differences lies in their management of objects and object references. The Java language allocates and de-allocates memory automatically as the program creates and destroys objects, without Java programmers having to think about it. In opposite, C++ programmers must allocate and free memory on their own in the source code, which can lead to memory leaks and unpredictable behavior as programs attempt to use objects that no longer exist. In the area object references there is also an advantage for Java programmers since they use well-behaved, type-safe direct references to objects. C++ programmers use instead pointers, which can be modified directly, making them point to memory the application does not own. Because Java handles the memory allocation Java becomes slower in execution than C++, but is easier to use, since Java does some of your job.

The Java source code is run on a virtual machine (VM). This means that instead of being run on top of the operating system (like an .exe file), the code is being executed in a program, the virtual machine, that is run on the operating system. One advantage with this architecture is that you don't have to adapt the code to the different operating systems, you let the virtual machine adapt instead, so that the compiled code can execute independent of the operating system. Another advantage is the tight control of the executed binaries, which protects you from untrusted code.

The big strength in the Java platform is that it includes an extensive set of standard class libraries called application programming interfaces or APIs. These have a wide user area, from graphical user interface (GUI) through cryptography and CORBA.

The Java platform, consisting of the Java language, Java Virtual Machine (JVM) and Java APIs, is used over a wide range of computer hardware, everything from smartcards through enterprise servers. Since the range is so wide Java comes in three editions, see Figure 2-1. Java 2 Platform[JAV02]:

- Java 2, Standard Edition (J2SE) is designed for desktop computers.
- Java 2, Enterprise Edition (J2EE) is a comprehensive platform for multi-user, enterprise-wide applications.
- Java 2, Micro Edition (J2ME) is a set of technologies and specifications developed for small devices like smart cards and mobile phones. J2ME uses subsets of J2SE components, such as smaller virtual machines and leaner APIs.

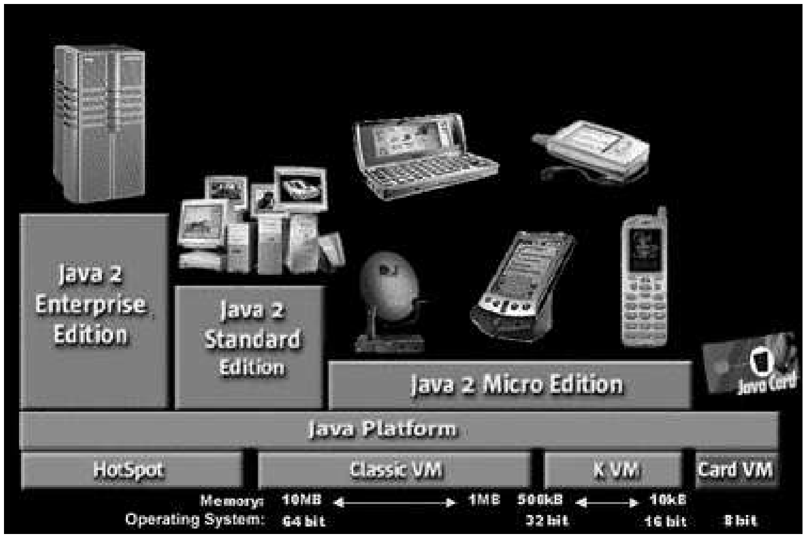

Figure 2-1. Java 2 Platform

2.1.2 Overview of J2ME

J2ME spans over a great variety of devices which differ a lot from each other. Therefore it would not be efficient to have software that supported all devices. J2ME needs to be flexible to cope with the problems of different devices and hardware configurations, the development of the technology for these devices and the range of different applications. This lead to that J2ME was created as a collection of technologies and specifications that are designed for different parts of the small device market. It's designed to be modular and scalable and is therefore divided into *configurations* and *profiles*. There is also an underlying layer consisting of a Java Virtual Machine (JVM) [JAV02][CMD00].

The Java Virtual Machine supports a particular J2ME configuration and is customized to fit a particular device [CMD00].

A configuration is a specification that defines the minimum Java libraries and JVM features that a developer can expect to find on all the mobile devices implementing the current configuration. A configuration, for example, might be designed for devices that have less than 512 KB of memory and an intermittent network connection. The set of APIs is customarily a subset of the J2SE APIs. To ensure portability across the device range, configurations cannot contain any optional features.

A profile is built on a configuration but adds more specific APIs to make a complete environment for building applications for specific devices. While a configuration describes a JVM and a basic set of APIs, it does not by itself specify enough detail to enable you to build complete applications. Profiles usually include APIs for application life cycle, user interface, and persistent storage. Applications written for a specific profile can be ported to any device that supports that profile [JAV02].

The different configurations and profiles are produced by the Java Community Process (JCP) by discussing and accepting different Java Specification Requests

(JSRs). The JSRs handle, among other things, new features to the configurations and profiles.

There are many different configurations and profiles as shown in Figure 2-2. J2ME tree. In the diagram, the finished specifications are shown with solid backgrounds, while specifications that are still under development are shown with dashed backgrounds. The corresponding JSR numbers are also shown.

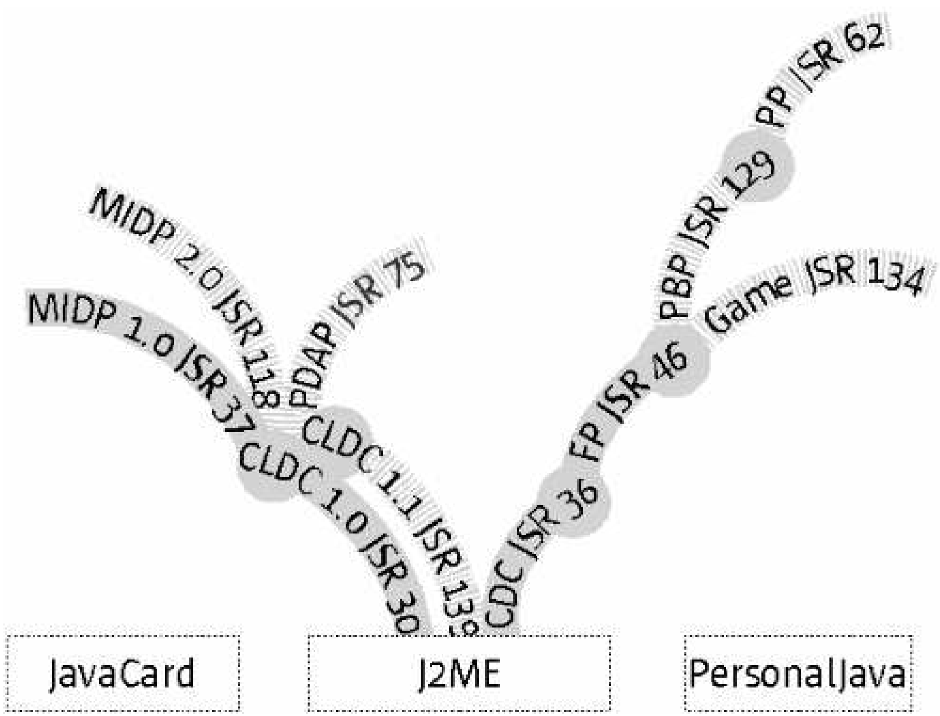

Figure 2-2. J2ME tree

The J2ME tree has two main branches Connected, Limited Device Configuration and Connected Device Configuration as shown in Figure 2-2:

Connected, Limited Device Configuration (CLDC). This configuration is for small wireless devices with intermittent network connections, like pagers, mobile phones, and Personal Digital Assistants (PDAs). These devices have a memory budget in the range of 160 kilobytes (KB) to 512 KB. Since this configuration is the one that were used in the master thesis at SmartTrust., the focus will be on this one. The CLDC includes a specification for a JVM that has been implemented by the K Virtual Machine (KVM), developed by SUN. The requirements for the CLDC were to find the lowest common denominator of Java technology so that it could be applicable to many different mobile devices, and to guarantee the portability of application code between different mobile devices. The size of the complete implementation of CLDC (java libraries and virtual machine) had to be smaller than 128 KB. The CLDC also specified that applications assumed to be able to run in as little as 32 KB of Java heap space. The CLDC has focused on security in several ways: Low-level-virtual machine security is achieved by requiring the downloaded Java classes to pass a class file verification step, applications are protected from each other by being run in a closed "sandbox" environment and classes in protected system packages cannot be overridden by applications [CMD00].

Connected Device Configuration (CDC). This configuration is for larger devices (in terms of memory and processing power) with robust network connections. Set-top boxes and internet appliances are good examples of CDC devices, although high-end PDAs like the Sharp Zaurus also fit this configuration well. The Foundation Profile extends CDC and serves as the basis for several other profiles. It provides fundamental APIs gleaned from J2SE, including classes and interfaces from java.lang, java.io, java.security, java.util, and more [JAV02].

The first finished profile was the Mobile Information Device Profile (MIDP), which is based on CLDC and thus is the first finished J2ME application environment. MIDP-compliant devices are already available.

The world of the Java platform and small devices also includes Java Card, for smart cards, and PersonalJava. PersonalJava, an application environment based on JDK 1.1, had an independent existence before J2ME was formulated. The Personal Profile will bring PersonalJava into the J2ME fold by making the next version of PersonalJava a CDC-based J2ME profile specification.

The software for a device usually consists of a configuration, a profile, and optional APIs. For the J2ME mobile phones of today this is made up by the configuration CLDC 1.0 and the profile MIDP 1.0 [JAV02].

2.1.3 Advantages of Wireless Java

There are several reasons why the Java Platform works well with wireless development. Here are the main qualities [JAV02].

- The Java platform is safe. Since the java binaries are executed in the JVM, it can be controlled so that it does not do unauthorized actions.
- The Java language encourages robust programming; it handles allocation of memory which decreases development time.
- The Java code has great portability. A single executable binary can run on multiple devices. A MIDlet (a MIDP application) can run on any device that implements the MIDP specification. Another advantage of the portability is the ease of downloading applications to a device.

2.1.4 Loading a Java application into a mobile device

To be able to download a java application to the mobile phone, the phone needs to have a Java application manager (JAM) that physically loads the midlet into the phone. The MIDP profile loosely specifies that a manager that handles download over a wireless connection, knows where to store the code and how to run it, is required. A scenario of a download from a Web page (or WAP page) of an application is the following:

The user selects the game from the Web page. A descriptor file, with information about the application, is sent to the user. The application manager (in the phone) checks if there is enough space left on the phone and that the version of the application isn't already stored. If everything is ok then the application manager downloads the application and stores it in the phone. It is then ready to be run by the user, see Figure 2-3[DAY01].

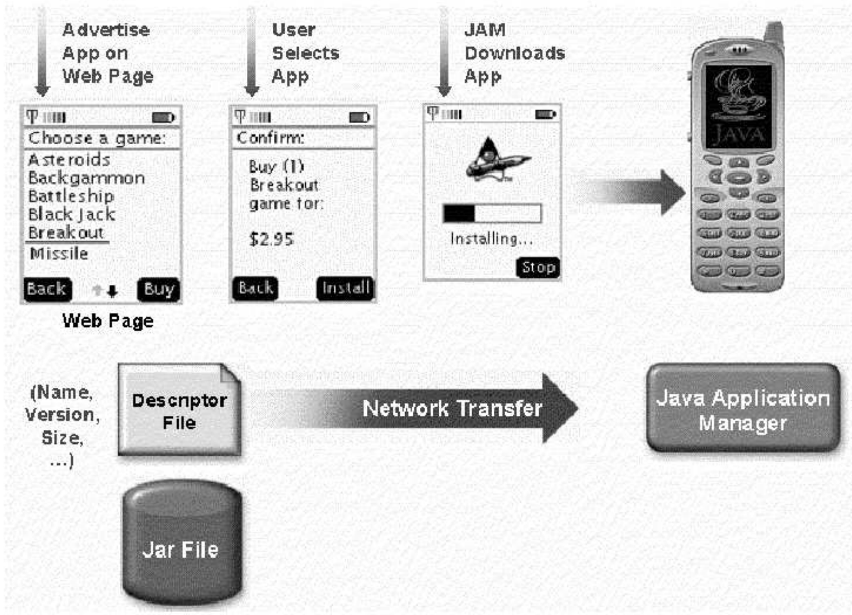

Figure 2-3. Download procedure of a Java application

2.2 SIM cards

2.2.1 Basics about SIM and smart cards in general

The type of smart card used by mobile operators in their customers cell phones is called a SIM (Subscriber Identification Module) card. Smart cards are generally divided into two groups' memory cards and microprocessor cards [EFF99]. Since the master thesis is only involved with SIM cards and those falls into the category of microprocessor cards thus memory cards will not be considered in detail. They are used to store data that can be read and written, typical usage areas are pre paid telephone cards and health insurance cards.

A smart card is like a small "computer" with the possibilities to store and do IO operations on data. So the basic building blocks are a microprocessor used for processing and the memory that consists of EEPROM, ROM, RAM and FLASH [JAC02]. Each card needs a power source either the card itself or usually a card reader. If it is the card itself that supply the power there must be some form of battery on the card. In Figure 2-4 there is an ordinary smart card configuration while on a SIM card the plastic support usually is much smaller.

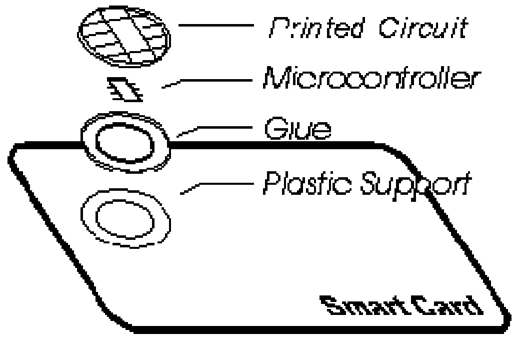

Figure 2-4, the different parts of a smart card [JAC02].

Since the card needs an OS (Operating System) it is burnt into the ROM when the card is manufactured. The EEPROM is the chip's non-volatile storage i.e. it is left there between uses of the card. Non-volatile storage can be used both for data and code and it is read and written under the control of the OS [JAC02]. As mentioned previously it is also possible to do IO through a single register through which data are transferred bit by bit. The RAM memory is used during a session and is not available at the next session as EEPROM is.

Microprocessor cards are very flexible since they may either contain a program for a highly tailored or predefined task and, on the other hand, they may have an OS allowing several different programs and at the same time allow several different application areas. The card OS may also allow updates and/or the installation of additional programs after the card has been personalized i.e. handed out to the end user.

The clock that the microprocessor on the card uses is not an internal generated clock instead it is the card reader that provides a clock to the smart card. Most smart card controllers allows the clock signal to be turned off when the CPU is in sleep mode to save power.

Security in ordinary smart cards

The main purpose of security regarding smart cards is to prevent unauthorized users to gain access to read or write data on the card. One big advantage that smart cards have compared to ordinary magnetic strip cards is that the PIN code is contained on the card and isn't sent on any communication channel that could be tapped [JAC02].

To not allow unauthorized access, a hierarchical file system is used. At the top of the file system is the master file MF. Below the master file there are elementary files and/or dedicated files, see Figure 2-5. To get access to a file it is necessary to pass through the hierarchy since it is the parent file that knows the access rules of the lower layer files. An elementary file can have no children instead it consists of a header and the data stored while a dedicated file can be viewed upon as a directory.

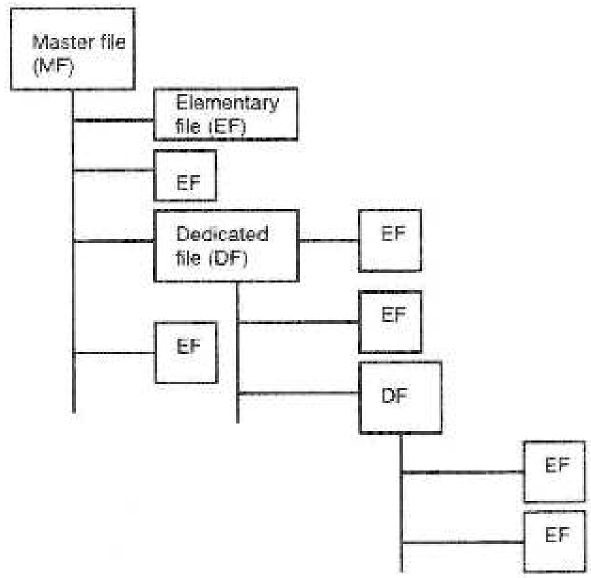

Figure 2-5 The hierarchical file system of a smart card [JAC02].

The way data are managed inside a file depends on the OS of the card e.g. it may just use offset and length or use a fixed or variable length of records as GSM do. To be able to use a file it must be opened or as it is called selected. When the power is turned on the master file is automatically selected there from it is possible to descend and later ascend in the file structure [JAC02].

The header of the elementary file specifies both status of the file as well as the particular access conditions or requirements of the file in question. The fundamental idea is to use PIN codes to verify access to a particular file. Depending on operating systems different levels of access control can be used. More fine grained solutions exists but these are the basic levels:

- **Always** file can always be accessed.
- **Card holder verification 1** (CHV1), access only if valid CHV1 value is presented.
- **Card holder verification 2** (CHV2), access only if valid CHV2 value is presented.
- **Administrative** these files are under the control by the one who has the administrative authority over the card.
- **Never**, access to the file is forbidden.

The PIN codes are usually stored in separate elementary files and if to many tries with the wrong PIN code are done the card will be blocked and can only be unblocked with the special unblock code. If the wrong unblock code is presented to many times the whole card will be blocked and thus useless.

Similar to ordinary computers the main flaw is the programs constructed to run on the smart card, since those can lead to weakened security, deadlocks etc. [JAC02]

GSM traffic is encrypted but there are still many ways to tap GSM traffic due to this some secure method is needed if a high degree of security is wanted. There are different methods e.g. some military officials as well as government officials uses GSM phones that scrambles the conversation when they talk about sensitive matters [UST02] or it could be done in a way so that only the most sensitive information is encrypted such as VISA numbers used for authentication etc.

There are a lot of different attacks available when trying to hack smart cards. Most of them will not be considered in the thesis since they are on a hardware level e.g. expose the smart card to abnormal voltage, since it has been shown that the random generator for cryptographic keys get almost all 1s [JAC02].

2.2.2 Java enabled smart cards

One of the greatest benefits with Java enabled Cards is that the OO design and JCVM let the applets to be encapsulated much easier than it was when smart cards were programmed in C or assembly. Because when cards were programmed in C or Assembly security consideration was needed to take the whole application into account, while Java Card can encapsulate data so only those that should have access can have it. Java Card applications cannot interfere with each other unless they explicitly want to and thus leading to code that is easier to understand and maintain [JCS01].

Java enabled smart cards are used to run J2ME (Java Micro Edition) applets. Since the capacity of a smart card is much less than that of a workstation the smart card only supports a subset of the usual features. In the Table 2-1 some differences are noted. Programs that are run at a smart card are called applets since they can be downloaded and run after the card has reached the end user with similarities to an applet on a website while for example many embedded J2ME systems doesn't have this feature of dynamic loading of programs [CHE00].

Table 2-1 Supported and unsupported Java features

Supported Java features	Unsupported Java features
• Small primitive data types: boolean, byte, short • One dimensional arrays • Java packages, classes, interfaces and exceptions • Java object-oriented features: inheritance, virtual methods, overloading and dynamic object creation, access scope and binding rules. • The int keyword and 32 bit integer data type support are optional.	• Large primitive data types: long, double, float • Characters and strings • Multidimensional arrays • Dynamic class loading • Security manager • Garbage collection and finalization • Threads • Object serialization • Object cloning

The “ordinary” JVM has in JCVM (Java Card Virtual Machine) been split into two parts one that runs off card and one that run on the card. Tasks that aren’t required to execute at runtime will be run at the workstation before deployment of the applet at the smart card [CHE00] e.g. optimisation, resolution and linkage aren’t required to run during execution. Off the card the converter is also run, which loads and preprocessces the class files that make up a Java package and then outputs a CAP Converted Applet File. The CAP file is then loaded on to the smart card and executed by the interpreter that resides on the card. How the CAP file is used see both the Figure 2-6 and the following paragraph.

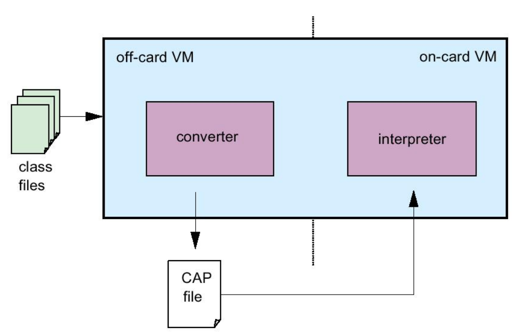

Figure 2-6, show the separation of the JCVM into on-card on off-card parts [CHE00].

The Java Card Runtime Environment (JCRE) consists of framework classes (APIs), industry extensions e.g. required for financial purposes, an installer used to install CAP files on the card (even after the card has reached the end user), system classes responsible for resource management, network communications etc, on the bottom closest to the hardware and the native system is where the JCVM resides together with native methods [CHE00] for a closer layout of the layers see Figure 2-7. Tasks for the JCVM are to execute bytecode, control memory allocation, manage objects and enforce the runtime security i.e. on-card system and applet security.

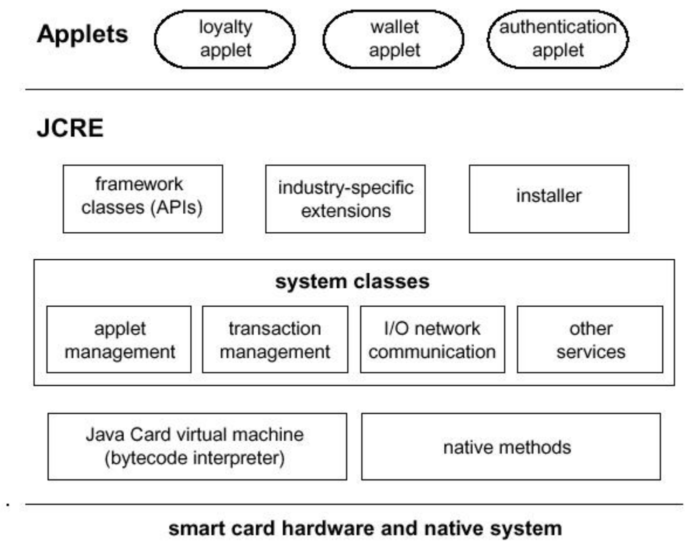

Figure 2-7, the layers of the JCRE [CHE00].

On a workstation the JVM is started as an ordinary process and ends when the process is killed, while on the smart card the JCVM is initialised when the card is created and doesn't "stop" until the card is discarded. To preserve information when the card doesn't have any power supply non-volatile memory (EEPROM) is used. It is also there that objects are created, which should be persistent between sessions. When the power supply is removed the JCVM simply suspends and when the power later comes back on the JCVM continues, but not where it last stopped instead it starts over with the main loop. If there are some transactions that haven't been completed the JCVM performs some clean up to come into a consistent state. The difference between being started again and initialisation is that objects have been created and stored on persistent memory.

When a smart card is inserted into a CAD (Card Accepting Device) it accepts APDU (Application protocol data units) commands, which are data packets exchanged between an applet and the host application (program at the card reader). At the time when the JCRE starts up it waits for an APDU command (e.g. withdraw money if there is a wallet applet on the card) that either states in the command which applet to run otherwise the APDU command is forwarded to the currently selected applet. The applet that receives the APDU processes the command and may send back an APDU to the host application and then gives back control to JCRE. This process continues when the next APDU command arrives. APDUs are sent between the card and the CAD using the serial communication interface on the card.

Besides the usual JRE the JCRE has support for three additional features:

- **Persistent and transient objects**. The default is that objects are created in persistent memory and are available between sessions. For security reasons transient objects can instead be created in RAM memory i.e. objects are only available during and not between sessions [CHE00].

- **Atomic operations and transactions**. JCVM ensures that all write operations are performed atomically i.e. either the value gets written or the old value remains. The API also has support for transactions where either all values get written or none of them.

- **Applet firewall and the sharing mechanism**. The firewall isolates applets so that each applet runs inside a designated space. Since the applet is running in a designated space the existence and execution doesn't have any effect on other applets. When applets need to share information or use JCRE resources it is the JCVM that permits such functions through secure sharing mechanism see the section *Applet Firewall and Object Sharing* for more information.

The Java API used on smart cards has also been largely reduced, but has instead gotten some special card extensions e.g. has the API for IO, communication and GUI been removed [JAC02]. The extensions added instead are:

- **javacard.framework**, the core package which defines classes such as Applet, PIN, APDU, System and Util that are fundamental building blocks as well providing runtime and system service.

- **javacardx.framework**, the package provides an object-oriented design for an ISO 7816-4 compatible file system. It supports elementary files (EF), dedicated files (DF) and file-oriented APDUs as specified in ISO7816, see previous section about security in smart cards for more information.

- **javacardx.crypto and javacardx.security**, these support cryptography that is required in smart cards, but they only define API interfaces and not class implementations. Instead it is up to the provider of the JCRE to subclass them and provide a suitable implementation, due to US export regulations regarding cryptography [CHE00].

In regular Java programs packages and programs are uniquely identified using a string and a naming scheme like the one on the Internet, while applets on smart cards are identified using an application identifier (AID) consisting of two parts: a resource id RID and a proprietary identifier extension PIX. The RID which is the same both for the package and the Applet should be 5-byte large and the PIX can vary in size from 0 to 11 bytes, but no two AIDs can be the same since it should uniquely identify a resource [CHE00].

There is some difference between applets that can be downloaded to the card and applets that are burned into the ROM during manufacture, this since ROM applets are allowed to define/call native methods while non ROM applets aren't allowed this. Native methods aren't controlled by the JCVM and thus could they be a security hazard. There is also something called preissuance applets i.e. applets that are downloaded to EEPROM before the card reaches the end user and these applets are treated the same way as ROM applets. Applets that later are downloaded by the cardholder aren't allowed to declare native methods since the JCVM can't control native code [CHE00].

To load an applet after the card has reached an end user a sequence of APDU (Application Protocol Data Unit) commands are sent to the card, the commands were previously created by the off card installer from the CAP file. At the time when the on card installer receives the APDU commands it writes the contents i.e. the CAP file to EEPROM and links the classes in the CAP file with classes on the card if necessary. If several packages are required the same procedure is repeated. As a last step an instance of the applet is created by a call to the `install` method:

```
public static void install(byte[] bArray, short offset, byte
length)
```

The install method is mandatory in the applet and has some similarities with the main method in an ordinary Java program inside the install method the constructor should be called to instantiate the applet. Both the installation parameters and their format are decided by the developer of the applet and sent together with the CAP file. Installation must be certain to only reference packages already installed at the card i.e. circular reference aren't possible e.g. if package A reference package B then package B must be installed first and aren't allowed to reference package A.

After the applet has been both initialised and registered together with it's AID at the JCRE (Java Card Runtime Environment) then it is possible to select the applet and run it.

If the installation fails due to some reason e.g. out of memory, card is removed from card reader etc. the installation is aborted and then it performs a clean up by reclaiming memory, remove any instances created i.e. it gets back to a consistent state before the installation [CHE00].

Applet Firewall and Object Sharing

Because it is common with sensitive information on smart cards such as electronic money, private cryptographic keys etc. it is essential that applets that shouldn't share data with each other cannot do so. To accomplish this with separation of data something called an Applet Firewall is used i.e. each applet is assigned a designated space and it cannot do anything outside that space. But if applets really need to share data or to use something like e.g. a phone book then it can be done through a secure interface.

The applet firewall protects against the most common security hazards such as lack in design and programming as well as hacking, since if an other applet gets a reference to data owned by an other package the applet can not read the data unless the other package explicitly has granted access to the data item [CHE00].

To achieve the above named security the applet firewall divides up the space into different contexts. Access is only allowed within a context and not between contexts. When an applet is instantiated it becomes a member of a context, see Figure 2-8 for an example of division into contexts. All instances within the same package share context and thus can read each other's data and interact directly with each other, but not with anyone outside the context (package). The JCRE is also assigned a context and from the JCRE context all other contexts can be reached but the controversy is not true i.e. an applet cannot access the JCRE context.

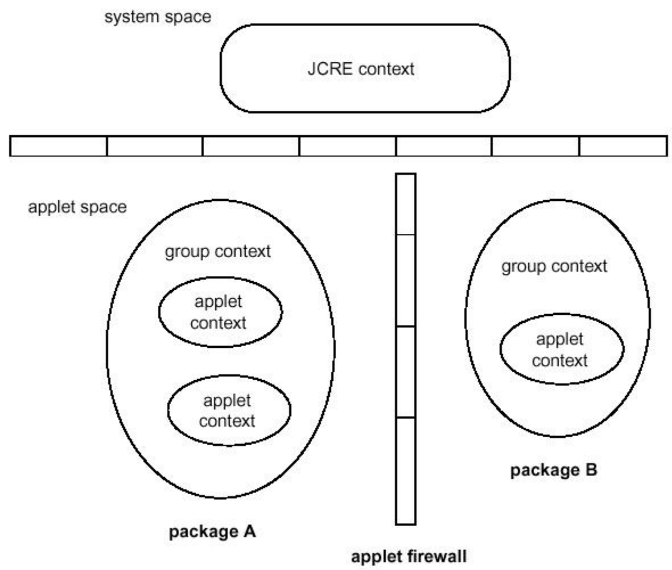

Figure 2-8, the use of the applet firewall and the division into different contexts [CHE00].

At any time there is only one active context i.e. the one executing either the JCRE or the context of an applet. When a new object is created it is assigned an owning context which is the currently active context. Primitive type static arrays are created and initialised by the converter before they are downloaded to the card and as such they are accessible by everyone in the package i.e. the owning context is the package.

When an object is accessed an access control is performed to see e.g. that not a private method is called from a public context and it's also here that is controlled that the applet accessing the object is allowed to access the object from the context it is executing in, otherwise a `SecurityException` is thrown [CHE00].

Since only instances are owned by a context and not classes, static fields are accessible by everyone, but the objects (including arrays) referenced belongs to a context and in addition to that ordinary rules applies such as if it is a private field only instances of the class may access the field. When a static method is called it keeps executing in the caller's context and thus all objects created become members of that context and all access controls are made with respect to the callers context.

To be able to share between contexts in a secure manner the basic idea is to use a context switch i.e. a applet calls the JCRE and the context switches to JCRE, since JCRE has special privileges it is possible to access any data item. A couple of different methods are used to achieve this context switch: JCRE privileges, JCRE entry point objects, global arrays and shareable interfaces and they are described in more details in the following paragraphs.

JCRE privileges, e.g. when JCRE receives an APDU command it calls the currently selected applet's `select`, `deselct` or `process` method and the context changes to the applets context and all objects created belongs to the same context as the applet.

Then the applet returns from the method and context is switched back to JCRE, sometimes several context switches are made since they can be nested.

JCRE entry point objects, to able to offer services that are accessible for everyone JCRE entry points are used, they are objects with public methods belonging to the JCRE context and that are possible to call from another context besides the JCRE context. When an entry point method is called there is a context change to the JCRE context, which performs the service before switching back to the callers context. Fields of the objects are still protected by the applet firewall i.e. only the public methods are accessible outside the JCRE context not any data. There are two types of JCRE entry point objects temporary and permanent objects. Temporary objects are not possible to store a reference to since they have special protection to prevent unauthorized reuse. Typical objects are the APDU and all JCRE-owned exceptions objects. On the other hand permanent objects are possible to store e.g. AID instances that are created at the same time as the applet.

Global arrays must be of a primitive type and can be viewed upon as a global memory buffer and they must also be a temporary JCRE entry point object and thus their references cannot be saved. There are a couple of global arrays that must exist e.g. the APDU buffer, which is cleared before a new APDU command is received or when an applet is selected to avoid leaking of sensitive information [CHE00]. Since there are no threads no consideration to synchronization to access of the buffer is needed.

Shareable interface is an interface that is somewhat similar to the RMI `Remote` interface in the case that both their implementations are empty and they need to be extended. The shareable interface defines a couple of methods that are accessible from a different context. A class that implements a subclass of the shareable interface is called a SIO (Shareable Interface Object). When executing in the context of the SIO it behaves like every other object, but when executing in an other context the SIO's methods declared in the shareable interface are accessible but not any other fields. Thus a shareable interface is a secure way to give access to some public methods defined in the shareable interface everything else is protected by the applet firewall. As described in the picture below: first the client applet request a SIO from the JCRE by providing a AID (unique identifier) number 1, then the JCRE invokes the servers method to retrieve the SIO number 2. When the servers `get` method is called the server is given the client's AID as a parameter and can based on that decide whether to grant service or not. After JCRE receives the SIO from the server number 3, then the JCRE sends the SIO to the calling client number 4. If either the server doesn't grant the SIO or if JCRE cannot find the AID null is returned to the calling applet. Since the client applet could give the SIO away to an other applet, a challenge may be needed in each public method at the server to verify that the applet's method call is allowed. A challenge could be that server sends a phrase to the client to encrypt and then check that it was encrypted correctly

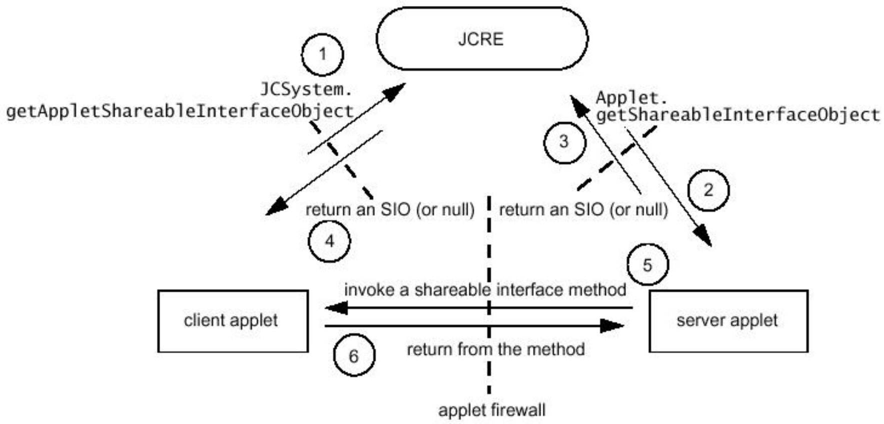

Figure 2-9, the use of SIOs [CHE00].

In the JCRE return values cannot be used in the same manner as in ordinary JREs, since the applet firewall may throw a `SecurityException` e.g. if a SIO has returned a reference to an object in its context to the calling client and the object returned isn't a SIO and used then the applet firewall will throw an exception. Thus the ordinary firewall rules apply to return values and only primitive values, static fields, JCRE entry point objects, global arrays and SIOs can safely be used as return parameters between contexts.

To avoid performance loss excessive contexts switches should be avoided and to reduce the amount of stack space used also a deep nesting of context switches should be avoided.

SIM Toolkit Applets

A mobile phone using a Java enabled SIM card doesn't need to know anything about Java. The interface seen by the phone is that of just an ordinary SIM card. All communication between the phone and the Java enabled SIM card is done using the exchange of APDUs as all SIM cards do. There are two types of applets that can be installed on Java enabled SIM cards, "ordinary" applets and SIM Toolkit applets. The difference between a SIM Toolkit Applet (STK Applet) is that a STK applet has the possibility to issue proactive commands e.g. display text on the ME device screen or ask the user to input something. A STK applet also has the possibility to construct it's own menu item that will appear on the ME device and the possibility to register for toolkit events e.g. listen to SMs.

To be able to handle proactive commands a small extension to the Java Card API has been made by ETSI and thus providing some special classes used in Java enabled SIM cards. On Java enabled SIM cards there is a GSM applet that manages all STK applets and also the communication with the actual phone using APDUs.

All STK applets must be installed with an AID of 16 bytes where the bytes 13, 14, 15 is the TAR value for the installed applet. The TAR value can be used to listen for SMs that are sent specifically to a certain applet on the phone, e.g. in a wireless wallet applet a SM could be sent to the phone that a deposit has been made then the applet has the possibility to deposit that amount to the balance stored on the SIM card. The

TAR (Target Application Value) value indicates to which application the data is sent its similar to a port in the IP world.

2.2.3 Wireless Internet Browser WIB SIM card

The WIB (Wireless Internet Browser), stored locally on the handset's SIM card, allow users to navigate simple menu-driven services and interact with mobile e-services delivered by the mobile operator via a Delivery Platform using simple SMS as the communication channel.

The WIB is an open specification done by SmartTrust. Over 20 different SIM card manufacturers have implemented the WIB and there are over 25 million SIM cards at end users [STW02]. To use the WIB there is no need for a WIG too see Figure 2-10, but without a WIG (Wireless Internet Gateway) only simple text based services could be offered e.g. retrieve the balance on a pre paid card. The job of the WIG is either to support requests form the user or to push data to the user.

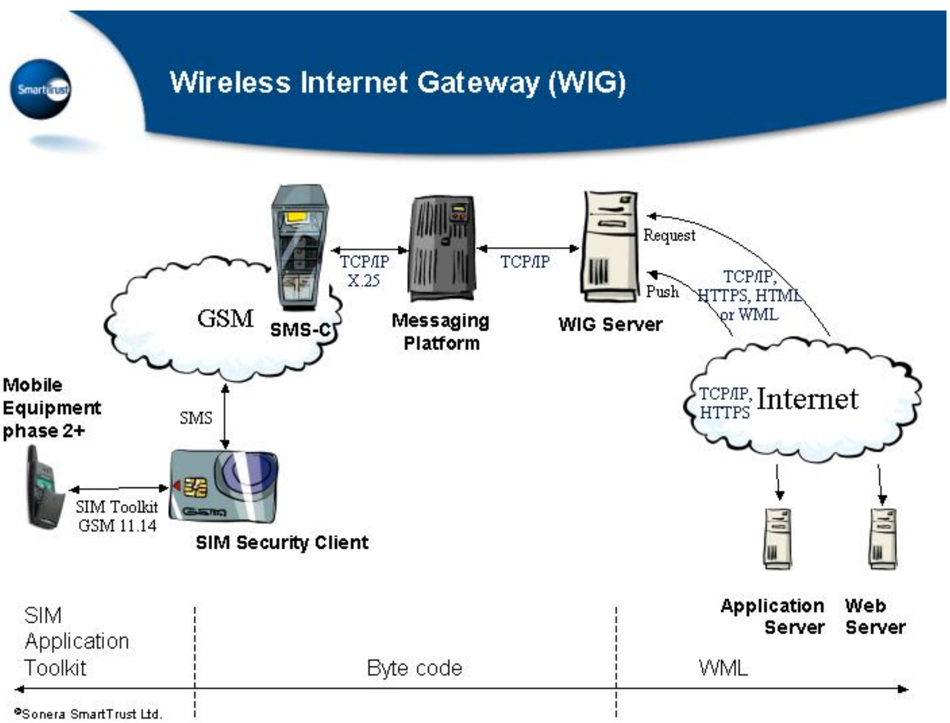

Figure 2-10, the interaction between the WIB on the SIM and the WIG server.

To develop a service, a subset of WML is used, when the WML page arrives at the WIG server it is encoded into a compressed bytecode that is understood by the WIB and is sent to the WIB through the SMS channel. As the bytecode arrives at the WIB it transforms it into SIM toolkit commands that almost all mobile phones understand since the last 2.5 years (2002). SIM toolkit is an ETSI/SMGE standard for GSM phones and it enables the SIM card to drive the handset e.g. a bill service like the one that was offered by Telia and Postgirot in Sweden [CEL02]:

1. A PIN code is needed to start the application.

2. Enter the postgirot account number, date and amount.
3. The SIM card packages this information into a SMS.
4. If the user wishes he/she might add a comment to recognize the payment in the future.
5. The card calculates an electronic signature.
6. The SMS is sent to Telias SMSC (Short Messsage Service Central) and is later processed by Postgirot's SmartSec system.
7. Once the bill has been processed, an acknowledgement is sent to the user.

Since SMS is used as a transport channel there can be quite large delays for the end user. The amount of data available in an SMS is around 120 bytes depending on the security level. If there is a large WML page that should be sent to the WIB several SMs can be used, but there is an upper limit usually around 5. When several SMs are used to send a single WML page the user experiences larger delays e.g. when around 5 SMs are used it can take as much as 25 seconds before the user's request has been answered [WAG02]. There is also a restriction in the WIB (depends on SIM vendor) that it's only able to send around two consecutive SMs with a payload of about 120 bytes.

The application at the handset can either be residing on the SIM card or fetched from the Internet and it is also possible for the operator to add, update and remove different services from the SIM over the air.

The WIB also specifies a plug-in interface, which can be used to call a plug-in from a WML page or provide additional features such as cryptography and file and data management. Plug-ins are useful for operators to add functionality that aren't available in the WIB.

2.2.4 SIM WIB with Java plug-in

WIB with Java plug-ins is a combination of two techniques i.e. Java Cards and the ordinary WIB, since in this implementation the WIB is implemented as an applet residing on the Java enabled SIM card.

Applets on Java Cards can access functionality from each other through the *shareable* mechanism, which allow access through the firewall that normally exists between the card applets. Figure 2-11 shows an example of an implementation of the Shareable interface, with WIBPluginInterface as the interface extending the Shareable interface. WIBPluginInterface contains the method *doAction()* that is accessible by other applets. The class WirelessWallet extends the JavaCard Applet and implements the interface WIBPluginInterface. By doing this it becomes an applet that can be accessed by the method *doAction()*. If another applet wants to access the WirelessWallet applet it calls the static method *getAppletSharableInterface(AID,byte)* with the AID value corresponding to WirelessWallet, on the javacard.framework.JCSystem. This method returns an object defined by the Shareable interface. Since it's known what interface to expect it's just to cast it to WIBPluginInterface and then use the method *doAction()*.

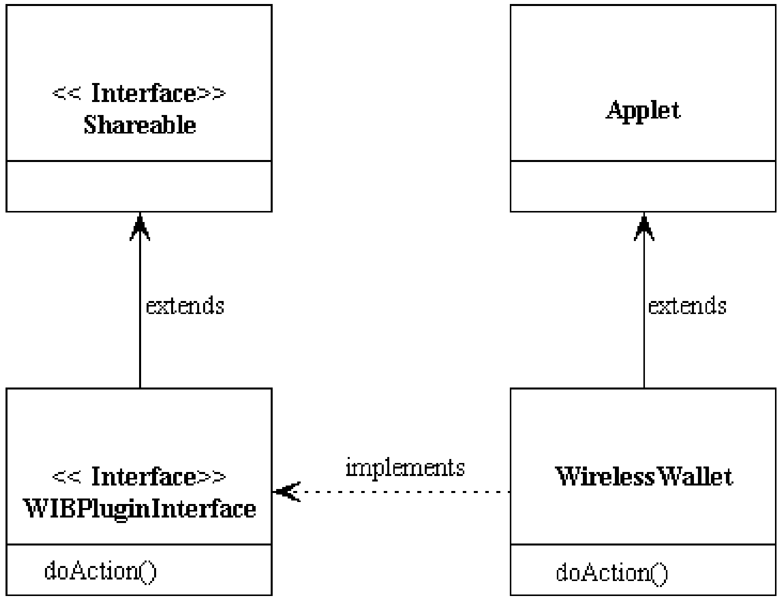

Figure 2-11. Implementation of the Shareable interface

2.2.5 Over The Air (OTA) overview

OTA is the procedure to do something on a SIM card that has been handed to the end user e.g. to add functionality through the download of an application to the SIM card or maybe to start an application that already is residing on the SIM card. OTA means that data messages are sent from the network to the mobile device to access data or start an application on the SIM card i.e. it's performed "over the air". ESMS (Enhance Short Messages Service) can be used for GSM file administration commands, application management, resident application triggering [G+02].

Since the things that you can do with OTA and ESMS requires some form of security the requirements that has to be fulfilled could be summarized as in Table 2-2.

Table 2-2 Security mechanisms in OTA [G+02]

Requirement	GSM 03.48 Mechanisms
Authentication	Cryptographic checksum
Message integrity	Redundancy checksum
Replay detection and sequence integrity	Counter
Proof of receipt and execution	Acknowledgement
Message confidentially	Encryption

Checksums are used to count the number of bits in a transmission so that the recipient can make sure that the correct number of bits have been received and are intact. All these "features" aren't necessarily used instead there are some bits in the SM header that indicates whether cryptographic checksum and so forth are used. The type of redundancy checksum used is XOR, while the crypto checksum is DES-3DES but digital signature isn’t implemented.

When an ESMS is sent to the SIM there is a TAR (Target Application Value) value that indicates to which application the data is destined for its similar to a port in the IP world. The process of sending data to or alter the existing data on the SIM card is controlled by the mobile operator owning the SIM.

3 Development and evaluation

The main idea with the evaluation was to have a typical application and to develop this example application in all of the different technologies and to use it as a "framework" for the evaluation and benchmarking.

The example application is described in section *3.1 Application* and the evaluation it self is in sections *3.2 WIB evaluation* and *3.3 Java Card evaluation*.

Results from Peter Lindströms thesis [PEL02] has been added in *3.4 Java in the phone* and *3.5 WIB with Java plug-in evaluation*even even though they aren't complete, due to their relevance for the comparison of different architectures

3.1 Application

Throughout the evaluation of the different architectures the idea is to develop and use the same type of application. One suitable application for this is an application that requires some security and at the same time a bit of processing to be able to see how the architectures are compared to each other.

The application chosen is a pay service (wireless wallet) for micro payments (a small amount per transaction), since it's concerning micro payments e.g. a scheme with a VISA card couldn't be used due to it costs too much for each transaction instead a scheme similar to cash cards were decided to be used. With the cash card scheme there is a rather low cost for each payment so it's suitable to use even for small payments.

3.1.1 Basic Requirements

The functionality that is desired from the application should be that of a wireless wallet. The functionality selected is:

- One should be able to withdraw money from an account connected to the cell phone.

- There should be an identification of the user so money is withdrawn from the correct wallet.

- The wallet should inform the requesting party somehow that money has been withdrawn.

- The information sent by the application may be encrypted or in some way secured.

3.1.2 Use Case examples

The idea is to have a wallet that "applications" can take money from and that the transaction are cleared/signed by the owner of the wallet. These use cases described below are some ideas of how the wallet could work.

Get a stock analysis

The user has a stock analysis application stored on the SIM and wishes to view a certain analysis. A request for the analysis is sent to the content provider, which pushes the analysis to the user as soon as payment has been verified.

In the WIB case the following happens when the user requests the stock analysis: a signing request, containing information about the amount to pay and what it is for, is sent from the bank server to the user. If the user wants to pay, he/she signs the request so that information about the signing is sent back to the server that verifies the signature. Once the "bill" has been paid the user will receive a SM containing the analysis.

If a Java enabled SIM card is used and the user requests a stock analysis the stock analysis applet will activate the wireless wallet applet, which verifies that the user has a sufficient balance. If there is enough money in the wallet the applet request a PIN from the user before it executes the transaction. When the transaction is completed information is sent from the wireless wallet applet to a bank server so that money could be transferred to the content provider. The wireless wallet applet also informs the stock analysis applet that money has been withdrawn and then it is up to the stock analysis applet to request the analysis from the content provider.

In the WIB Java plug-in case the user will, after trying to access the restricted page, receive a page that activates the Java plug-in at the Java Card. The plug-in will check in some *money file*, or similar, to see that the user has sufficient balance and will then request a pin from the user before it executes the transaction. If the transaction is completed, information is sent back to the server. The user can then view the restricted page.

Open up a level in a game

The user has a game that he/she has previously downloaded to the phone and now wishes to open up a new level.

In the case of Java Card the user selects the option in the game that says upgrade or something similar. This will activate the wireless wallet applet at the Java Card that begins to see that the user has sufficient balance and will then request a PIN from the user before it executes the transaction. If the transaction is completed information is sent to a bank server so that money could be transferred to the content provider. The wireless wallet also communicates with the game so that the user will get access to the new level.

In case of Java in cell phone the user will select the upgrade option in the game applet and will then be asked to enter username and signing PIN. The username and signing PIN will then be sent to some server (game or bank) for verification. The server checks to see that username and signing PIN are ok and the user's credibility is sufficient. Information about the result is sent back to the applet. If the verification was successful the new level will be opened for the user.

3.1.3 Limitations

To avoid the problem that the implementation of the application becomes too large, from a development perspective, the focus will be on the mobile device and the connection between it and a server. On the server side a "black box" technique will be used in some cases, i.e. an application on the server side may not be fully implemented, e.g. money transactions between the wallet and the requesting party will not really take place instead some kind of notification will be implemented so the requesting party is notified that money has been withdrawn.

3.1.4 Security

To protect the data sent between the server and the client some kind of encryption of the data should be used. One of the most common algorithms is DES (data encryption standards) that is a symmetric crypto algorithm i.e. the same secret keys is used both to encrypt and decrypt. DES is also a block cipher algorithm i.e. output has the same size as the input. An evolvement of single DES is triple DES.

To encrypt a message with triple DES the following is performed $C=E_{k3}[D_{k2}[E_{k1}[P]]]$ where E_{ki} is the key used for encryption and D_{ki} means that the decryption scheme is used and P being the plain text. The only idea with using the decryption scheme is that it allows users to decrypt data that was encrypted with single DES. The methods used when trying to break crypto algorithms are brute force and statistical analysis and in 1998 a DES cracker was showed, which could crack single DES in less than three days. To day it would probably be even faster and cheaper [STA00]. [STA00] is also a good starting point for an introduction into the areas message authentication, message integrity and encryption.

When it comes to security in the GSM world there is always a basic security since 03.48, the GSM standard, encrypts traffic. It's not especially safe but at the same time not any ordinary person could decrypt it, but it would be possible if someone enough educated wanted to. For further details regarding the security features on the GSM layer see *2.2.5 Over The Air (OTA) overview* or look into the specifications available from ETSI (European Telecommunications Standards Institute).

3.1.5 Evaluation technique

The idea is to have the wireless wallet as a framework for the evaluation, i.e. all technologies well be evaluated with this common denominator and thus evaluating the difference in communication, security features, performance, development environment and learning threshold.

Communication, here the following aspects will be considered: the architecture's need for communication, set-up time for the communication channel and how the communication medium responds to different amounts of data.

Security features, some of the architectures support encryption, message integrity and message authentication directly "inside" the architecture while other must implement itself thus these factors will be considered.

Performance, since there is a rather big difference between the different architectures the performance both when it concerns matter like encryption as well as communication will be looked upon.

Development environment, there will probably be a significant difference between the architectures, since some offer little choice to the developer while other offer a lot of different tools. This is also one of the most important factors, sine it's what the developer will experience every day.

Learning threshold is also a very important aspect, since training might cost a lot and thus it takes time until the developer is productive.

3.2 WIB evaluation

3.2.1 Application design

The main idea with the design is to have some form of pay service (wireless wallet) available from the cellular phone. The actual account that resides at a web server is here after commonly called wallet server, wallet or bank. Responsibilities of the bank are to first accept incoming bills from different sources e.g. a content provider on the Internet. Secondly once the bill arrives from the issuer (e.g. the content provider) to send a payment request to the one that should pay it accordingly to the issuer. Finally once a bill comes back to the bank from the one that should pay it. The bill is verified and the issuer of the bill is informed so that he/she can take an appropriate action.

The process described in Figure 3-1 is where a user requests some restricted information needed to pay for from a content provider and steps taken to perform the whole transaction. If a user chooses not to pay for a bill request sent to his/her phone she simply discards the requests and the processing is stopped at number 3 in Figure 3-1 i.e. nothing is sent back to the wallet server. Either some form of mechanism should be used to flush bills that have been pending too long or to let the user choose among issued bills to pay.

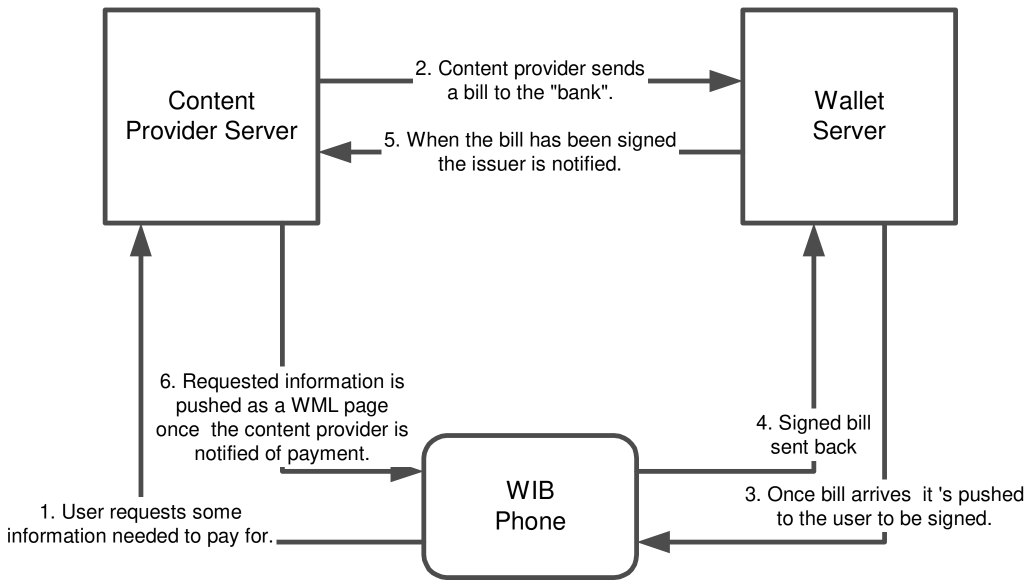

Figure 3-1 The process when a restricted page is requested and paid for.

The solution chosen isn't just limited for initiation by a phone you could easily think of other scenarios where the initiation is done through some other means e.g. a browser on a desktop computer or a candy machine where they connect to the wallet server over the Internet. Then the wallet server sends a bill request to the phone. Once the bill has been paid then the issuer e.g. the candy machine is informed of the payment and it releases the candy.

A bill is identified through a unique identifier created by the content provider according to a scheme like time@contentprovider the same identifier is then used throughout the lifetime of the bill. Each user has an account at the wallet server that is identified through the user's MSISDN in addition to the account number (MSISDN) there is a balance and a PIN code, which must be given by the user when he/she wants to pay a bill.

How do the content provider know which bank to send the bill to? Here a couple of different schemes are possible. Easiest to implement is to let the user inform the content provider either as he/she register for the service which bank to use or just send the bank to use as a parameter to the content provider in the first step in Figure 3-1.

As the bill is sent to the wallet server it is sent together with a MSISDN of the one to pay the bill and a unique identifier for the bill. As the wallet server has received the payment, it notifies the content provider about which unique bill that has been paid so the content provider can take the appropriate action.

So the information to be sent to the user is the unique identifier of the bill along with a description so that he/she knows what to pay for. Then if the user chooses to pay the bill he/she types in their PIN that is sent back to the wallet server together with a unique identifier of the bill. A user might have a couple of different bills to pay the unique identifier is used so the server knows which bill that is to be paidFigure 3-1..

For each transaction the balance and password are checked before money are withdrawn and information about bill payment are sent to the issuer of the bill.

3.2.2 Security design

There is both a plain text version of the stock analysis application and one version, which use encryption for the essential data parts, password and the unique identifier of the bill, with the communication between the phone and the wallet server.

Encryption used between the WIB and the Wallet server is 3DES see *3.1.4 Security* or [STA00] for further details.

When the bill arrives at the WIB phone it sends it's password encrypted to the server, which decrypts it and checks whether the password and the user id matches. If the password and the user id were matched money are withdrawn and the issuer of the bill is informed of the payment.

The encryption and decryption on the server side is done by using a Smarttrust product known as the security center. To encrypt or decrypt something a TCP connection to the security center is set up and the type of request (encrypt/decrypt), MSISDN and data to be encrypted/decrypted are sent over the connection once the request has been performed the data is sent back. Since the encryption/decryption is done in both ends of the connection it's possible to have a secure end-to-end connection without plain text in any intermediate place.

In addition to the security offered by the GSM layer see *2.2.5 Over The Air (OTA) overview*, the WIB offers extra security by giving the possibility to sign or encrypt the information using P7, 3DES, etc, and thus providing a very good security as long as the handling of the keys is done securely.

One commonly used method to find a specific key is that the SIM card manufacturer uses a special scheme when manufacturing a batch of SIM cards. For each batch the specific key can be derived from the MSISDN using a master key i.e. to find the specific key for a SIM card an algorithm is used that uses the MSISDN and the master key to calculate that SIM card's key.

3.2.3 Implementation and limitations

In a real world implementation, money really has to flow to the content provider e.g. that the wallet server constantly transfers money for each transaction to the content provider's account or once in a while a bigger amount. This has not been implemented in any way. But all bills that have been paid are stored in a database so it could later be possible to do an extension that actually transfers money between different accounts.

The implementation has been done in a way so that both the content provider server and the wallet server are run on the same web server and the communication between them is done through direct references, but it would be rather easy to implement some form of Inter Process Communication between them. This communication between the content provider's server and the wallet server have been done simple and without any form of encryption, since the focus of the thesis rather lies on the communication from the phone to the bank. Communication between the wallet server and the content provider could be done in a number of more or less secure ways e.g. SSL, IPSec and other systems that are commonly used for secure communication over the Internet. Here a consideration must be done of what level of security that is needed and then do an investigation of the available techniques and choose a suitable one.

3.2.4 Evaluation

Stock analysis program used for the case study and evaluation.

For the evaluation of the wireless wallet an example program in the form of a stock analysis program was developed. The stock analysis program could also work for the analogy of paying for information and in this case paying for the analysis of a stock or a restricted page.

In the prototype application a WML page containing the stock analysis program is stored on the SIM card in byte code format according to the WIB specification. Since the WML page is stored on the WIB SIM card it appears on the phone as a menu item. Once the user selects the stock application on the phone he/she can choose to either request an analysis of Ericsson, Nokia or search for any other stock that the content provider offers an analysis of. After a stock has been chosen the user selects whether to use plain text or 3DES in the communication with the bank.

If the user chooses the stock analysis item on the phones menu and then e.g. an analysis of Ericsson the WML page stored on the SIM sends a request together with a parameter of which stock analysis he/she requested to a JSP page located at the content provider. The location of the JSP page was given at install time of the application on the SIM card.

As can be seen from Figure 3-2 all communication with the WIB phone goes through the WIG server before connecting to either the content provider or the wallet at the web server.

It could be said that both the content provider and web server essentially is built up of a couple of JSP pages, Java Classes and a database for the paid and unpaid bills.

At the time the content provider has issued the bill and informed the wallet server of the new bill. The wallet server knows which MSISDN to push the payment request to. A push is accomplished by connecting to a WIG server, which listens for push requests on a certain port. See Figure 3-2 for how the communication between the different parts is made.

As the pushed bill arrives at the user, he/she sends it back together with his/hers encrypted password and unique identifier. When the bill arrives at the wallet server a connection to the security center is set up, but the communication with the security center is done using plain text communication over the socket even though it is possible to use SSL. It would be a rather simple matter to use an SSLSocket instead.

As the bill has been verified the content provider is informed and the analysis is pushed to the WIB phone.

Testbed

The testbed used both for the performance and security evaluation (i.e. evaluation of the security features available/used for the WIB) were made with the stock analysis program as a base. The most important parts are described in Figure 3-2 in some places of the test bed it was hard to do measurements since e.g. there was no possibility to make timestamps from the "inside" of the WIB phone. With the wallet it's rather easy to do it from inside the code. All measurements involving the sending of SMs had to be measured manually while others like encryption at the web server could be measured from inside the code.

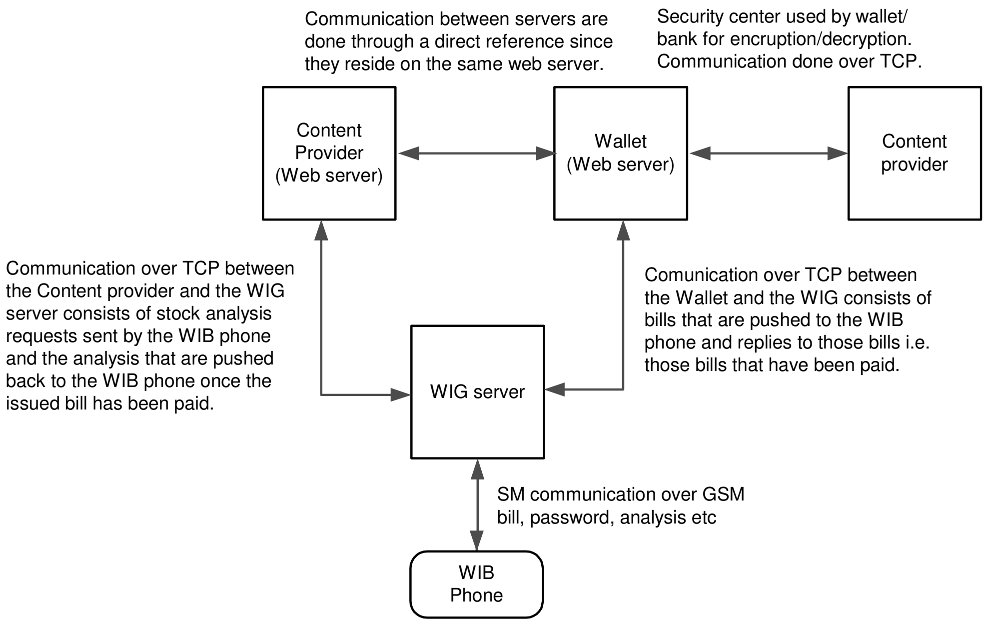

Figure 3-2 The Testbed used.

The total time was measured and divided into different parts. Figure 3-3 shows the parts that were measured individually.

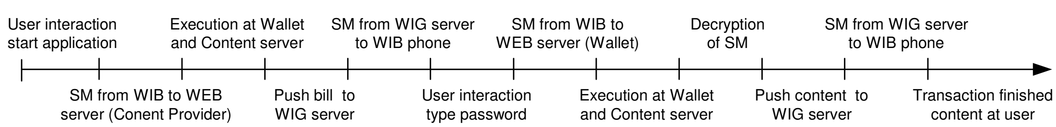

Figure 3-3 the total execution time for a stock analysis session divided into it's different parts.

First the total time were measured and divided into its different parts using both 3DES and plain text for the communication between the WIG and WIB.

Then a detail study of the different parts were made to see how size would effect encryption/decryption time both at the SIM card and server side, also the time taken to push different sizes of WML pages to the WIB were investigated.

In the following paragraphs the evaluation results are handled in detail and a summary is available in the section called *Evaluation Summary*.

Estimation of time distribution in the wireless wallet case study

Before the evaluation and measurements were done an estimation of the time spent in the different parts during the execution of the stock analysis program were done as showed in Figure 3-4. Dashed areas should be much smaller and it can clearly be seen that the most time is assumed to be spent by sending SMs and engaging in user interaction together with the encryption/decryption of data. Since the time spent in doing encryption/decryption is rather short there shouldn't be such a large difference between using plain text and 3DES in the communication between the wallet and the WIB phone.

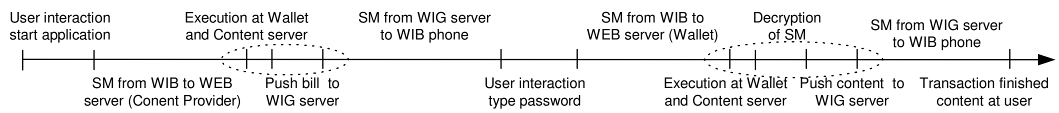

Figure 3-4 Estimation of proportions where time is spent during a transaction, dashed areas are "zoomed" in i.e. they should be much smaller compared to the other.

3DES/Plain text execution time

Measurements both for the 3DES and the plain text case were done using both logging of times that were of interest from inside the code and timing done manually by a human since there is no way to measure execution time on the phone. To measure the time period when a SM was in the air the measurement was started as the user pressed the final button in the user interaction and the time was stopped as the request arrived at the web server. In the same way the time from the web server to the phone was measured i.e. timing started when the web server pushed the message to the WIG server and the timing was stopped when the message was displayed on the phone.

For both the 3DES and plain text case there were 11 measurements done i.e. in total 22 measurements. More measurements could be made to further ensure that the values lies around the values presented in the tables, but it's hard to automate the testing since there is no easy way to automate the measurement of the time it takes to send a SM. Instead a manual measurement must be made each time an SM is sent.

Table 3-1 Measurement of times in stock analysis program using encrypted communication between WIB and WIG.

3DES (all times in ms)					
Action	Average	Max	Min	Median	Std. Deviation
User interaction start app.	8149	15653	6149	7261	2954
WIB send SM to Content provider	7955	11918	6350	7551	1689
Whole processing (Content & Bank)	42	80	30	40	14
Push Bill (Wallet server)	27	40	20	30	6
SM from Wallet to WIB	15437	17966	12047	15352	1833
User interaction type in password	4730	6810	3655	4446	983
WIB send SM to Wallet server	12792	15493	11276	12238	1562
Whole processing (Bank & Content)	406	481	311	411	56
Decrypt SM from User	366	441	271	371	53
Push Content (Content server)	24	30	20	20	5
SM from Content provider to WIB	8513	9984	6460	8562	1074
Total time for transaction	57575	65505	52415	55420	3962
Total time for transaction minus user interaction	44696	48159	42500	43894	2034

Table 3-2 Measurement of times in stock analysis program using plain text communication between WIB and WIG.

PLAIN TEXT (all times in ms)					
Action	Average	Max	Min	Median	Std. Deviation
User interaction start app.	7803	9964	6559	7872	981
WIB send SM to Content provider	6728	8151	5668	6679	599
Whole processing (Content & Bank)	36	50	20	40	8
Push Bill (Wallet server)	26	40	10	30	9
SM from Wallet to WIB	16874	22552	13880	15733	2909
User interaction type in password	4673	5849	3545	4927	862
WIB send SM to Wallet server	6597	7971	5678	6349	783
Whole processing (Bank & Content)	45	90	30	40	16
Push Content (Content server)	35	80	10	31	18
SM from Content provider to WIB	8337	12748	6719	7681	1751
Total time for transaction	50952	57322	44063	51053	4132
Total time for transaction minus user interaction	38476	44083	33899	37747	3389

A high standard deviation for the user interaction and sending of the SMs can be seen for both the plain text and 3DES cases in Table 3-1 and Table 3-2. That the user interaction has a high standard deviation isn't especially surprising since the time it takes for a user to press the buttons needed may vary rather much. The sending of the SM having a high standard deviation may have many reasons e.g. the workload of the SMS-C (the server responsible for the handling of the SM), traffic in the network, reception quality (retransmissions), the one thing that have the largest impact is the traffic level in the network closest to the phone.

The times that are the most interesting in aspect of the user is the total time for the transaction with the user interaction time removed since the pressing of buttons isn't experienced as a large waiting time. It's also good to remove it since it has a high standard deviation. When removing the user interaction time during the comparison between the plain and 3DES case only the technical aspects that inflicts time are compared and not the time a user spends on typing e.g. his/hers password.

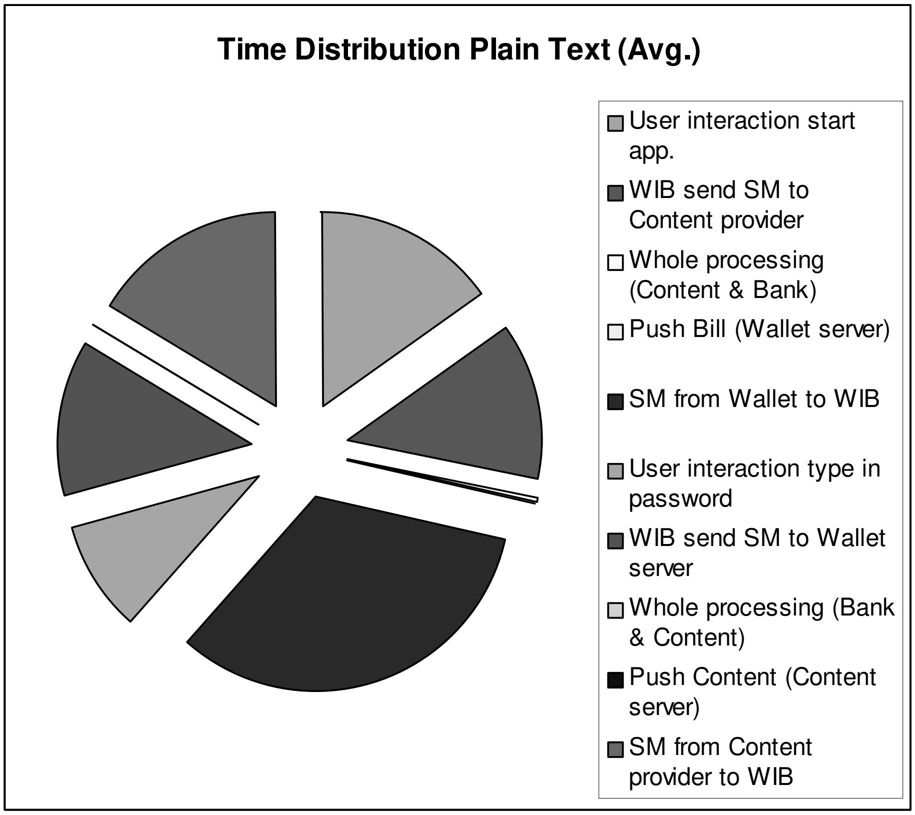

Figure 3-5 Time Distribution during a session using plain text for the communication.

From both Figure 3-5 and Figure 3-6 it can easily be seen that the largest contributor to the total execution times is the communications part i.e. the sending of SMs. The user interaction time also contributes fairly much to the total time, but that time isn't experienced that much as waiting time since the user is doing something.

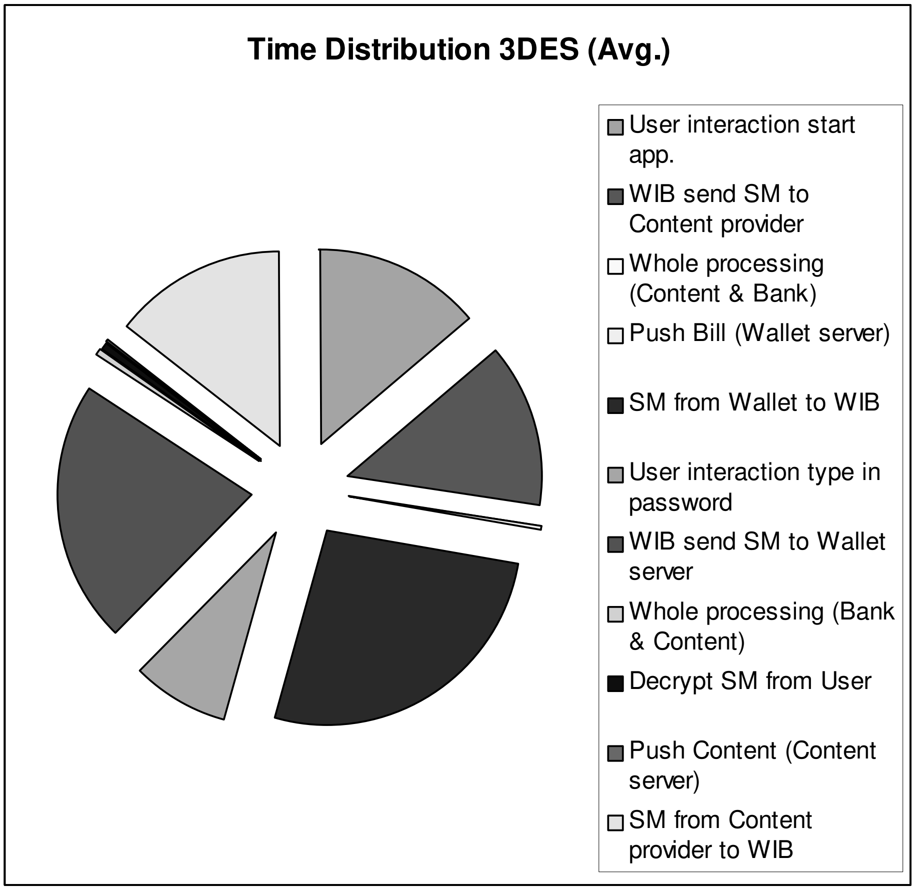

Figure 3-6 Time Distribution during a session using 3DES for the communication.

In Table 3-3 there is a comparison between the plain text and 3DES case, it was performed by normalizing 3DES to plain text i.e. taking the quotient 3DES/(plain text) for the median and average values. In the normalized case the plain text is faster if it's a value above 100 otherwise it gets a value below 100. Because not just the relative difference between them are important the same thing was done by taking the following subtraction 3DES-(plain text) for the average and median values, i.e. a positive value means that the 3DES case takes that much more time and a negative value the opposite that the 3DES case is that many ms faster.

Table 3-3 Comparison between 3DES and plain text execution time where 3DES is normalized to plain text also the real difference in ms is taken between them.

Comparison of times in 3DES &PLAIN TEXT cases				
Action	Normalised 3DES/plain text		Real difference (ms)	
	Average	Median	Average	Median
1. User interaction start app.	104	92	346	-611
2. WIB send SM to Content provider	118	113	1227	872
3. Whole processing (Content & Bank)	115	100	5	0
4. Push Bill (Wallet server)	103	100	1	0
5. SM from Wallet to WIB	91	98	-1437	-381
6. User interaction type in password	101	90	57	-481
7. WIB send SM to Wallet server	194	193	6196	5889
8. Whole processing (Bank & Content)	906	1028	361	371
9. Push Content (Content server)	68	65	-11	-11
10. SM from Content provider to WIB	102	111	176	881
11. Total time for transaction	113	109	6623	4367
12.Total time for transaction minus user interaction	116	116	6220	6147

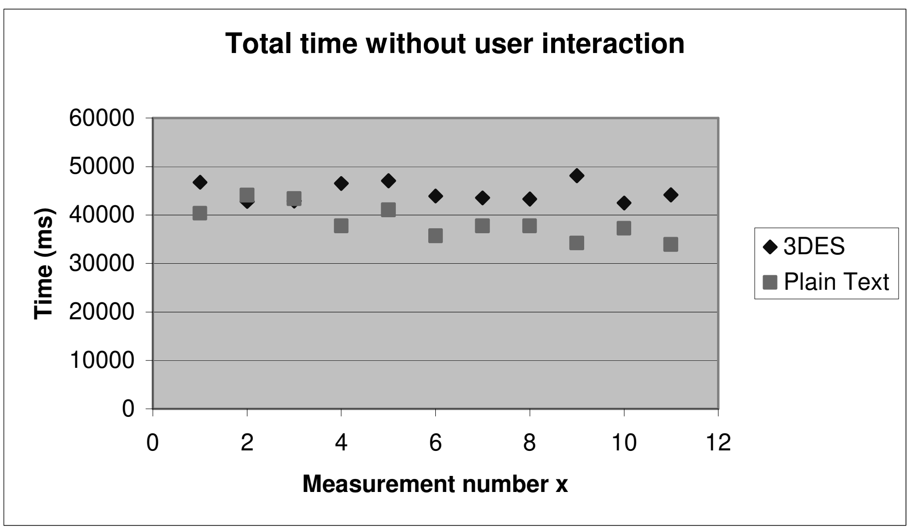

Figure 3-7 Total execution time in the stock analysis example with user interaction time removed

From Figure 3-7 it can be seen that 3DES usually takes longer time but the times are rather stochastic.

If just looking at the proportion in Figure 3-8 it first looks like the biggest contributor in the 3DES case is the decryption of the encrypted message from the WIB, since the execution time at the bank and content provider is almost 10 times longer than in the plain text case. Looking at real time contributors in Figure 3-9 it looks like the biggest contributors are the sending of the first SM from the WIB (initiation of the application), the last SM sent to the WIB (sending of the content to the WIB) and the sending of the encrypted SM from the WIB to the wallet server. There is no difference between the last SM sent in the two cases and when it comes to the initiation SM the only difference is that the initiation SM uses a different URL, which is 4 bytes longer. At first it seems a bit strange to have this difference, but it's explained due to a couple of extreme values that inflicts this difference for both the cases. But the difference of the sending of the encrypted SM from the WIB to the web server could not be explained in this way since this sending always takes around 6 seconds longer than the non encrypted version. In this case it's because of that 2 SMs are sent, due to the padding performed by the 3DES plug-in.

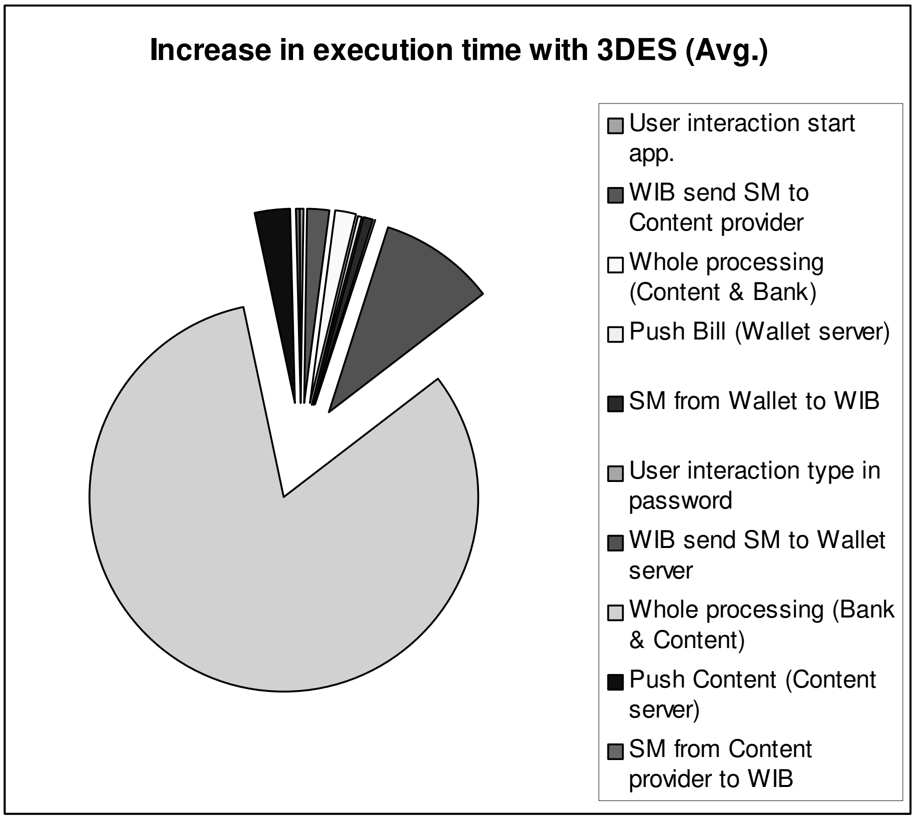

Figure 3-8. Increase in execution time with 3DES i.e. which part has the largest increase in execution time compared to when plain text is used. Values have been taken from Table 3-3.

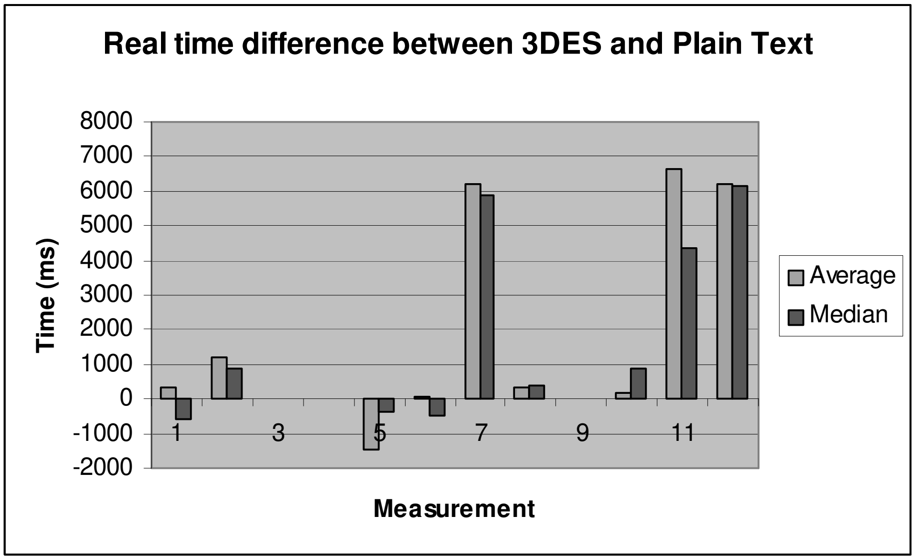

Figure 3-9. The real difference in execution time between the 3DES and plain text case. See Table 3-3 for which measurement the different numbers corresponds to. A positive value means that the 3DES case takes that many more ms and a negative value the opposite.

As expected there wasn't a large difference between the plain text and the 3DES case when looking at the mean and median values for the total execution time with the user interaction time removed, since it then took around 16% longer time to use encryption or in real time approx. 6 seconds longer.

The assumption made before the evaluation was started that the biggest contributor in the 3DES case was going to be the communication with the security center was correct in the sense it was the largest relative contributor, but when it came to the largest contributor measured in real time the sending of the encrypted message from the WIB surprisingly turned up.

SIM card encryption/decryption time

The first measurement performed was to try to find out the encryption time when doing encryption on the SIM card. It was done by first encrypting some data and then measuring the time until the encrypted data could be displayed on the device.

It was discovered that the encryption/decryption was really fast so to be able to measure the time 8 consecutive encryptions were done and then the total time minus the reaction time was divided by 8 to calculate the time for a single encryption.

Because of that there is a typical restriction in two SMs that could be sent from the WIB and each can contain around 120 bytes. The idea was to measure the encryption time from around 10 bytes up to around 200 bytes, the upper limit on the amount of data that can be sent. But the attempt to measure the encryption/decryption time for a message using the size 200 failed. It wasn't able to handle that large amount of data at least if you tried to do 8 consecutive encryptions. So a decision was made to just measure the time for encryption up to 150 bytes since at least there was never going to be more than that to encrypt in the case study.

Table 3-4 Measurement of encryption/decryption time in a SIM card, there were 5 measurements made each size.

SIM Card Encryption (all times in ms)					
Amount data to enc.	Average	Max	Min	Median	Std. Deviation
10 bytes	118	128	109	119	7
64 bytes	164	179	153	163	10
150 bytes	232	237	228	233	4

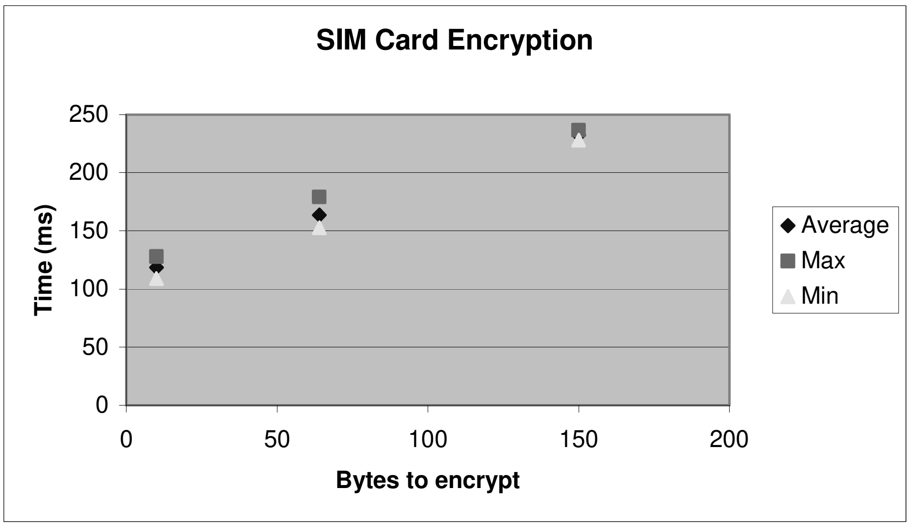

Figure 3-10 Time to encrypt data on the SIM card as a function of the number of bytes.

When looking at the times for encryption on the SIM card it looks rather clear that the encryption will not effect the final execution time that much regardless of the size to encrypt. As can be seen both in Figure 3-10 and Table 3-4 the time seems to increase rather linearly with more data to encrypt, but the encryption time will still be very small compared to time spent with communication. Due to the upper limit on how much data the SIM card is able to encrypt the real time will still be small even though something at the upper limit is encrypted.

Wallet server encryption/decryption time

Encryption/decryption on the server side was done using a Smarttrust product called the Security Center, which communicates with the wallet server over a TCP connection. The connection setup time was measured as well as the time it took for the sending of an encryption/decryption request until the result was sent back. The data that was encrypted was a random set of bytes.

The measurement were done using 1-256 bytes with random values even though 256 bytes is a bit more than the WIB ever could send to the wallet server see *2.2.3 Wireless Internet Browser WIB SIM card* or [WAG02] for further details regarding this limitation.

Table 3-5 Measurement of encryption/decryption time at the security center setup time measured separately.

Security Center (all times in ms)					
Action	Average	Max	Min	Median	Std. Deviation
encryption 1 byte	881	901	851	891	20
decryption 1 byte	785	861	721	761	62
encryption 2 byte	647	701	571	671	58
decryption 2 byte	655	661	651	651	5
encryption 64 byte	873	902	831	871	30
decryption 64 byte	871	1261	741	761	222
encryption 128 byte	853	901	801	841	42
decryption 128 byte	801	801	801	801	0
encryption 256 byte	1027	1332	791	911	264
decryption 256 byte	1112	1693	801	901	375
Set up connection	432	1953	310	311	335

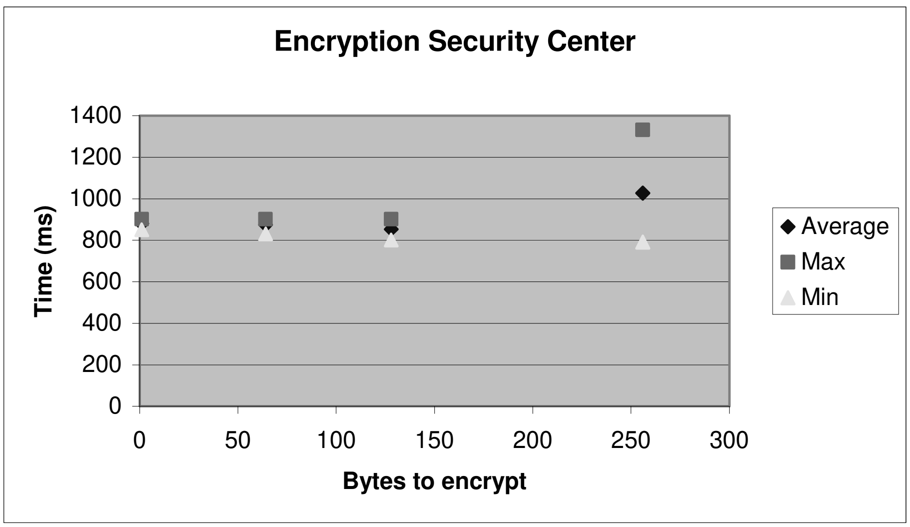

Figure 3-11 Time to encrypt data with the Security Center as a function of the number of bytes.

As can be seen from Table 3-7 an encryption/decryption time of around 1.1s (including connection set up) could be expected regardless of the amount of data to encrypt decrypt.

From the Figure 3-11 it can be seen that the encryption time stays rather constant with the amount of data to encrypt at least when it concerns the small amounts of data that the SIM card is able to handle. Worth noting is that the standard deviation seems to increase with larger amounts of data to encrypt/decrypt.

In the Table 3-5 it can be seen that the connection set up time contributes to the total time for encryption/decryption with almost one third. So it looks like the total effect on using encryption/decryption on the server side should increase the execution time with a bit more than one second regardless of the size to encrypt, since the difference between encrypting/decrypting 1 byte compared to 256 is neglect able

Push measurements

To investigate the effect on the size of a push message to both the WIG and the whole way to the WIB phone a couple of measurements were done. As an upper limit for the message size 600 bytes were used, since the limit that most WIB SIM cards have is that they are able to reassemble 5 SMs each carrying around 120 bytes see *2.2.4 SIM WIB with Java plug-in* or [WAG02].

The largest page pushed in the stock analysis is a WML page of 495 bytes but which during transformation to byte code was reduced to around 185 bytes.

Table 3-6 Total times for push communication i.e. the communication between the WIG and either of content provider or the wallet servers. Number inside parenthesis indicates number of measurements.

Time for communication with WIG during push (times in ms)					
Bytes pushed	Average	Max	Min	Median	Std. Deviation
100 bytes 1:st time (5)	133	141	121	131	8
100 bytes after 1:st time (8)	34	50	20	31	9
200 bytes 1:st time (5)	138	190	100	130	33
200 bytes after 1:st time (8)	29	40	20	30	6
600 bytes 1:st time (5)	116	120	110	120	5
600 bytes after 1:st time (8)	36	71	20	30	15

The amount of bytes mentioned in Table 3-6 is the size of the WML page pushed, since the page is transformed into byte code at the WIG server the actual number of bytes sent to the WIB is reduced. Push times to the WIG and WIB were measured at different occasions therefore the difference in number of measurements.

During measurements it was discovered that the SIM card used couldn't handle the reassembly of more than 3 SMs, or if it was the handset that wasn't able to display so much text at once, since the page pushed only consisted of letters to be displayed.

Measurement of the time to push a WML page the whole way to the WIB phone began at beginning of the push to the WIG server and ended when the page was displayed at the phone.

Table 3-7 Time to push a page the whole way to the WIB phone. Ten measurements were made for each page size. Number of bytes sent were less than page size due to transformation to byte code at the WIG server.

Time to push a WML page the whole way to the phone (times in ms)					
WML size/bytes sent	Average	Max	Min	Median	Std. Deviation
62/18 bytes	7250	8222	6008	7225	773
162/118 bytes	17350	23193	15021	16739	2409
292/250 bytes	24077	26297	20970	24179	1340

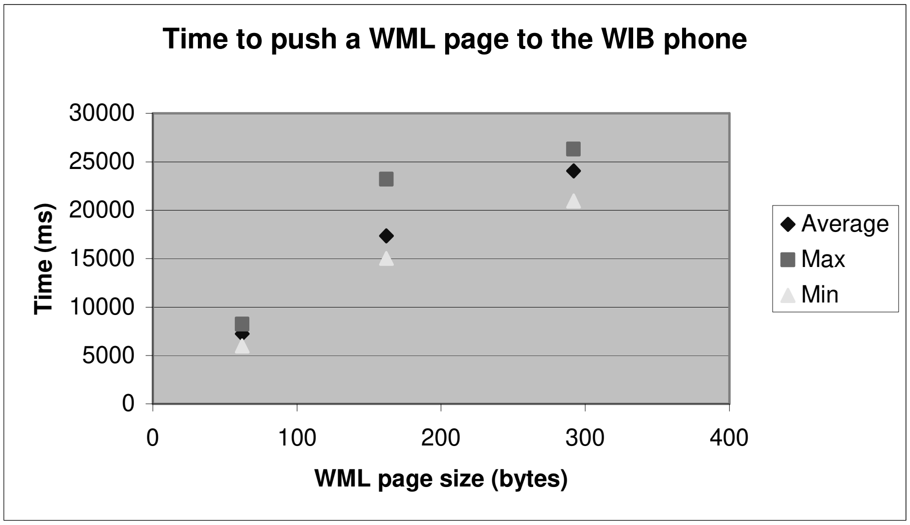

Figure 3-12 Total time to push a WML page the whole way to the phone from a client connecting to the WIG server as a function of the page size.

When sending a push at the web server it was discovered that the first time it took almost four times longer than the following pushes see Table 3-6. A closer inspection of the times showed that the creation of objects at the jsp page went much faster the second time due to some optimizations done either at web server or in the JRE. Since a HttpURLConnection is used it isn't necessarily true that the underlying socket is closed when a disconnect() is performed and thus it may reduce the connection time the second time.

It seems like the actual push of a message to the WIG server never can become a bottleneck in the system, since the push of a message that lies at the upper limit takes about the same time as the others. But as can be seen from Table 3-7 there is a very large increase in waiting time for the end user when increasing the page size form 92 to 292 bytes then it takes approx. 24 seconds for the page to reach the WIB phone instead of 7 seconds.

One thing that is interesting to note is the need to know what kind of WIB SIM that the end users have, since they differ in the number of SMs that they can reassemble.

From Figure 3-12 it can be seen that there seems to be a linear increase in time to push a page the whole way to the phone, but it should be noted that the linearity really isn't linear its only linear in the increase when an other SM is required. If for example two WML pages fitting in one SM but of very different size they still take about the same time but a WML page just slightly bigger requiring 2 SMs nearly doubles the execution time. So the time increase is discrete rather than linear since it goes up "one level" for each new SM required.

The somewhat higher time per SM that arises during the push of a larger page could be explained from that the reassembly takes some time and the preparation taking time to display such a large message on the device's screen.

When comparing the setup time for encryption in Table 3-5 and Table 3-6 it looks to be a rather large difference. It takes 0.3 seconds to setup the connection to the security center compared to 0.1 seconds for the push of a WML page to the WIG including the setup time. The large difference in time could be from poor server coding at the security center compared to the WIG server as well as the possibility of reusing the socket that is used in the HttpURLConnection, it might also has something to do with that the servers reside on different networks or that they use different operating systems leading to TCP/IP stacks with different performance.

Security evaluation

One part that is interesting for this thesis is the communication between the wallet server and the WIB. During the evaluation a SIM card was used that utilized both encryption on the GSM layer as well as the 3DES plug-in available on the SIM card. But one thing to remember is that the 3DES plug-in doesn't provide any message integrity or authentication i.e. there is no mechanism to ensure that the data hasn't been altered and to be sure of who has sent it. Even though the GSM layer might offer some message integrity, authentication and protection against reply attack see Table 2-2, it's not necessarily true that they always are used since they are optional.

With the WIB there is the possibility to achieve end-to-end security that wasn't possible with WAP until the release of WAP 2.0. It was with WAP 2.0 and the transition from WTLS (Wireless Transport Layer Security) to TLS (Transport Layer Security) that made end-to-end security possible in the WAP protocol [IBM02]. Before WAP 2.0 the data were in plain text for a short while in the WAP gateway during the translation between the protocols used in the fixed and wireless "worlds". Having plain text in the WAP gateway made it an ideal target for "hackers" or those who wanted to express the insecurity with WAP.

As so many times the hardest part is to distribute the keys in a secure manner, if that has been done it is possible achieve a very high degree of security with the WIB. Not just encryption, but also message integrity and authentication.

3.2.5 Evaluation Summary

When it comes to the comparison between using 3DES and plain text it is quite clear that there is an increase in time experienced by the end user when using 3DES. The increase is around 16 % or 6 seconds for the 3DES case and that could be justified by the higher degree of security. The increase comes mainly from that the reply with the encrypted password seems to take almost 6 seconds longer than sending the password in plain text. This increase in time comes from the sending of an additional SM.

The individual measurements of some individual factors such as size showed that:

Encryption time at the SIM card increased almost linearly with the amount of data to encrypt, but that increase in time is neglectable compared to the total execution time, at least since there is an upper limit on how much data the SIM card is possible to handle.

When it came to encryption/decryption on the server side the total time stayed almost constant with just a very small increase in time when data size was increased numerous of times. As much as one third of the total encryption/decryption time came from setting up the connection to the security center.

Push measurements showed as easily could be expected that the time to push a WML page to the WIG was almost constant regards less of size this since a fast Internet connection was used. On the other hand when it came to the time it took to push a WML page the whole way to the WIB phone it was a large increase in time for each additional SM that was needed. For example if first a page where pushed that only required one SM and then a page that required two SMs the time was almost doubled. If an even larger page were to be pushed so three SMs were needed there would be an other 50% increase in time and so forth, i.e. the time taken to push a message the whole way to phone could be approximated to about 7-8 seconds times the number of SMs required.

The assumption made before the evaluation was started that the biggest contributor in the 3DES case was going to be the communication with the security center was correct in the sense it was the largest relative contributor, but when it came to the largest contributor measured in real time the sending of the encrypted message from the WIB surprisingly turned up, due to the padding performed by the 3DES plug-in.

Possible Improvements

As can be seen the part that effects execution time the most is the communication done with SMs, even though the communication with the WIG server and security center takes some time it is very small compared to the time taken to get a SM delivered. So it's very important to try to reduce the amount of data since using several SMs for the delivery dramatically increases execution time. It would be desirable to see if it's possible to use some form of compression on the data sent.

One improvement that could be done would be to keep connections to the WIG server, security center and between the content provider and wallet server always open. One drawback is that this would increase load on those servers. On the other hand it would decrease the setup time with only a slight impact on the complexity. Reducing the connection set up time could be neglected compared to the delivery time of a SM and thus it might not be justified to keep the connections open.

Execution time at the web server could be improved since today each update of the database involves the writing of a file.

Since there is no message integrity except the one offered by the OTA layer, see Table 2-2, this solution is only suitable for the payment of "small" amounts. An extension using message integrity and authentication so all parts are sure whom they are communicating with would have impact on the amount of data sent and may thus increase the total execution time. Increasing the execution time even further would be very frustrating for the user since he/she would experience an even larger waiting time. The most frustrating part is that there is no sort of progress bar so the user has no idea how much longer time it will take.

Future Work

No measurements with a high number of clients were made due to a couple of reasons it would require a real large number of clients to effect the WEB server at which the service is running. Not possible by using a number of WIB phones since they wouldn't generate enough traffic to affect the web server. One possibility would be to generate a number of requests from a desktop computer that emulates a number of WIB phones, but the focus on the thesis doesn't lie in writing a high performance server even though the performance of the server is important in a real life application. The performance that could be gained by optimizing the servers is very small compared to the total execution time.

One thing to investigate further is if there is any performance to gain in trying to do compression of the data sent with SMs, since it's in the communication with SMs that takes the most time. A possibility to reduce that time by a couple of percent would have a much larger impact than improving the server performance or the communication between the different servers.

First impressions

The WIB solution tends to feel "locked in" since the operator, the end user and the content provider needs to use this special technology with a WIB, WIG and a subset of WML together with the extension for plug-in calls. If instead the content provider could use an "open" technology like WAP it would reach all end users equipped with WAP phones, which is a lot more than those with WIB phones. One additional drawback with the WIB is that the content provider must have a contract with a mobile operator, which in turn offers the services to his/hers users. This due to the high degree of security required when it comes to changing files on the SIM card and this is only possible for an operator that "administrates" the SIM to do.

To develop WIB applications feels easy and rather fast even though the developer doesn't have any previous knowledge of web programming. Despite that fact it might be a lot harder to write a real world server cable of handling a couple of thousands simultaneous users, but the basics needed are easy to acquire at least for a user with any previous knowledge of say HTML/Java/jsp/Asp. One thing that can raise some problems are the SIM cards since they sometimes are a bit hard to please.

Only small applications that aren't interactive are suitable e.g. the payment of a bill. One possible scenario is a user who is afraid of giving away his/hers credit card number on the Internet instead the e-commerce site has a contract with a mobile operator providing a pay service to which the e-commerce site issues a bill. The pay service then pushes the bill to the mobile phone user and the user signs the bill. Once money has been withdrawn from the bank it informs the e-commerce site that it can send the goods to the user without having him/her giving away their credit card number on the Internet.

For interactive services e.g. games there is too much latency involved at least for users who are custom with low latency e.g. a Counter Strike gamer (one of the largest first person shooter games) that is custom to play his/hers favorite game over a fast Internet connection. Much more suitable for small services as e.g. weather, horoscope and other services that aren't interactive and at the same time only requires small amounts of data to be sent.

Only suitable where small amount of data are sent/received due to response times e.g. if amounts like 0.6 kilo bytes (5 SMs at each approx. 120 bytes) were to be transferred the expected waiting time could be as much as 25s [WAG02].

Finally there seemed to be quite a lot of errors in the internal documentation regarding the WIB, which probably wouldn't be the case if a large open standard like WAP was used instead.

3.2.6 Development: problems, environment & time

Problems

During the implementation a couple of annoying errors came mostly due to the lack of documentation revision e.g. it was said in the WIG/WML specification how an encryption plug-in was to be called. But the format given there was incorrect one parameter was given as 'encr' but it was supposed to be 'ENCR'. This was found by the old trial and error method. An other miss was in the guidelines for updating SIM cards where it in the guideline was given that the hex value of 8F was used to identify the beginning of a new menu item, but by inspecting the old menu file it was discovered that the supposed hex value was 0F. One other annoying thing was that their WAC (WIG Application Creator), which is a WIB phone emulator that Smarttrust have developed was a lot easier to please than a real world phone. Code that worked on the emulator didn't always work on a real WIB SIM.

For the first SIM card that was used the 3DES keys were not known and at the time an other SIM card was found it didn't work as expected it received the SMs but didn't show anything. Got some help and after a couple of hours work on the behalf of Gunilla she got it to work with one WIG server but not the one previously used. If was probably due to some security issues on 03.48 GSM layer.

If it weren't to factors out of my control e.g. getting hold of SIM card, phone, specifications and guidelines being incorrect and finally the need of help from others to set up some servers, the work probably would have been done in a much sorter amount of time than it now was done.

Development time and environment

The development environment used consisted of the WAC (WIB phone emulator), tomcat web server and Emacs for the coding while ant (Another Neat Tool) was used for the make support. Emacs was used since Jbuilder felt a bit slow on the computer on which the development was done. An IDE is nicest to use if the development consists of some GUI programming otherwise an editor like Emacs is just as good at least if there is a decent .emacs and make support available. It must be mentioned that ant really is a neat tool. It has all the common features as the ordinary make in the Unix world but not the limitations and peculiarities e.g. an XML file is describing the build so there is no issues with spaces and tabs as in a Unix Makefile. Worth mentioning is that the latest versions of Jbuilder has integrated Ant for their build support.

To develop services and applications for the WIB goes fast even though security is needed since there isn't much more job that has to be done to get end-to-end security including message authentication and integrity. The time to develop a simple service can take from a couple of days up to a couple of weeks depending on the prerequisites and the complexity on the server side.

3.3 Java Card evaluation

3.3.1 Application Design

The main idea with the design is to have some form of pay service (wireless wallet) available from the cellular phone and in contrary to the WIB case trying to do as much as possible locally on the phone. Doing as much as possible locally reduces communication with the bank and thus reducing the total execution time.

The actual account resides in the phone itself as a STK (SIM Toolkit) applet that has been installed to a Java enabled SIM card. The responsibilities of the wireless wallet applet is to listen for incoming bill requests and then ask the user to present his/her PIN code if he/she wishes to pay that bill. Once a bill request has been handled the bill issuer is informed whether the bill was paid or not.

As can be seen in Figure 3-13 the wireless wallet implements a shareable interface containing a public method bill. Applets wishing to send a bill to the wireless wallet call the bill method of the WalletSIO. For more information regarding SIOs see *2.2.2 Java enabled smart cards* or [CHE00].

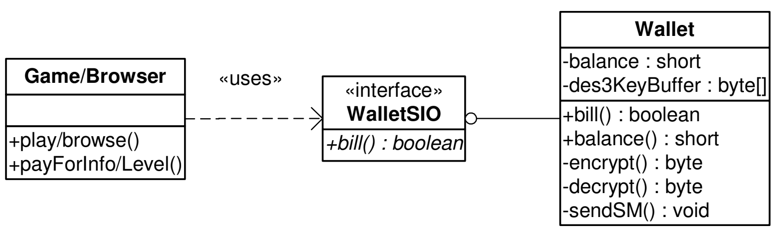

Figure 3-13 The most central methods and attributes of the Class diagram of the wireless wallet and the applet bill issuer.

The solution chosen isn't just limited for initiation by another applet you could easily think of other scenarios where the initiation is done through some other means e.g. a browser on a desktop computer or a candy machine where a SM is sent to the wallet. A bill sent as a SM need to be correct formatted and sent to the TAR value of the wireless wallet applet otherwise no payment will be possible and the bill will be rejected, for more details regarding TAR values see *2.2.2 Java enabled smart cards*.

Once a bill has been paid regardless of initiation way an SM containing the amount withdrawn and the id of the bill issuer is sent back to the "bank". This information is sent to the bank so the money actually can be transferred from the user to the bill issuer.

The process described in Figure 3-14 is where an applet e.g. a browser or a game issues a bill by calling the bill method, which is available through the shareable interface. Then the wallet applet prompts information about the bill to pay i.e. amount and a short description what to pay for. It is then up to the user to decide whether to sign the bill or not by typing his/her PIN code.

If the user has presented the correct PIN an encrypted SM using 3DES is sent to the bank to inform it of the transaction so the actual money transfer can take place. Then the method returns true to the bill issuer if payment was done and else it returns false. Both the triple DES key and user PIN is decided at install time of the applet, but the user may change the user PIN if desired.

Once the method call has returned to the bill issuer it takes the appropriate action e.g. granting/denying access to the next level once the bill payment has been verified.

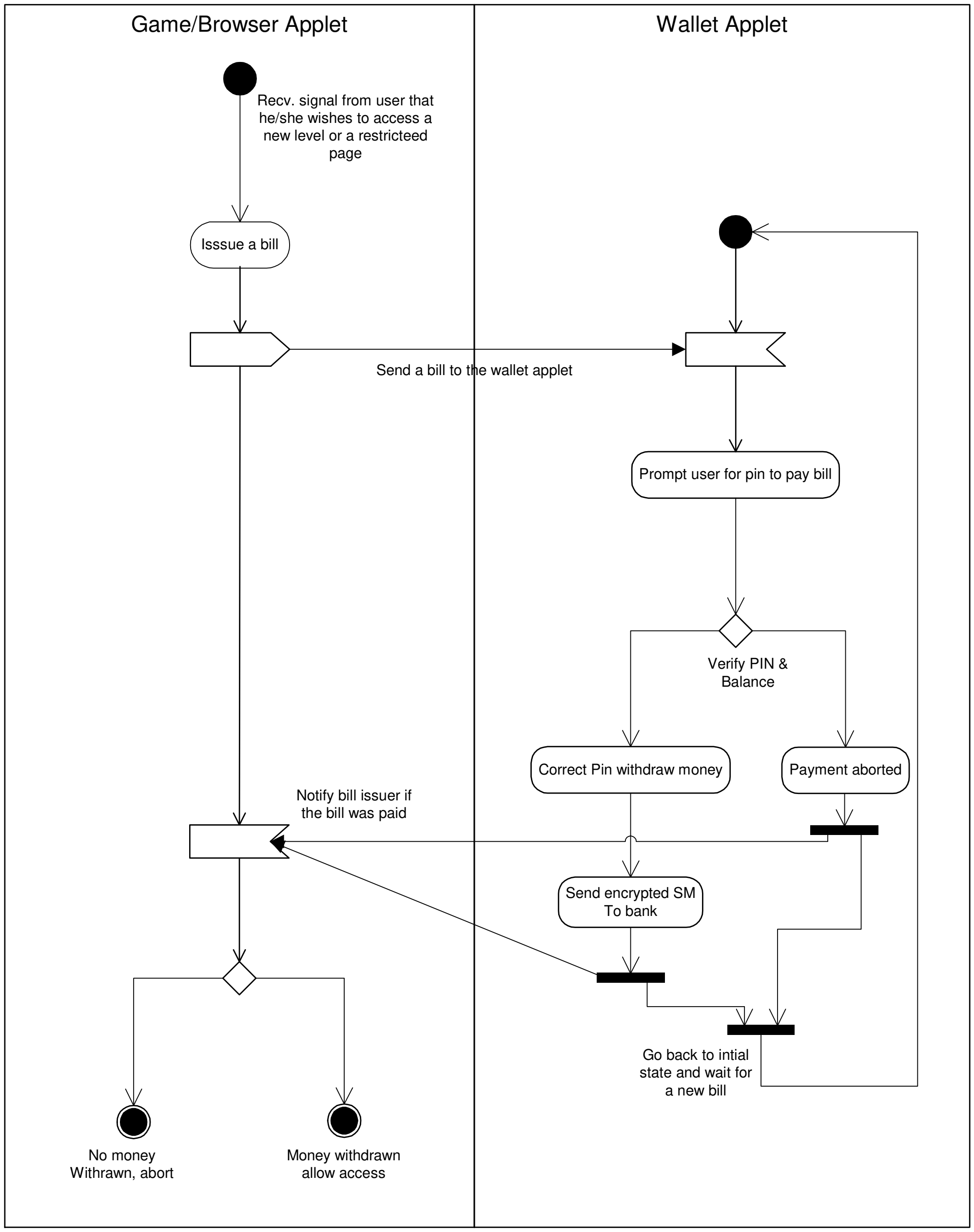

Figure 3-14 Program flow when issuing a bill to the wireless wallet applet.

3.3.2 Security Design

There are both a plain text version and one version, which encrypt all the communication between the wallet and the bank.

Encryption used between the WIB and the Wallet server is 3DES that is a symmetric crypto algorithm i.e. the same secret keys is used both to encrypt and decrypt. For more details regarding 3DES look at e.g. *3.1.4 Security* or [STA00].

The 3DES key used for the communication between the wallet and the bank is decided at installaton time as a parameter to the install method.

Once a bill has been paid an encrypted SM containing the amount withdrawn and the id of the bill issuer is sent back to the "bank" from the wireless wallet applet. A copy of the users account should be hold at the bank to verify that the user doesn't tamper his/her card i.e. make sure that the balance never goes below zero.

Encryption and decryption on the server side is done by using a freeware implementation of the Java crypto API from www.bouncycastle.org. At the server side a database is used which matches a MSISDN to a 3DES key so all messages can be decrypted regardless of sender. Since the encryption/decryption is done in both ends of the connection it's possible to have a secure end-to-end connection without plain text in any intermediate place.

3.3.3 Implementation and limitations

In a real world implementation, money really has to flow to the content provider e.g. the wallet server constantly transfers money for each transaction to the content provider's account or once in a while a bigger amount. This has not been implemented in any way. But all bills that have been paid are stored in a database so it could later be possible to do an extension where the money really flows to the right account.

Due to the fact that a Java enabled SIM card without subscription was used, both the "content server" and "bank" were run at the same machine in a simulated environment. That the servers were run in a simulated environment means that they must be edited to some degree to be able to function in a live environment.

3.3.4 Evaluation

A Java enabled SIM card without a subscription was used so the SIM was connected to the simulation environment using a card reader inside the simulation environment there was a simulated phone using the real Java Card as well as a SMS-C and the other servers needed. Due to the fact that the Java enabled SIM card would use GSM to send the SMs if there was a subscription the SMs should take about the same time as in the WIB case. The only difference that might arise between the Java enabled SIM card and the case with the WIB are the number of SMs sent, server side execution and the encryption time at the SIM card.

Browser/Game example used for the case study and evaluation

For the evaluation of the wireless wallet an example program in the form of a stock analysis program was developed. The stock analysis program could also work for the analogy of paying for information e.g. gaining access to a new level in a game or paying to view a restricted page.

In the prototype application the wallet is stored as an applet and then other applets can send bills by calling a method available as a SIO see Figure 3-13. Since both the wallet and stock analysis applications were installed as SIM Toolkit applets they appeared as menu items on the phone.

Once a user chooses the stock analysis program a menu appears asking what the user wishes to do. If the user e.g. wishes an analysis of Ericsson a bill request is sent to the wallet applet from the stock analysis applet. For the process how a bill request is handled see Figure 3-14. As control is returned to the bill issuer i.e. the stock analysis program an SM is sent to the content provider with the request of the analysis and the content provider sends back the requested analysis.

The communication between the bank and the wallet applet is done using 3DES for the data part of the SMs. Both the content provider and bank essentially are built up of a couple of Java Classes and a database for the paid and unpaid bills.

To insert money on the wallet a SM with the TAR value of the wallet applet is sent to the phone, then the wallet applet checks the SM to see that it is valid i.e. contains specific information such as whether to insert or withdraw money and the amount. This meassage is also sent using 3DES. But these features are not evaluated in performance issues, since they can't be compared to the WIB case in a good manner.

Testbed

The testbed used both for the performance and security evaluation were made with the stock analysis program as a base. The most important parts of the testbed are described in Figure 3-15 in some places it was hard to do measurements e.g. there was no possibility to take timestamps "inside" the code that was executed on the Java Card in contrary to the part of the application that was run on the server side. Since a Java Card without subscription were the same measurements for sending SMs as those from the WIB case were used, for a more detailed motivation see *3DES/Plain text execution time*.

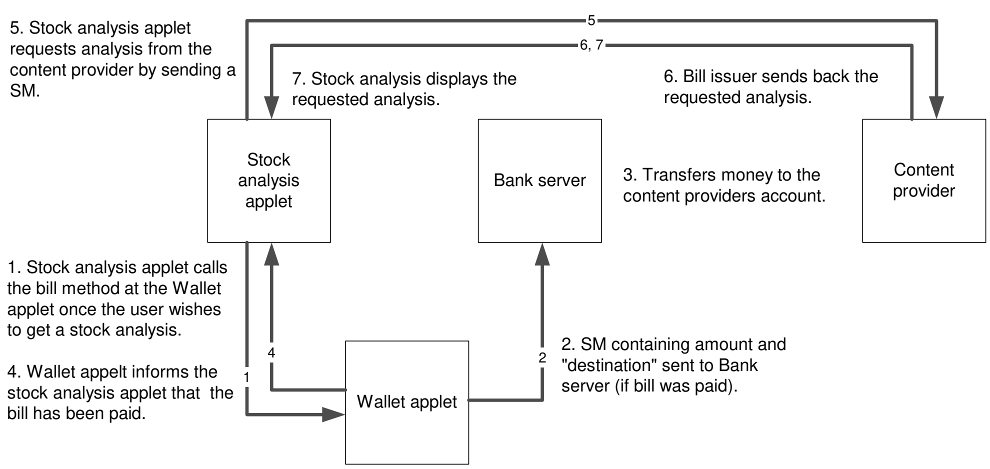

Figure 3-15 The test bed used for the evaluation.

The total time was measured and divided into different parts see Figure 3-16 for the parts that were measured individually. Times measured individually were: user interaction time to start the application, user interaction time to type password and execution time at the content provider as well as the bank. The time to send SMs were taken from the WIB case due to the use of a Java Card without a subscription.

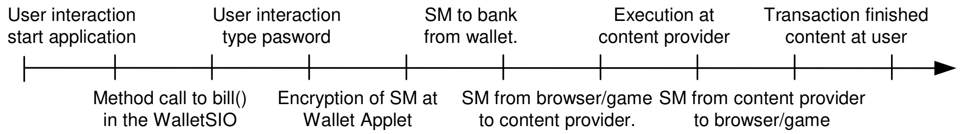

Figure 3-16 The test bed used for evaluation. Actions numbered x-1 or x-2 may take place in parallel

First the total time were measured and divided into its different parts using both 3DES and plain text for the communication between the wallet applet and the bank.

Then a detail study of how the different parts would effect execution time were done, e.g. to see how size would effect encryption/decryption time both at the SIM card and server side. There is also a small investigation in the difference in using transient arrays (array stored in RAM) compared to using non transient arrays arrays stored in EEPROM this since read writes in RAM are much faster than the counterpart in EEPROM.

In the following paragraphs the evaluation results are handled in detail and a summary is available in the section called Evaluation Summary

Estimation of time distribution in wireless wallet case study

Before the evaluation and measurements were done an estimation of the time spent in the different parts during the execution of the stock analysis program were done as showed in Figure 3-17. Dashed areas should be much smaller and it can clearly be seen that the most time will be spent by sending SMs and engaging in user interaction as well as in the encryption/decryption of data. Since the time spent in doing encryption/decryption is rather small there shouldn't be such large difference between using plain text and 3DES in the communication between the wallet and the Java Card.

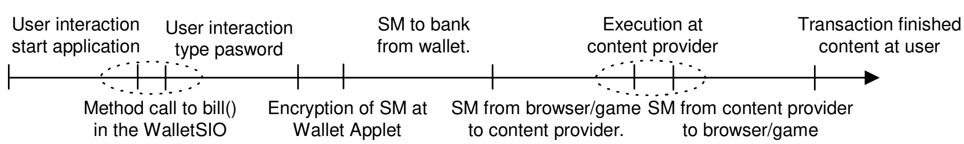

Figure 3-17 Estimation of proportions where time is spent during a transaction, dashed areas are "zoomed" in i.e. they should be much smaller compared to the other.

3DES/Plain text execution time

Measurements both for the 3DES and the plain text case were done using both logging of times that were of interest from inside the code and timing done manually by a human, since there is no way to use timestamps on the phone.

For both the 3DES and plain text case there were 11 measurements done i.e. a total of 22 measurements. More measurements could be made to further ensure that the values lies around the values presented in the tables, but it's hard to automate the testing, since there is no easy way to automate the testing due to the fact that some measurements need to be done manually.

Due to that a SIM card without a subscription were used the time to send a single SM as well as the time taken to push a SM to the SMS-C from the server side were taken from WIB case. These values are in italic in Table 3-8, Table 3-9 and Table 3-10. It's OK to use the same values since the parameter having the most impact on the push of a SM to SMS-C is the traffic in the network and the latencies involved, so they should be around the same since the same network is used and the size to push is about the same. Even though there might be a couple of bytes (around 10 more to push) the impact is neglible since a network with the speed of 100 Mbs was used. The same thing applies to the sending of a single SM since if a SIM card with a subscription were used the same SMS-C as in the WIB case would be used and the parameters impacting time is primarily the workload of the SMS-C and the traffic in the network closest to the subscriber.

It could be argued that it is even more correct to do as described above i.e. to use the same values for the sending of the SM and the push of the SM to the SMS-C, since then the network will not have impact on the final time measured. Otherwise it would be imperative to do the measurements simultaneously or such things as behavioral pattern would effect the measurements e.g. if the WIB case measurements were done at peak time and the others not.

So the difference should not come from the network instead it should come from the number of SMs sent as well as the execution at the client (SIM card) and server side.

The time to send a SM in both the 3DES and plain text versions should lie around the same time, since the same number of SMs are sent the Java Card and the bank and content provider.

Table 3-8 Measurement of times in stock analysis program using encryption between wallet applet and the bank server. Italic values are taken from the WIB case.

3DES (All times in ms)					
Action	Average	Max	Min	Median	Std. Deviation
User interaction start app.	5407	6519	4276	5287	883
User interaction type in password	2368	3595	1662	2413	530
Preprocessing and Wallet encrypt SM	1629	2393	1371	1512	320
SM from Wallet to bank	*7955*	*11918*	*6350*	*7551*	*1689*
Whole processing at the Bank	511	842	320	461	162
SM from wallet to content provider	*6728*	*8151*	*5668*	*6679*	*599*
Processing at the content provider	1.42	2.60	1.00	1.20	0.56
Push SM to SMS-C	*35*	*80*	*10*	*31*	*18*
SM from content provider to browser/game	*8513*	*9984*	*6460*	*8562*	*1074*
Total time for transaction	33146	35973	29895	33221	1853
Total time for transaction minus user interaction	25372	27311	23957	25106	996

As can be seen from Table 3-8 and Table 3-9 is that both the plain text and 3DES case have a high standard deviation for the user interaction and the sending of the SMs. That the user interaction has a high standard deviation isn't especially surprising since the time it takes for a user to press the buttons needed may vary rather much. The sending of the SM having a high standard deviation may have many reasons e.g. the workload of the SMS-C (the server responsible for the handling of the SM), traffic in the network, reception quality (retransmissions), the one thing that have the largest impact is the traffic level in the network closest to the phone.

Table 3-9 Measurement of times in stock analysis program using plain text wallet applet and the bank server. Italic values are taken from the WIB case.

Plain text (all times in ms)					
Action	Average	Max	Min	Median	Std. Deviation
User interaction start app.	5062	6108	3425	4987	744
User interaction type in password	2213	2784	1653	2353	406
Preprocessing SM	1233	1472	1132	1232	91
SM from Wallet to bank	*7955*	*11918*	*6350*	*7551*	*1689*
Whole processing at the Bank	414	731	231	331	170
SM from wallet to content provider	*6728*	*8151*	*5668*	*6679*	*599*
Processing at the content provider	1.22	2.51	0.80	1.10	0.46
Push SM to SMS-C	*35*	*80*	*10*	*31*	*18*
SM from content provider to browser/game	*8513*	*9984*	*6460*	*8562*	*1074*
Total time for transaction	32155	35712	29945	31965	1628
Total time for transaction minus user interaction	24880	27239	23807	24415	1057

From both Figure 3-18 and Figure 3-19 it can clearly be seen that largest contributors to the total time are user interaction and the time spent in sending SMs i.e. the communication.

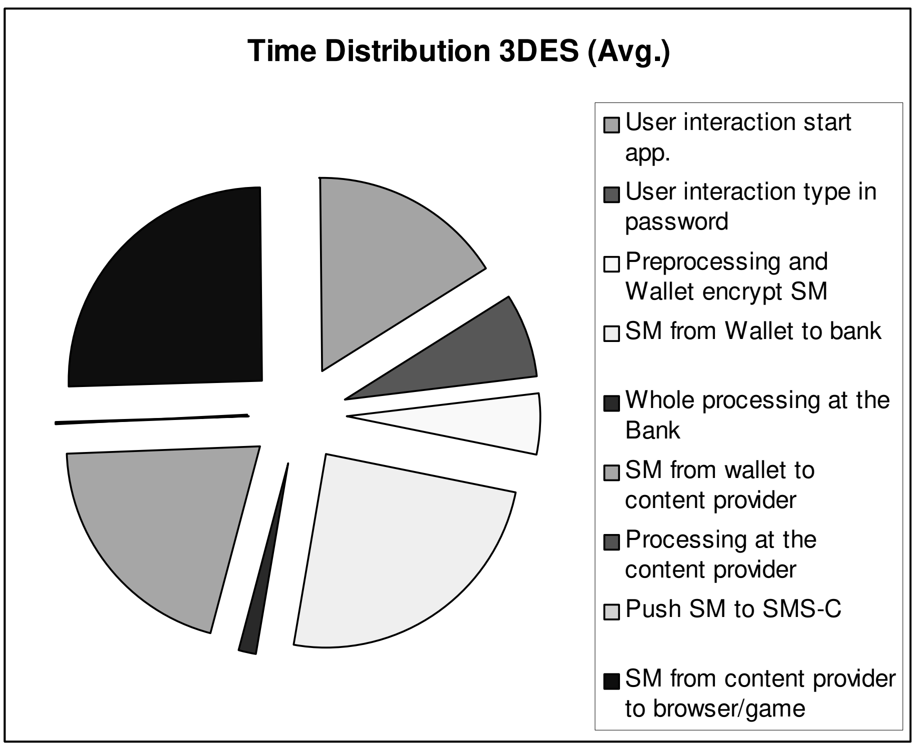

Figure 3-18 Time Distribution during a session using 3DES for the communication.

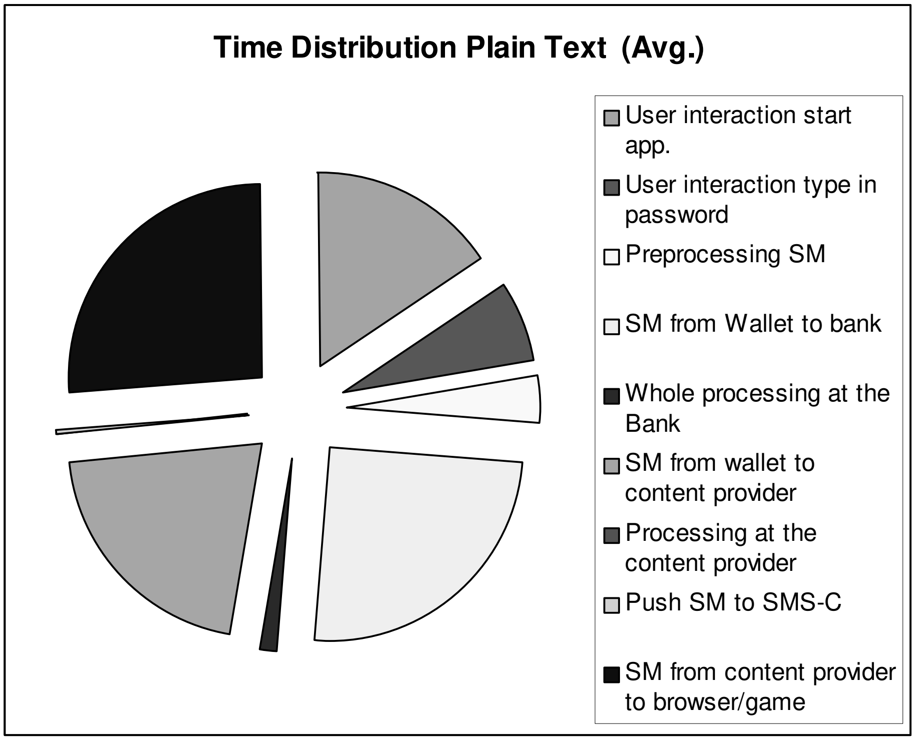

Figure 3-19 Time Distribution during a session using plain text for the communication.

In Table 3-10 there is a comparison between the plain text and 3DES case, it was performed by normalizing plain text to 3DES i.e. taking the quotient 3DES/(plain text) for the median and average values. In the normalized case the plain text is faster if it's a value above 100 otherwise it gets a value below 100. Because not just the relative difference between them are important the same thing was done by taking the following subtraction 3DES-(plain text) for the average and median values, i.e. a positive value means that the 3DES case takes that much more time and a negative value the opposite that the 3DES case is that many ms faster.

Table 3-10 Comparison between 3DES and plain text execution time where 100 is the 3DES A positive value means that the 3DES is faster than the plain text version and a negative value the opposite. Italic values are used from the WIB case and since no Java Card with a subscription was available.

Comparison of times in 3DES &PLAIN TEXT cases				
Action	Normalised 3DES/plain text		Real difference (ms)	
	Average	Median	Average	Median
User interaction start app.	107	106	344	300
User interaction type in password	107	103	155	60
Preprocessing SM (inc encrypt 3DES) at SIM	132	123	395	280
SM from Wallet to bank	*100*	*100*	*0*	*0*
Processing at the Bank (inc. decryption 3DES)	123	139	97	130
SM from wallet to content provider	*100*	*100*	*0*	*0*
Processing at the content provider	116	109	0.20	0.10
Push SM to SMS-C	*100*	*100*	*0*	*0*
SM from content provider to browser/game	*100*	*100*	*0*	*0*
Total time for transaction	103	104	991	1256
Total time for transaction minus user interaction	102	103	492	691

The time that is the most interesting for the user is the total time for the transaction with the user interaction time removed, since the pressing of buttons isn't experienced as a waiting time. It's also good to remove it since it has a high standard deviation. When removing the user interaction time only the technical aspects that effects the execution time are compared and not the time a user spends on typing e.g. his/hers password.

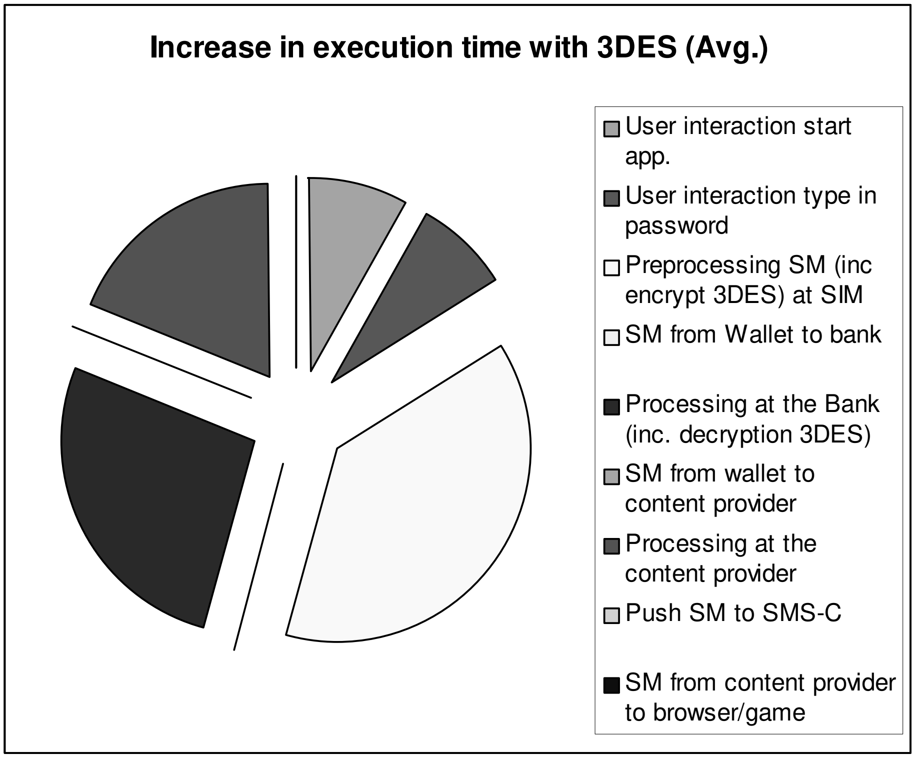

Figure 3-20. Increase in execution time with 3DES i.e. which part has the largest increase in execution time compared to when plain text is used. Values have been taken from Table 3-3.

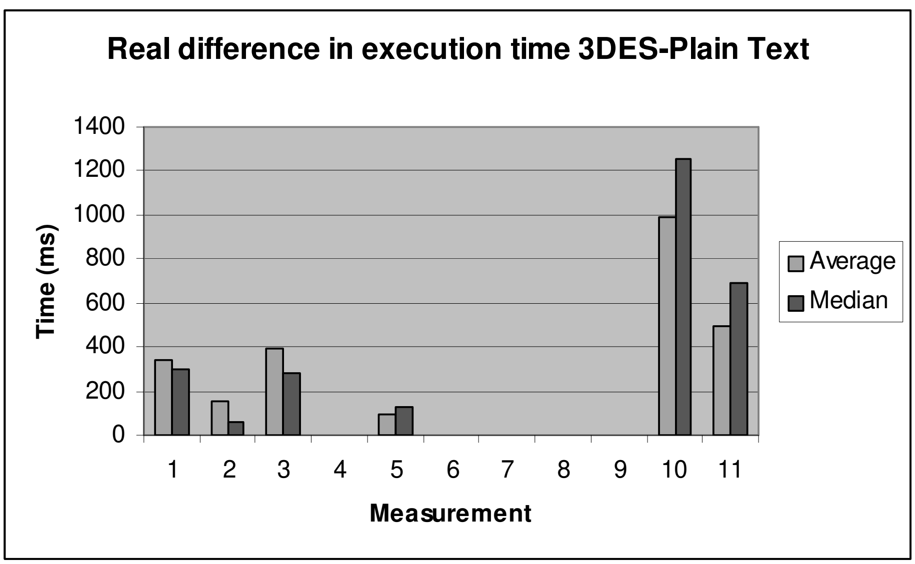

Figure 3-21 The real difference in execution time between the 3DES and plain text case. See Table 3-10 for which measurement the different numbers corresponds to. A positive value means that the 3DES case takes that many more ms and a negative value the opposite.

The largest relative difference which can be seen both in Table 3-10 and in Figure 3-20 comes from the encryption/decryption both at the SIM card as well as at the bank. This increase is very natural due to the extra worked required for the encryption/decryption even though the increase in time is small with an increase around 100 ms for the bank and around 400 ms for the SIM card encryption The largest contibutor i.e. the SIM card encryption is expected due to the lack of processing power there.

According to Table 3-10 there is a difference in computation time at the content provider at around 10-16% or 0.1 ms in real time between the 3DES and plain text case, but this difference is neglible in size. It probably comes from context switches, since a context switch might have a rather high impact on the measurement of such small values. There shouldn't be a difference since the same code is executed at the content provider regardless whether 3DES is used or not. The only secure communication used in the case study is between the wallet applet and the bank.

But an addition in time might occur if there is a need to also use a secure communication between the content provider and the applet issuing the bill. In this case there would be a small increase in computation at both at the content provider and SIM card

As expected the only difference in execution time comes from the parts involved in encryption and decryption see Figure 3-21 and Table 3-10 for the increase in real execution time as 3DES is used instead of plain text.

The difference in user interaction time between the 3DES and plain text versions in Table 3-10 is pure random since the same interaction by the user is required in both cases.

An increase of approx 2-3 % (0.5 s) of the total execution time should be expected if 3DES is used in the communication between the wallet and the bank, this a very small increase, but at the same time providing a much higher degree of security.

From Figure 3-22 it can be seen that execution time differs rather much, this due to the high standard deviation for the sending of SMs.

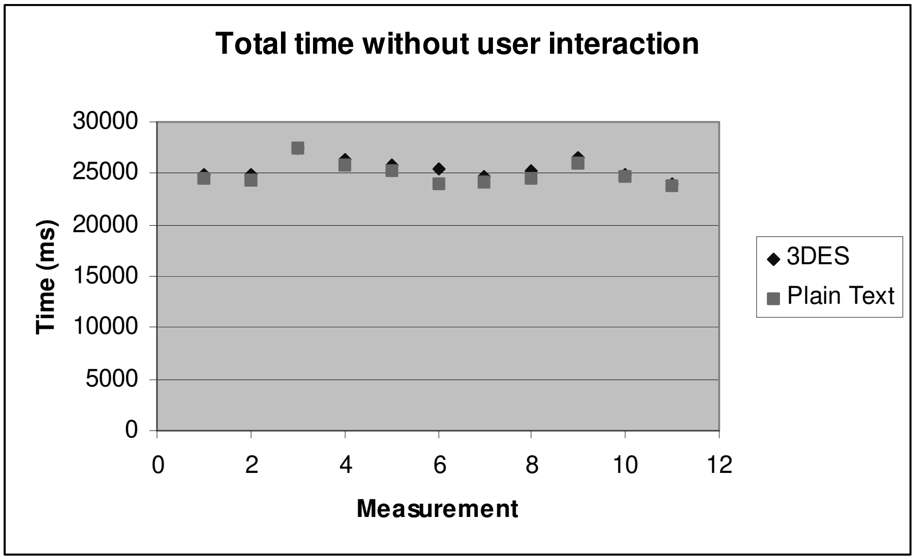

Figure 3-22 Total execution time in the stock analysis example with user interaction time removed.

SIM card encryption/decryption time

The first measurement that was performed was to try to find out the encryption time when doing encryption on the Java enabled SIM card. It was done by first encrypt some data and then measure the time until the encrypted data could be displayed on the device.

To be able to compare the measurements with the WIB SIM card the measurements were done in the same manner, both the number of encryptions as well as the amount to encrypt, i.e. 8 consecutive encryptions were done and then the total time minus the reaction time were divided by 8 to calculate the time for a single encryption.

The amount used as an upper limit was 150 bytes since a SM can have a payload of around 120 bytes depending on the features used at the GSM layer.

There is one small distinction between the measurements in the Java Card case and the WIB case since in the Java Card case encryption time was measured both for encrypting an ordinary byte array and the time to encrypt a transient byte array. The difference between a transient and "ordinary" byte array is that the transient is stored in RAM (fast) while the "ordinary" is stored in EEPROM (slow) memory.

Table 3-11. Encryption time on Java enabled SIM card.

SIM Card Encryption					
Amount data to enc	Average	Max	Min	Median	Std. Deviation
10 bytes (Tranisent Array)	249	257	239	250	9
64 bytes (Tranisent Array)	453	456	451	453	2
128 bytes (Tranisent Array)	864	870	860	863	4
10 bytes	307	309	306	307	2
64 bytes	675	676	671	676	3
128 bytes	1343	1356	1326	1346	13

As can be seen both in Figure 3-23 and Table 3-11 the time seems to increase rather linearly with more data to encrypt. Notable is that the non-transient array encryption got a steeper curve. Encryption time will still be very small compared to time spent in communication. Due to the upper limit on how much data a SM can have as payload the upper limit for encryption time will also be small, around 1 s depending on if a transient array is used or not. Clear is that the encryption will effect the final execution more than in the WIB case when the same amount of bytes are encrypted.

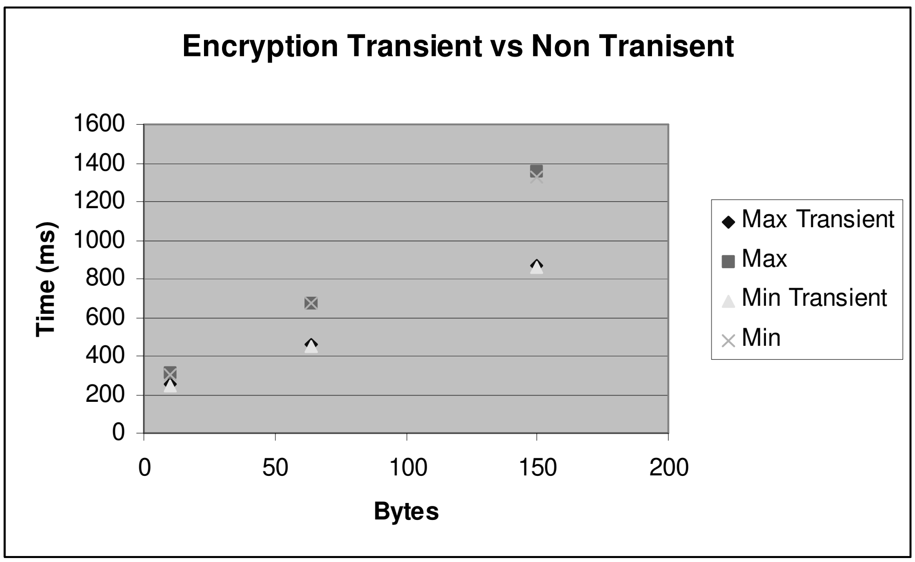

Figure 3-23 Time to encrypt data on the SIM card as a function of the number of bytes, using both a transient array as well as an ordinary.

The difference in speed between the EEPROM and RAM memory is around 1,000 times, so it would be possible to save around tens maybe hundreds of milliseconds to use a transient array [JGU02], but it's only possible to use a transient array in the case if it doesn't matter if data are lost or not in the case of a card reset. So in the wallet case it may not be the best idea to use a transient array for the data in the SM that should be sent, since it must be transactioned/logged i.e. if money has been withdrawn you are sure that the one issuing the bill is informed of the payment and if a SM is sent it needs to be logged so it's not sent twice.

Server side encryption/decryption

Encryption and decryption on the server side was done by using a freeware implementation of the Java crypto API from www.bouncycastle.org. The data that was encrypted was a random set up bytes.

The measurement were done using 1-256 bytes even though 256 bytes is a bit more than the wallet applet ever could send to the bank server, due to the restrictions that one SM usually can have a payload of about 120 bytes. To increase the amount of data sent one possibility would be to send a couple of consecutive SMs.

To be able to measure the times shown in Table 3-12 for a single encryption/decryption a 1000 encryptions/decryption were made of the same size, then the resulting time was divided by 1000. This measurement was also done eight times to see if there would be any big changes. No measurements regarding the standard deviation are shown since they wouldn't show the standard deviation between a single encryption/decryption but instead between the different measurements of a 1000 encryption/decryption that were made.

Table 3-12 Time taken for encryption/decryption on the server side i.e. the bank and contentprovider.

Server side encryption/decryption i.e. the bank (times in ms)				
Bytes enc/enc	Average	Max	Min	Median
encryption 1 byte	0.336	0.340	0.331	0.340
decryption 1 byte	0.327	0.331	0.321	0.330
encryption 2 bytes	0.308	0.311	0.300	0.310
decryption 2 bytes	0.305	0.311	0.300	0.301
encryption 64 bytes	0.375	0.381	0.361	0.380
decryption 64 bytes	0.381	0.391	0.371	0.380
encryption 128 bytes	0.455	0.461	0.450	0.451
decryption 128 bytes	0.467	0.471	0.461	0.470
encryption 256 bytes	0.625	0.651	0.611	0.621
decryption 256 bytes	0.627	0.641	0.601	0.631

From Table 3-12 an encryption/decryption time of around 0.3-0.6 ms could be expected depending on the amount of data to encrypt/decrypt.

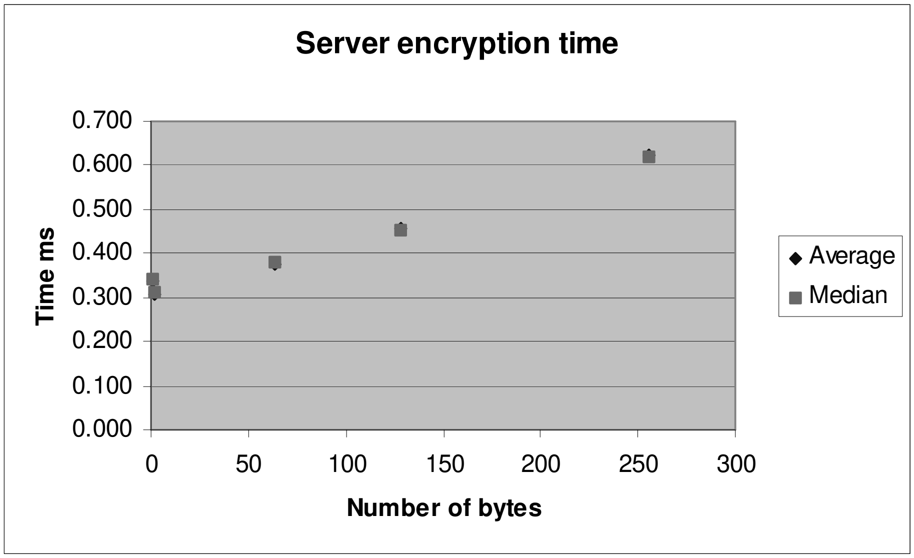

Figure 3-24 Server encryption time i.e. bank and or content provider.

From Figure 3-24 it can be seen that the encryption time stays rather linear with the amount of data to encrypt at least when it concerns the small amounts of data that the SIM card is able to handle. Worth noting is that with the sizes that are concerned there will be no problem to handle a lot of encryption/decryptions simultaneously since the time to perform one single encryption/decryption is rather negliable.

So the total effect on using encryption/decryption on the server side looks like it shouldn't increase the execution time notable since the time taken for a single encryption lies way below one ms regardless of the size.

Security evaluation

The most important part concerning security is how the applet is installed since it is then it gets the 3DES key, which it uses for the communication with the bank. Installing an STK applet can be done in primarily 3 ways, OTA, during manufacturing and using a card reader. When using OTA it's very important to use encryption on the GSM layer to reduce the risk of anyone decrypting the SMs used to download the applet and getting hold of the 3DES key. The applet could also be installed during manufacturing then it's imperative that the manufacturer can be trusted. An applet could also be installed using a card reader in this case it's imperative where the card reader is and how to distribute the key to that place e.g. a bank office or point of sales.

One other possible solution is to not decide the 3DES key at install time since it's just a byte array and not decided in hardware it is possible to change it and decide the key after installing the applet. Here a couple of different scenarios could be thought of e.g. letting the user choose a key. One drawback with choosing the 3DES key after installing the applet is that it is error prone, since the user has to type in a 24 bytes long key.

One thing to remember is that the 3DES encryption doesn't provide any message integrity or authentication i.e. there is no mechanism to ensure that the data hasn't been altered and to be sure of who has sent it. Even though the GSM layer might offer some message integrity, authentication and protection against reply attack see Table 2-2, it's not necessarily true that they always are used since they are optional.

As so many times the hardest part is to distribute the keys in a secure manner, if that has been done it is possible to achieve a very high degree of security with the Java Card. Not just encryption, but also message integrity and authentication of the data could be accomplished, even though that hasn't been used.

No access control is made in the wallet to see if the billing party has the right to issue a bill the easiest would be to check the AID of the callee, but since there is no full control of the installation and removal of applets it would be possible to impersonate an AID. So the most secure thing would be to use a handshaking mechanism based on the exchange of cryptographic keys [CHE00]. e.g. issue a challenge send data to be encrypted and then check that it was correctly encrypted.

Since the bank is double checking all transactions to make sure no one tampers with the Java Card a couple of different schemes could be used if such an event occurred: one idea is to send a SM to the wallet and terminate the applet so it stops functioning. Other possibilities is to change the design so a special SM from the stock analysis program to the content provider isn't sent and instead send the request for the stock analysis as a part in the SM sent to the bank. Then it is up to the bank to inform the content provider of the bill information i.e. what analysis that was requested. One last solution is to keep it as it is today but requiring that the content provider wait to send back the requested stock analysis until he has received information from the bank that the bill has been paid.

As mentioned previously there is no message integrity control but if it could be assumed that no one else has they cryptographic keys i.e. they have been distributed safely all that has to be added is an error detection code and sequence number. Then the receiver would be assured that no alterations have been made and that sequencing is proper. If the message also include a time stamp it could be assured that it has not been delayed more than what is expected from the network [STA00] An other scheme could be used since the information from the content provider doesn't require to be encrypted but only requires to be authenticated. Thus a scheme with a common key used by the sender to calculate a checksum, which is added to the message and then once the message is received the receiver calculates the checksum for the message and sees if it is the same that is attached to the message. See e.g. [STA00] for further details of different authentication schemes.

3.3.5 Evaluation Summary

As can be seen from Table 3-10 there is very small increase in execution time between the 3DES and plain text versions thus it lies around 3 % or 0.7s for the median values i.e. it's neglible to the user whether encryption is used or not. The increase in execution time mainly comes from the encryption on the Java Card even though the decryption also takes some time to perform.

For performance issues it worth noting that a considerable performance increase can be accomplished in those cases where it's possible to use a transient array instead of a non transient array. In the measurements done there was a difference of 56% when encrypting a 128 byte array. The use of transient arrays are not just for encryption, but should be used for all temp data that is read/written a lot and doesn't need to be persistent between sessions. This since the transient array is cleared either at "deselect" of the applet or "card reset" depending on how it was created.

The individual measurements showed that: encryption time both at the SIM card as well as at the server increased almost linearly with the amount of data to encrypt. This increase in time is neglectable compared to the total execution time, at least since there is an upper limit on how much data the SIM card is possible to handle.

Worth noting is that it's impossible to transaction the sending of a SM since there is no way to reclaim a SM that has been sent, but it might be possible and desirable to log such an event so that the wallet always is in a consistent state.

With the design used there should be a bit better performance than what is shown since the second SM from the phone to the content provider might be sent right after that the first SM and thus they are sent somewhat in parallel.

The assumption made before that there shouldn't be much difference between the 3DES and plain text versions were to true even though there was a small increase in execution time. When looking at execution time the majority of the time lies in the communication done using SMs between the phone and the servers.

Possible Improvements

As can be seen the part that effects the execution time the most is the communication done using SM. Even though the communication between the SMS-C and the content provider and bank servers takes some time it is very small compared to the time it takes to get a SM delivered. It's very important to try to reduce the amount of data, since using several SMs for the communication dramatically increases execution time.

If a closer integration between the applets issuing bill requests as well as the wallet were done no separate SM to the content provider would be necessary. Instead the bank would inform the content provider once a bill has been paid and supply the additional information about the wanted request from the content provider. By doing this one less SM would be needed and a significant decrease in the total execution time would be accomplished.

To ensure that if money has been withdrawn the billing party is informed, transactions and logging should be used. Transaction works as in databases i.e. either all operations are committed or none. This to ensure that money hasn't been withdrawn without an SM being sent.

Execution time at the server side could be done much faster since today each update of the database involves the writing of a file.

Since there is no message integrity except the one offered by the OTA layer, see Table 2-2, this solution is only suitable for the payment of "small" amounts. An extension using message integrity and authentication so all parts are sure whom they are communicating with would have a impact on the amount of data sent and thus increasing the total execution time. The most frustrating part is that there is a silent interface i.e. no sort of progress bar so the user has no idea how much more time it will take.

Future Work

No measurements with a high number of clients were made due to a couple of reasons since a SIM card without subscription were used and the simulation environment were run at the same machine it would not be possible to test the performance in a good manner. If tests are to be run later one important aspect to consider is to have the clients running on different machines since otherwise the requests would be made in serial order and not in parallel.

One thing to investigate further is if there is any performance to gain in trying to do compression of the data sent with SMs, since it's in the communication with SMs that takes the most time. A possibility to reduce that time by a couple of percent would have a much larger impact than improving the server performance or the communication between the different servers.

First Impressions

It was fun to code Java Card applets since the programming were done on a much lower level than you are custom to when you code ordinary Java applications, but at the same time more burden must be carried by the programmer to write high performing code. This since Java Card have very limited processing power and thus it's imperative that the programmer really understands the hardware and what is going on. It's also true for ordinary Java applications, but there it's much easier to buy a little more memory or CPU power, which can't be done for a Java Card.

In Java Card programming a much more through understanding is needed since e.g. it is the programmer that must set the parameters used in the SM header him/herself. This part is poorly documented in the Java card API regarding SIM Toolkit instead a look at the ETSI specification is required. Not just the SM part, but also the other parts of the SIM Toolkit are poorly documented in the API e.g. it is hard to understand all of the proactive commands without prior knowledge of the SIM Toolkit.

3.3.6 Development: problems, environment & time

Problems

When choosing to develop a new SIM Toolkit applet, the development environment suggests a AID to use, but the problem is that the AID is suggested doesn't conform to the standards regarding SIM Toolkit applets. The length of the suggested AID was eight bytes but the standard says that a SIM Toolkit applet should have a length of 16 bytes where bytes 13-15 is the TAR value for that applet. This led to that getting hold of the SIO was impossible and it took a lot of time and troubleshooting to find out that it was the suggested AID that was the problem.

One interesting thing was that after installing an applet and package and then removing them the free memory available had diminished, even though it had diminished with just a couple of bytes, but anyhow it had diminished. So if the applets were installed and then removed repeatedly the card would finally run out of memory.

Case3 Gemplus development environment is said to support breakpoints in the code during debugging, but it was impossible to get this feature to work. Instead a scheme of displaying a short value on the device screen was used to be able to trace the program almost as using print statements in an ordinary Java program. But this possibility to use print statements is only possible if it's a SIM Toolkit applet if it instead was an ordinary applet some other scheme would have to be used.

Other problems that occurred was when development was moved from the simulator to the real card, then the encryption stopped to function since a security exception was thrown, which was solved with a little assistance from the thesis supervisor Jesper Söderlund. The problem was that when a SIM Toolkit applet is running the JCRE sees the selected applet as the GSM application i.e. the application managing the communication with the phone as well as all the SIM Toolkit applets. This led to that a security exception would be thrown if the cipher instance couldn't be reached through a SIO depending on the parameters it was created with.

Development time and environment

Can feel unusual for an ordinary Java programmer to code Java Card applets since most parts are done on a much lower level e.g. there are no Strings and no possibility to use print statements etc. Also a much more thorough understanding of the JCVM compared to the JVM is needed to develop good products due to e.g. to know how the SIO works and so forth. One good thing with SIM Toolkit applets compared to ordinary applets is the possibility to display information on the device screen and thus having a possibility to trace the program. This is a possibility that is not available for other types of applets even in simulated environments at least those that are available for free from e.g. SUN. On the other hand there are no free environments or tools for the development of SIM Toolkit applets at least not today.

As mentioned in the previous section *First Impressions* there is a rather high burden on the Java programmer to really think about what he/she is doing, e.g. the implementation of garbage collection on a Java Card is only optional, i.e. it is imperative that the programmer knows what he/she is doing and on what type of hardware the program will run.

The development environment used, GemXplore Case 3, consists of a couple of separate programs and an add on to either BEA Visual Café or Jbuilder. The separate programs are used to e.g. simulate a Java Card as well as the sending of SMs to and from a simulated GSM phone. Case 3 is also said to support breakpoints in the code during debugging a feature that never worked.

When it comes to downloading applets to the phone from Jbuilder a tool called JCardManger was used there are a couple of downloading methods supported e.g. download simulating the use of SMs. Most annoying was if several applets were to be installed each applet AID and package id had to be typed in manually and then installed in turns instead of having a batch or makefile feature allowing several applets to be installed "automatically". Deleting applets were also cumbersome since each AID and package id to delete had to be entered manually instead of having a feature of selecting among applets that were present on the card. When using one type of installation method the AID and package id wasn't needed to be typed in manually, since then the old values that had been used were present in a dropdown list. It was most certainly a bug that the dropdown list wasn't available for all the different download methods.

The most tedious part when developing for Java Card was the amount of delete and installs of applets that were needed. For example if a there are two applets A and B where B depends on A e.g. B using a SIO of A. Then if a small change is needed to be done to A first B has to be deleted then A and after that installing A and finally B again. Here some sort of automatic or batch like procedure would be wanted since it takes time and is extremely boring to do this especially during debugging when it has to be done often.

Several different programs need to be used one to develop code, Jbuilder, one extension to Jbuilder that handles the download and then finally a totally separated program that handles the execution of the phone and the simulation environment. All this should instead be integrated to one program since then it would be much easier instead of having a lot of different applications running.

The user manual lacks a lot of information about the tools e.g. what different fields are supposed to be used for and so forth, there are some references to documentation that is installed on the local machine but not even that documentation got all the details necessary.

It's hard work when something works in the simulator and not on the actual card since it's difficult and cumbersome to debug and sometimes it feels frustrating not to be able to look inside the device to see what is going on.

Error messages from the development environment was sometimes very poor e.g. when trying to install an applet it once resulted in a error code, which just indicated an unknown error leading to a lot of trial and error to find out what went wrong.

Despite all problems mentioned there was just a couple of minor issues when moving from a simulated card to a real card, which was very surprising since it usually leads to a lot of problems.

One feature that was useful was the program to design the program flow by using "boxes" e.g. for the sending of SMs, displaying menus, using branches and so forth. After it had been designed the auto-generated code could be opened in Jbuilder and further developed there. One problem was that it wasn't possible to go back to the design and do changes without removing everything that had been done in Jbuilder i.e. just possible to do the major design once.

The overall experience when it comes to the use of GemXplore Case 3 it's a descent program but not much more, this probably due to it's a very new technology leading to error prone behavior until all bugs have been removed.

3.4 Java in the phone evaluation

For more detailed results a look should be made to [PEL02] since this is just a summary of the most vital parts.

3.4.1 Application Design

The application design is based on simple client-server architecture. The client side consist of an application with the purpose to give the user the possibility to upgrade a game. To be able to upgrade the game a connection and verification of the user must be established with a server that have the ability to withdraw money from an account where the user has money.

The application has been divided into two cases: one without any crypto and one with encryption and decryption of information sent between the client and the server. In the no crypto case the client sends the information to the wallet server in plain text and in the other case the client, and server, encrypts the information using single DES before sending.

3.4.2 Implementation and limitations

The client side consists of a MIDlet (a J2ME applet) that is responsible for the graphical user interface (GUI) towards the user, the establishment of a connection between the client and the server, and in the crypto case also responsible for encryption and decryption of the data sent between the client and the server.

The server side consists of a servlet that runs on Tomcat (a free servlet engine). The server receives information from the client and decrypts it if necessary, it requests user information that is stored on the server and validates the username and signing PIN. The server then examines whether it is possible to withdraw money from the user's account, whether money is withdrawn and information about the transaction is printed out at the server. Then the result is sent back to the client, either encrypted or in plain text. As the client has received and processed the result an appropriate action is taken e.g. update the game if the transaction was successful and display information about the result for the user.

Different servlets are used for the plain text and encryption so that the servlets know what information to expect. Different servlets will also be used during the evaluation when altering the amount of data sent between the client and the server (decrease/increase data).

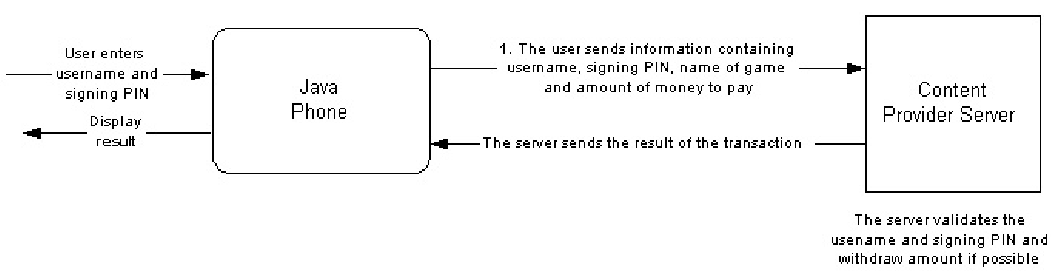

Figure 3-25 The communication between the Java Phone and the content provider.

The structure for the information sent from the client to the server is the following: variable#result#variable#result…etc. Where '#' is used as a separation character so that the server can easily extract the information it needs from the data received from the client.

Example of a message: 'username#peterl#PIN#0123#amount#1#game#Arkenoid'. What the message means is that the user "peterl" has entered his PIN "0123" and wants to pay $1 to the game "Arkenoid".

3.4.3 Evaluation

The most important aspects of the evaluation are the response time and security. Time is important because the user doesn't want to wait long for the application to finish. If it takes too long the user will be bored and will not use it again. The security issues are important since you want the payments to be performed correctly and in a secure way.

Time estimation of the most contributing parts

The parts that were assumed to contribute the most to the total time were the encryption and decryption of data and the transfer of data over GSM or GPRS.

The encryption and decryption of data should be time consuming since the application needs to perform complex algorithms on the data to transform it and the processing power of a mobile phone is much lower than of a PC. This means that the encryption and decryption on the server will probably not contribute that much.

The transfer of data will probably contribute the most since it takes much time to connect and to transfer the data [ROS02]. It will probably differ much in time between using GSM or GPRS since GPRS connects faster and has a higher bandwidth [TEL02].

Testbed

The structure for evaluating the time has been the following. The application has been divided into different parts, so that it would be possible to see what part that contributes the most to the total time. The timer starts after the user has entered a username and a signing PIN and has pressed the upgrade button.

Altering the data sent

In the case of encryption there are measurements with respect to the number of bytes sent and received. The bytes sent have been changed to four different sizes. The different measurements of the different sizes are hereafter referred to as CryptoShort, Crypto, CryptoLong and CryptoExtraLong where Crypto is the original size. The information sent with Crypto is of the structure variable#result#variable#result…etc. With CryptoShort the information has been decreased to only contain the results: result#result…etc. CryptoLong and CryptoExtraLong contain the information sent by Crypto repeated several times. The sizes in bytes of the information sent and received by the different applications are shown in Table 3-13.

Table 3-13 Data sent from and to application (in bytes)

Name	Sent	Received	Total
NoCrypto	47	18	65
CryptoShort	34	8	42
Crypto	58	24	82
CryptoLong	202	64	266
CryptoExtraLong	770	96	866

Altering the transportation carrier

The first phone that was used in the testing was Siemens SL45i. It was one of the first phones to have J2ME and therefore it doesn't have GPRS. An old phone also often implies a lower processing power, which would probably lead to slower execution of the Java applet. After using the SL45i in the tests a first reaction was the long response time of the application, too long to be acceptable. It was important to evaluate some other alternative and therefore tests began on Siemens M50, which is a more recent phone and with GPRS. By doing the tests with M50 it is possible to see the impact on the response time when using GPRS instead of GSM as well if there is a difference in processing power as a newer phone is used.

Results

See either *4.1.2 Java Phone* for a summary of the results or [PEL02] for more thorough details regarding the performed measurements

First Impressions

The main advantage with J2ME is that it is Java, so if one have experience of the J2SE it is quite easy to understand the structure of this new technology. The terminology is similar and some of the classes are the same as used in J2SE, though many classes have been shrinked to a size more suitable for the smaller platform, which lead to that the functionality is not as powerful as in J2SE. The basic structure with an applet as an executing instance is good since applet is a well-known concept, even for those who do not know Java programming.

J2ME is an established technology and there are therefore many development tools and example applications. This is a great help, and makes it easier to get started, when trying to learn how to write an applet. Since the documentation is so extensive it should be quite easy even for someone that only has some or no experience from Java to get started.

3.4.4 Development: problems, environment & time

The basic set of development tools consists of a text editor, a J2ME compiler and an emulator for MIDlets. All these things can also be included in an IDE. The software that has been used in this project is a combination of different tools. An IDE named Sun One Studio Mobile Edition has been used sometimes and sometimes a text editor TextPad together with the J2ME complier and the emulator from Sun, J2ME Wireless Toolkit. In addition, other emulators have been used, both from Nokia and from Siemens. SonyEricsson had different skins that could be imported into the Wireless Toolkit.

Overall the development tools have been working satisfactory, but as always with emulators, they newer show a correct image of the real device. They always seem to handle a lot more and a lot faster that the actual device. One advantage with the Wireless Toolkit was that it was possible to adjust the capacity of the emulator (i.e. set the amount of byte code that the emulator should be able to handle), by lowering the amount of byte code one could get the emulator to emulate closer to the actual device. Even if one would succeed in matching the level byte code, it is most likely that the emulator still won't give a perfect image, but it will send you in the right direction. Other advantages with the Wireless Toolkit are that is possible to view the memory use and network traffic. Emulators from Nokia and Siemens were also used in the project and those emulators could give a more precise image of the actual devices.

Since the emulators aren't as good as one could expect, there is a need for testing the application (in its different forms) on actual devices. The first device to test on was SL45i Siemens, which was one of the first mobile phones with Java, which results in that it is slower than later phones. Another disadvantage is that it doesn't have GPRS, so the connection must be done over GSM instead, which makes it slower to perform the transfer of information to the server. Later on there was testing on M50 Siemens which has both Java and GPRS and is a quite new phone, released this summer. There was a big difference in execution time (including the connection and transfer) between the M50 and SL45i. The M50 was around three to four times faster than the SL45.

Security evaluation

The application has been evaluated with respect to security, and the following issues have been thought about:

- There is no solid defense against that someone changes the information that is sent on its way to the server since no sign of the information is done. Though it is possible to visualize the cryptographic solution that is used as some sort of sign, since the key is unique for the user, but at the same time someone could figure out the key and then be able to read encrypted information and send the information modified. In the application only single DES with a 56-bit key is used, which isn't so difficult to crack if you have powerful processor resources [STA00]. It would be preferred to use triple DES with a larger key (128, 162, etc) to make it more difficult to crack (find the secret key). Another approach of encryption algorithms was to use RSA instead of DES, but the lightweight crypto API [BOU02] didn't seem to support this, there were some support but not the whole chain of encryption.

- There is no checking at the server that the same request isn't performed twice, i.e. the user was too fast to press so two requests have been sent to the server and the server will then take care of and debits both. At the same time this might not be important for this application. This could be solved by giving the upgrade an id that is being saved with the transaction at the server. If the server would receive two payment requests with the same id, the last one would be ignored. If you still want to do the same upgrade several times, you could also send the current time to compare at the server and ignore requests that contains a timestamp that has already been used. This will reduce the risk for a replay attack (that someone get hold of one of your messages and sends it over and over again).

- The application doesn't use any MAC to see that the message hasn't been altered. (MAC, Message Authentication Code, is a symmetric cryptographic transformation of data that provides data origin authentification and data integrity). It would be good to have a MAC sum, but hard to implement since the messages sent look different. To be able to use MAC one could only have one game with one amount of money, which would be very narrow.

- The messages that are sent are quite similar, which could make it easier to crack the key. This might be improved if some random text is inserted into the message to make it vary more.

Problems

Some manufactures of mobile phones states that they support the MIDP 1.0, but it turned out Siemens didn't support a function called **delete**(int itemNum) that belongs to the Form object that is part of MIDP 1.0. The other emulators didn't complain about the function, but the Siemens emulator turned itself off. When the code was deployed at the Siemens SL45i, it caused the phone to sometimes turn itself off, the other times it reported a memory error. This is a bit strange since the idea of Java is that it supposed to be encapsulated in a way that it protects the phone from strange commands. Since neither the emulator nor the actual phone could give any error messages containing interesting debug information, the method trial-and-error had to be used.

It is often a problem with development tools for new technology that it is a long way to perfection. They don't work as well as you would want them to and the debugging is very limited.

Another error that occurred was a mistake in the writing of the code. To transfer information, the method POST (instead of GET) is used. When you prepare the connection for writing you must set the content length with the amount of data that is to be transferred. The data that is to be transferred consist, in the case of encryption, of the encrypted message, as a byte array, and the identification number, also as a byte array. These arrays are merged together. The problem was that the content length was set to the length of the encrypted message (in bytes) not the length of the merged array. This might had been quite easy to find if it wasn't for the fact that the Wireless Toolkit didn't complain about this, it seemed to ignore the size of content length and just transferred the data, and the same thing was with some of the Nokia emulators. But one Nokia emulator and the Siemens emulators didn't work properly, since the transfer was cut off, leading to that the encrypted message was cut off, since the identification number came first. When the encrypted message wasn't complete it couldn't be decrypted. This is a good example of how some of the emulators handle more than the actual devices.

3.5 WIB with Java plug-in evaluation

First Impression

The first impression was that this use of Java was quite different from the platforms one is familiar with. The development environment was different, with an unknown IDE and a special plug-in. The documentation that came with the development environment was not as detailed as one might had wanted, which made it hard to get started. The documentation on the Internet with information about Java Card and example applications is much smaller than of other Java platform like e.g. J2ME. Fortunately some example applications came with the development environment that could be examined to learn about the functionality one wanted to use.

It was a challenge to program for the Java Card since it feels like more low level programming than ordinary Java, but some functionality was missed. Especially string operations (the class String does not exist) and an advanced print out functions (it is only possible to print an array of bytes) would have been nice to have. The lack of functionality made the debugging of the code more problematic.

3.5.1 Development: problems, environment & time

The development environment consists mainly of Oberthur products and modules integrated into other products. An IDE from WebGain called VisualCafe is used for development of the Java Card applet. To this IDE Oberthur has written a module so that it is possible to compile the source code of the applets and convert them into a .cap file. The .cap file is used to download the applet to the Java Card via a card reader and a program called ALM Server. In the IDE it is possible to, in some extent, debug the Java Card applet via a mobile simulator that runs the applet. The mobile simulator connects to different readers that could either contain a real card or a simulated card, which is the case when debugging the applet in the IDE. The advantages with the simulated card are that it is easier to get an error message than running the applet on a real card, and the procedure from compiling the code and running the applet is faster than the procedure with downloading the applet onto the Java Card before being able to run it.

The card used for the WIB plug-in is a special card from Oberthur. To be able to call the plug in from the WIB, classes from Oberthur are being used. For the plug-in there is a converter environment, also from Oberthur, that is used to produce the cap file.

Problems

Even though Java Card has been around for a while, it is not as widely spread as J2ME. The first impression after working with the development environment for Java Card is that it has a long way until it can compare itself with the development tools for J2SE or J2ME in functionality and ease of use.

The license for the development environment has been a test license, which means that no advanced support has been given, though for really difficult problems support has been requested. The documentation for the tools hasn't been as thorough as one might have whished. After the installation of the IDE, example code was altered and was tried to be compiled and converted by the IDE. This however did not work and instead the IDE reported that there were a conversion problem and printed the error message "error=3". The documentation had no information about what the error could depend on, which made it very difficult to figure out what the problem was about. The answer to the solution was that the path (for execution) to the program used for conversion was wrong, since the program had been installed on another hard drive.

The Java Cards that came with the card reader had no note with the correct PIN to the cards. So one could not know what the correct PIN was and there are only three attempts for entering the PIN for each card. Fortunately there was an example application that had a simple PIN request and that PIN was 1111, that was tested and was correct.

The IDE with the Oberthur plug in is not always so stable, it freezes quite often, when run in debugging mode, and has to be restarted. Some of the problem with that the IDE freezes may have depended of a corrupted file, but the file had not been edited externally which probably means that the IDE had made the file corrupt in some way (or at least allowed changes in the file which lead to corruption).

There has also been a problem with trying to use cryptographic classes in the simulator, but that works ok when downloaded to the card. More specifically there is an exception thrown when trying to call the static method getInstance() belonging to the class Cipher, that should return an object of the class Cipher that is later used for encryption/decryption. The exception is probably generated because the algorithm used, DES, might not be implemented in the simulating environment [CIP02].

To be able to get the shareable interface object to work, the applet that is going to be shared needs to have an AID so that it can be identified and found by the other applet. Even if an applet is not using the shareable interface it will still be assigned an AID when downloaded to the card but in the case of shareable interface object the AID and the length of the AID is very important. Unfortunately no example code was supplied in the development environment, and almost no explanation of the structure of the AID, so it was not obvious how it should be. The default length of the AID was 12 characters which implies 6 bytes, but that had to be figured out. Even if the length of the AID in the configuration file was correct there could be wrong number of bytes in the source code, when trying to search for the applet with the AID, since one did not know from the start the relationship between the characters and the correct bytes. It finally worked when the relationship was understood and the AID that was to be used in the example plug-in was with the shareable interface application.

4 Discussion/Analysis

Worth mentioning is that both this thesis and [PEL02] were done in close collaboration with each other. The intention was to write one report together but due to factors at SmartTrust delaying [PEL02] it was decided to split the report into two separate parts.

4.1 Performance

4.1.1 WIB vs Java Card

In the following paragraphs there is a comparison between the WIB and Java Card regarding performance. First comes an overview of the total execution time then comes a more detailed comparison of the different parts concerning performance. Also the different environments used and experiences from the development are analyzed in section *4.2 Development environment/Learning threshold*.

In Table 4-1 there is a comparison between the measurements made for the WIB and Java Card see sections *3.2 WIB evaluation* and *3.3 Java Card evaluation* for the individual measurements. WIB measurements have been normalized to the Java Card i.e. the quotient WIB/(Java Card) have been taken for the median and average values. In the normalized case the Java Card is faster if it's a value above 100 and vice versa if slower. Because not just the relative difference between them are important the following subtraction WIB-(Java Card) were also made both for the average and median values, i.e. a positive value means that the 3DES case takes that much more time and a negative the vice versa that many ms faster.

Table 4-1 Comparison between WIB and Java Card execution time where the WIB is normalized to Java Card also the real difference in ms is shown.

Comparison of execution time between the WIB and a Java Card for the stock analysis application				
Action	Normalised WIB/Java Card		Real difference (ms)	
	Average	Median	Average	Median
User interaction start app.	151	137	2742	1974
User interaction type in password	200	184	2363	2033
Time spent sending SM 3DES	193	192	21501	20911
Time spent sending SM plain text	166	160	15340	13650
Tot. time for transaction 3DES	173.70	166.82	24429	22199
Tot. time for transaction minus user interaction 3DES	176.17	174.83	19325	18788
Tot. time for transaction plain text	158.46	159.71	18797	19088
Tot. time for transaction minus user interaction plain text	154.65	154.60	13596	13332

From Table 4-1 and Figure 4-1 it can be seen that the Java Card is significantly faster than the WIB when it concerns the total execution time. Looking at Table 4-1 it's obvious that most of that difference in time comes from the sending of SMs see Figure 4-2 for an illustration of that difference between the WIB and Java Card.

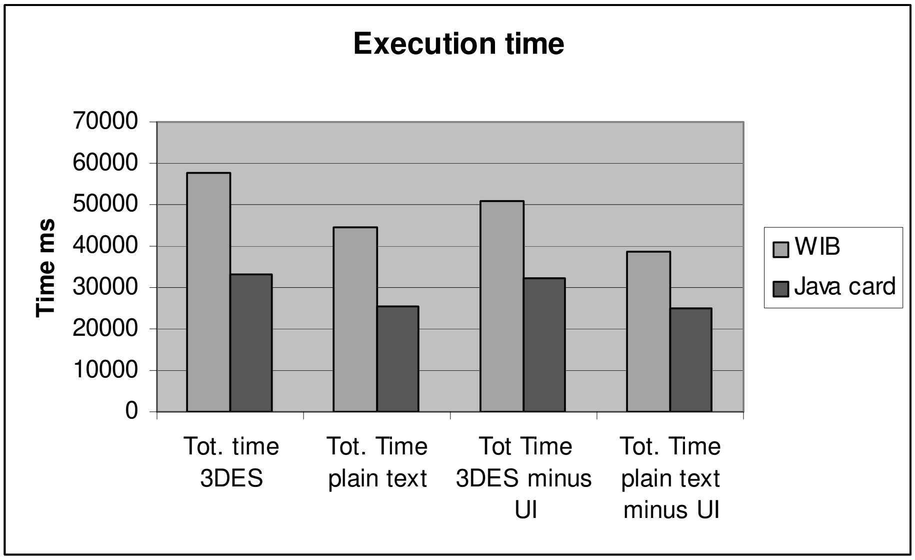

Figure 4-1 Comparison of the total execution time between the WIB and Java Card cases.

If a closer look of the program flow is made e.g. from Figure 3-4 and Figure 3-17 it can clearly be seen that more SMs are used in the communication between the phone and the servers when the WIB is used. This comes from the fact that in the WIB case all of the wallet logic is located at the bank server while in the Java Card case the wallet logic is located on the SIM card. Java Card has the possibility to maintain a state while the WIB is "stateless" and due to this much more can be performed locally, which reduces the communication with the servers. For example in the WIB version of the stock analysis program it must first request an analysis from the content provider then the bank pushes a bill to the user to sign and then the user sends the signed bill back to the bank, which in turn informs the content provider that finally pushes the analysis to the user, see Figure 3-1. In the Java Card version only three SMs are needed first the SM from the wallet to the bank informing it of the transaction then the SM from the stock analysis program to the content provider and finally the SM containing the analysis sent to the user, see Figure 3-15.

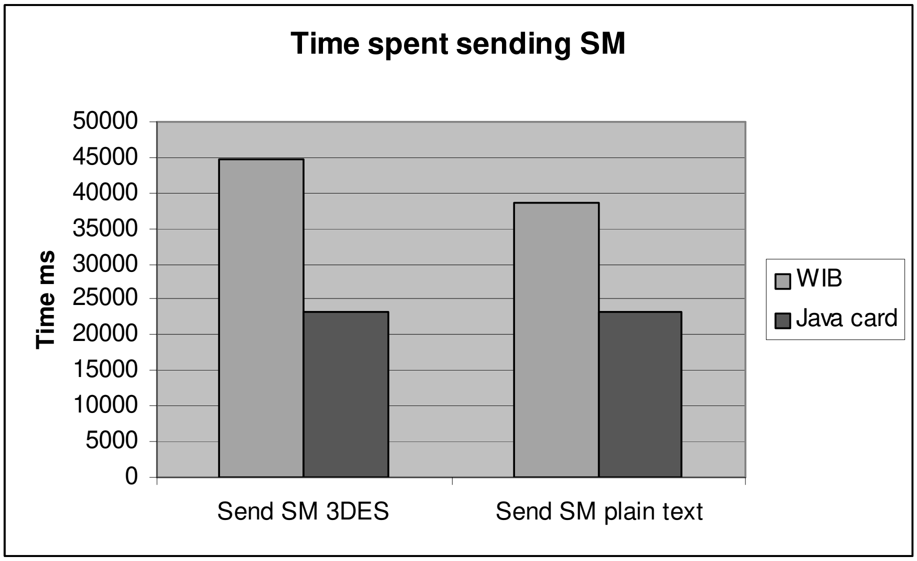

Figure 4-2 Comparison of time spent sending SMs between the WIB and Java Card case. Much more time is spent sending SMs in the WIB case since more SMs are required due to that the WIB is stateless, while the Java Card has the possibility to maintain a state.

As a more detailed comparison of the different encryptions times are made it's quite clear that when it comes to encryption on the SIM card the WIB is much faster than the Java Card, see Figure 4-3. This is at least true for the cards used, since performance vary among different WIB and Java Cards due to different SIM card vendors and their implementations of both the WIB and JCRE as well as different processors might be used on the SIM card. The huge difference probably comes from the fact that Java Card encryption is implemented as code that is run on a processor while the WIB card uses hardware optimised algorithms.

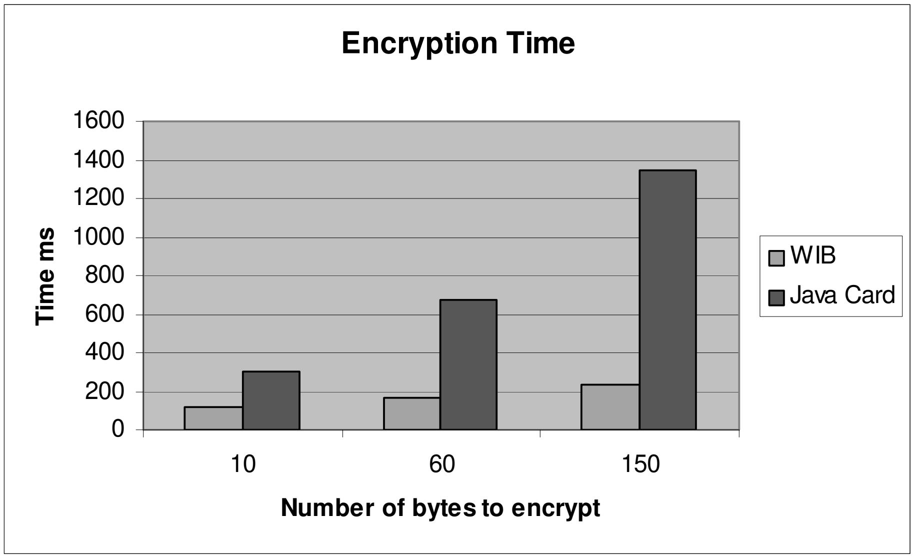

Figure 4-3 The time to encrypt a number of different bytes using either a Java Card or WIB SIM Card.

When it comes to encryption on the server side looking at Figure 3-11 and Figure 3-24 it is quite clear that to do the encryption using security center takes almost 1 s longer than using the Java crypto API. Most of this time doesn't have to do with the implementation of 3DES but rather with the fact that when using the security center most of the time comes from network communication and the latencies involved there. If a similar scheme with running the security center on the same machine as the server wanting to use the service this difference would be removed.

4.1.2 Java Phone

Results

Tests were done by running the applications five times or more, though most often only five times. The tests are also done with the change of using GSM (Siemens SL45i) or GPRS (Siemens M50). Several statistical values were then calculated from the results: maximum value, minimum value, average value, median value and standard deviation.

The largest contributor to the response time is, as predicted, the phase of communication with the server. This can be seen in tables Table 4-2 and Table 4-3 where the encryption has a proporation around 30% for the GSM phone and between 43-54% for the GPRS phone. Even though the GPRS phone has a larger fraction of the time spent in doing encryption the real time is even though shorter than for the GSM phone see Figure 4-7 for a comparison of the encryption time.

Table 4-2 Displays the total time in the GSM test as average and median and the percentage in both cases of how much server communication and encryption/decryption contributes to the total time.

Total time - GSM (time in ms)						
Test	Average	Server com	Crypto	Median	Server com	Crypto
NoCrypto	20191	95%	-	20181	96%	-
CryptoShort	37938	69%	28%	35961	64%	30%
Crypto	48581	74%	23%	36537	62%	31%
CryptoLong	51666	68%	29%	42536	61%	36%

Table 4-3 Displays the total time in the GPRS test as average and median and the percentage in both cases of how much server communication and encryption/decryption contributes to the total time.

Total time - GPRS (time in ms)						
Test	Average	Server com	Crypto	Median	Server com	Crypto
NoCrypto	6496	96%	-	5861	96%	-
CryptoShort	10380	54%	44%	10610	52%	43%
Crypto	10644	52%	44%	10384	53%	46%
CryptoLong	12057	48%	50%	12008	49%	49%
CryptoExtraLong	17239	46%	43%	16502	43%	54%

Another thing that can be noticed in general is that the standard deviation is much higher for the GSM case than the GPRS case. The high standard deviation shown in Figure 4-4 and Figure 4-5 probably indicates that setting up a connection and transferring data is much slower and randomized as a dialed up connection is used compared to a GPRS connection. A problem with the high standard deviation is that it makes it more difficult to see the connections between the time and different parameters.

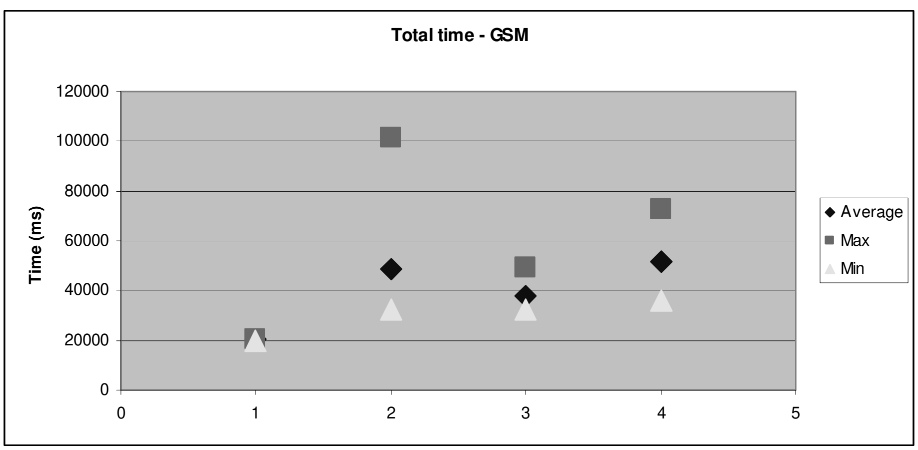

Figure 4-4 Total execution time using GSM note the high standard deviation.

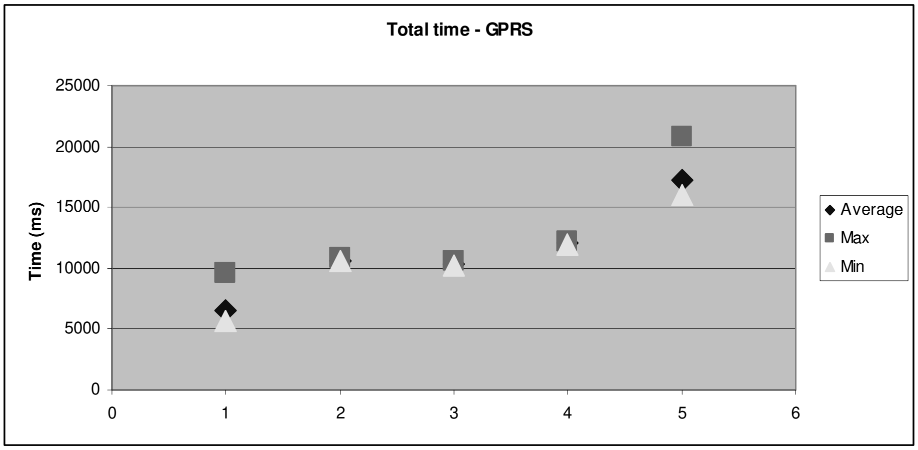

Figure 4-5 Total execution time using GPRS.

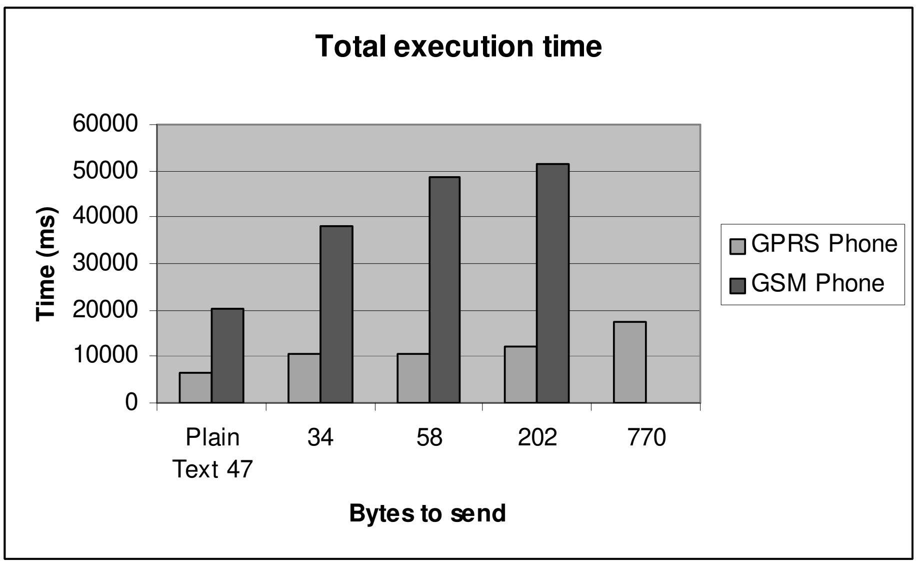

Figure 4-6 Comparison of the total execution time between using Siemens SL45i and GSM 03.48 compared to a Siemens M50 and GPRS for the communication.

The assumption made that the encryption would be a big contributor to the total time turned out to be true. The time that the encryption contributes with varies between 30-50% depending on whether GSM 03.48 or GPRS is used for the communication. In the test with GSM, the connection with the server is so time consuming that the encryption contributes with about 1/3 to the total time. But in the test with GPRS, the encryption contributed much more around 45-50% to be precise. Even though the encryption contributes much more to the total time as GPRS is used, the time taken to encrypt actually is less see Figure 4-7.

The large difference in encryption time that can be noted in Figure 4-7 is probably due to the fact that the GPRS phone is much newer and thus has a better processor as well as optimization that might have been done in the J2ME implementation.

Table 4-4 Time taken both to encrypt a number of bytes either using a Simens SL45i or a Simens M50.

GSM 03.48 phone Siemens SL45i used for encryption					
Bytes	Average	Max	Min	Median	Std. Deviation
34	2166	2211	2137	2146	32
58	2533	2561	2510	2534	19
202	4932	4980	4860	4947	43
GPRS enabled phone Siemens M50 used for encryption					
Bytes	Average	Max	Min	Median	Std. Deviation
34	836	845	830	831	8
58	950	965	940	947	9
202	1698	1763	1670	1685	37
770	4504	4513	4500	4502	5

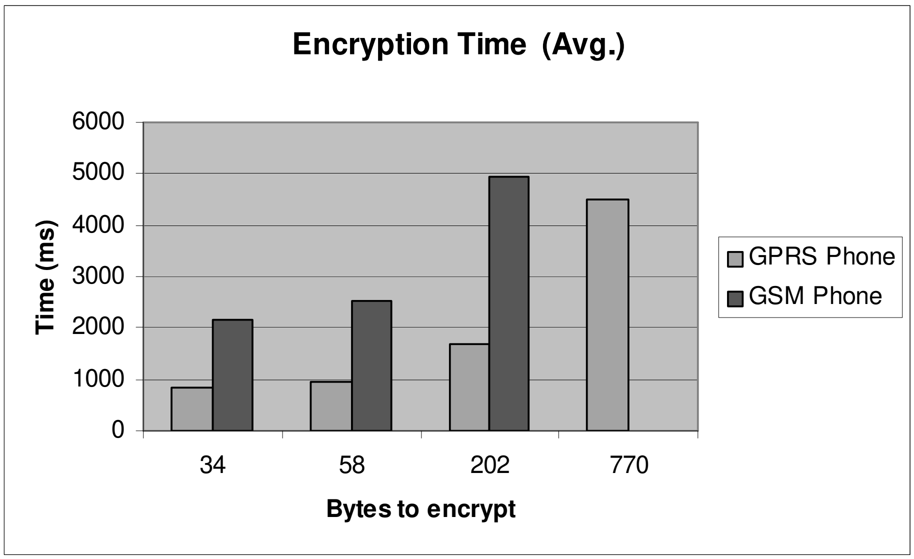

Figure 4-7 Comparison of the encryption type when using different phones a newer Simens M50 (GPRS Phone) and an older Siemens SL45i (GSM Phone).

As can be seen both from Table 4-5 and Figure 4-8 that the time stays rather constant instead of increasing linearly, due to this the conclusions could be drawn that probably most of the transmission time comes from the setup of the connection at least when it concerns sending such small number of bytes as has been done.

Also worth noting is that there is a much higher standard deviation for the dialed up connection compared to the packet service used by GPRS see Table 4-5

Table 4-5 Time taken both to set up a connection and send the actual bytes, both using GSM 03.48 and GPRS for the communication.

GSM 03.48 phone Siemens SL45i					
Bytes	Average	Max	Min	Median	Std. Deviation
34	24230	36486	19125	21750	6958
58	34260	87635	18829	21538	29863
202	33782	54965	18142	24709	15411
GPRS enabled phone Siemens M50					
Bytes	Average	Max	Min	Median	Std. Deviation
34	5546	5976	5344	5427	269
58	5465	5676	5363	5450	125
202	5737	5834	5653	5741	79
770	7808	11482	6516	7070	1894

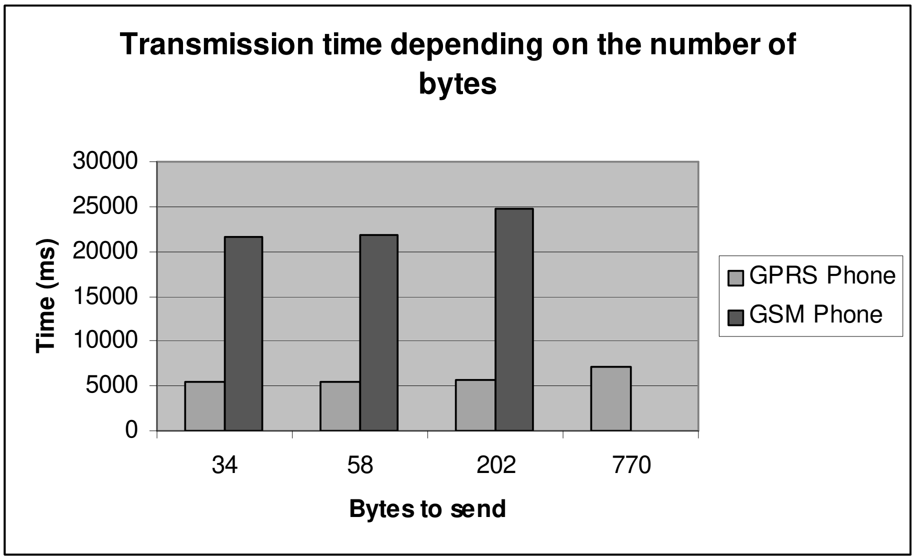

Figure 4-8 Transmission time and how it depends on the amount of data to send. For a closer detail on the measurements used in the graph see [PEL02].

4.2 Development environment/Learning threshold

First there will be a comparison of the different development environments available and then a bit in general concerning the differences in developing for the two architectures.

When developing for the WIB there is a lot of freedom it could be done using almost just freeware tools such as emacs, ant (Another Neat Tool) and a web server like tomcat running on Linux. Just some things are needed that aren't free: a simulator to simulate a WIB phone, the software used to convert WML to byte code, software to download the byte code to the SIM and finally the possibility to use a WIG server. The greatest benefit is that the development can be done in an environment that the developer is used to e.g. Jbuilder and JSP, Microsoft Visual and ASP, emacs and php or any other tool the developer might wish to use.

Developing for Java Card and WIB with Java plug-ins isn't as free as developing for the WIG when it comes to choosing the tools to use. There are freeware tools to develop applets but those tools don't offer a good simulated environment that can be used for debugging and are neither focused on SIM Toolkit applets. Instead some development kit must be bought from a card manufacturer that many times promote proprietary APIs leading to non portable code i.e. applets only working on the card manufacturers cards. One thing that the development environments had in common was that they were buggy, had a tendency to crash and were poorly documented.

A developer could be arguing that it is more "fun" to code a Java Card applet than to write a WIB application, but this is very individual and may very between different developers. But it must not to be forgotten since it's obvious if you are doing something that feels fun you do a much better work.

When it comes to development for Java on the phone (J2ME) it is obvious that it's a much more established technology than Java Card. Development environments are very well adopted and are available for free both in the sense that the developer can choose a environment that suits him well as well as it is free when it comes to cost.. The only thing that costs money is to download the midlet OTA, but it's also possible to download the midlet through a cable when it's connected to a computer. There are also a lot of emulators available for free both from SUN as well as the different phone manufacturers.

But as always with emulators, they newer show a correct image of the real device. They always seem to handle a lot more and a lot faster that the actual device. One advantage with the Wireless Toolkit was that it was possible to adjust the capacity/performance of emulated device (i.e. it was possible to set the amount of byte code that the emulator should be able to handle). One other nice feature with the Wireless Toolkit is that is possible to view both memory use and network traffic.

When it comes to learning threshold it's goes much quicker to get started with the WIB compared to WIB with Java plug-in and Java Card, even though the developer doesn't have any previous knowledge from web server programming. In contrary the learning threshold feels a lot higher for the Java Card environment, since everything is done at a much lower level than in the WIB case e.g. setting the bits in the SM header, instead of just writing a go href as would be the case for the WIB. To fully utilize the Java Card the programmer need to learn SIM Toolkit commands on a much more detailed level than is the case for the WIB where almost a basic knowledge of web programming is all that is needed. This problem with low level programming also occurred when WIB with Java plug-ins was used at least the part that concerned the Java plug-ins.

The obstacles regarding the WIB mostly had to do with documents not being up to date and settings on the WIG server or something similar e.g. whether the SIM card used encryption. While developing for Java Card the problems had much more to do with the development it self like problems of getting hold of SIO references and so forth even though some problems arose from the development environment.

Problems that occurred during the development for Java on the phone primarily had to do with the hardware since it said that it supported MIDP 1.0, which it apparently didn't do totally. Sometimes a bit more could be wanted from the tools when it came to debugging, but still the tools available for debugging J2ME applets are much better than those available for Java Card. If the developer has any previous knowledge of Java programming it goes very quickly to get started do develop applications for J2ME even though the developer may miss some of the features available in J2SE, due to the reduction that has been made to J2ME. At the same time the programming is made at a much higher level than what is the case for Java Card, very much due to the fact that the greater processor power allows a larger API that simplifies the development compared to the microprocessor on the SIM card.

Worth mentioning is also the example source code available e.g. on the Internet, since that is very good source of information to look at once a developer get stuck with a problem. J2ME comes way ahead of the other technologies as it comes to both source code available as well as other forms of written material, discussion forums and mailing lists. Shortly after comes the WIB information at least since the part that concerns web server programming is very well documented even better than J2ME. But when it comes to writing wml pages that are understood by the WIB/WIG SmartTrust documents are required and usually holds are as good as could be required. Ordinary Java Card applets aren't that well documented yet due to a couple of reasons. It's both a new technology and rather few developers are using it compared to other techniques out there. Especially there isn't much information available regarding STK (SIM Toolkit) applets that are used on SIM cards. Dead last concerning available information is WIB with Java plug-ins since it a very new and small technology hardly used anywhere at least yet, but the example code that came with the development environment was satisfactory.

When it comes documentation level concerning the development environment it's only worth mentioning that those for Java Card and WIB with Java Plug-ins lacked a lot of details and could have been much better. The other environments used either were self-explanatory or thorough documentation was easily available.

5 Future Work

There are numerous of things that could be looked upon e.g. in a real world service there must be performance evaluations of the servers done in order to see how they behave under a heavy load. This is important since mobile services are many times offered to millions of subscribers. Changes are also needed if the Java Card service were to be moved to a live environment, since now it only runs in a simulated environment.

In the future there is a lot of interesting stuff coming such as the JSR that will allow the midlet to communicate with the Java Card. This will allow an easier way for midlets to authenticate them selves as well as adding extra security and "safe" storage. So this would be very interesting to investigate further and see what the possibilities are.

An other interesting thing is that Microsoft has now started to ship software to run in "smart phones". Looking at history Microsoft has been neglecting security and instead focused on customer appealing applications and ease of use. The telecom market has on the other hand focused on reliability and security so it would be interesting to compare their Symbian based phones with phones running the Microsoft OS. This is especially interesting since it's likely that in the future most of the services will be "smart phone" based instead of SIM card based.

Nowadays both mobile operators and phone manufacturers are moving the focus more and more from the SIM card to the phone. Phone manufacturers are storing more and more information in the phone such as phone numbers instead of storing them on the SIM. Large mobile operators like Vodafone and Orange have started to require customised phones from the manufacturers with special menus that are not SIM based but reside in the phone [NYT02]. It would be interesting to see how the mobile operators look upon the SIM card in the future since today it's one of the most strategic points where they have "total control" over the customer.

On the Internet there have been a real hype regarding web services so it could be interesting to see if it would be possible to deploy the stock analysis service as a web service. It would also be interesting to look upon how it's possible to offer web services for mobile Internet users.

Many Java enabled smart cards are coming out especially in the form of credit cards due to the increased security level that a smart card offers compared to just having a magnetic strip on the card, also identification cards are coming as smart cards. One interesting aspect in this area is whether SmartTrust's in depth knowledge of SIM cards also could be applied to other types of smart cards. At the same time it would be interesting to see whether there will be a merge of different types of smart cards, e.g. today each store has their own club card an alternative solution to this approach would be that the store installed an applet on a Java enabled id or SIM card.

As more and more phones comes out that support WAP 2.0 the largest benefit with the WIB is lost i.e. the possibility to achieve end-to-end security that wasn't possible in WAP until the release of WAP 2.0 [IBM02]. Before WAP 2.0 the data were in plain text for a short while in the WAP gateway during the translation between the protocols used in the fixed and wireless "worlds". This is probably the largest threat to SIM based services since then it will be possible to get a secure and fast communication using GPRS and 3G and thus eliminating most of the communication time. This would have a large impact since in the case study approx. 1-3 s is spent in execution at the client and server side and the rest is made up of communication of sending SMs. Worth mentioning though is that in the future other transport mechanisms such as GPRS will be possible to use for the communication with the SIM, but there is still a limitation for the use of pictures embedded in the page to be shown [GSM02].

This increased security in WAP 2.0 and the JSR allowing communication between the SIM and the midlet looks as the most interesting issues to investigate further.

6 Conclusions

The idea was to have a wireless wallet as a framework for the evaluation, i.e. all technologies were evaluated with this common denominator when the following parameters were evaluated communication, security features, performance, development environment and learning threshold. A breakdown of the total execution time was made for each architecture to be able to see how much the different parts contributed to the total time. In addition to this breakdown of time other aspects like encryption and communication performance were investigated in detail.

When it comes to communication, the need and amount depend on both the communication medium available for the architecture as well as the design choice. All of the SIM card architectures uses SMS as the communication medium, but in the future other transport mechanism such as GPRS could be used [GSM02] and thus increasing performance dramatically. But as it looks today Java with a GPRS enabled phone stands out as it comes to communication. In section *3.2.4 Evaluation* the experiments showed that the sending of a SM took around 7-8 seconds per SM i.e. it would take approx 40-50 seconds to transfer 770 bytes through the SM channel. To do the same transmission using GPRS for the communication it only took around one sixth of that time or around 7 seconds see *4.1.2 Java Phone*.

All of the SIM cards offer the same security features as message authentication, message integrity and encryption but when it comes to Java on the phone that has to be implemented, since there is no support from the runtime environment. There are free third party products like bouncy castle but that leads to an increase in code size. One advantage with the Java Card is that information easily can be stored on the client side, but this may at the same time by a disadvantage from a security point of view. With the WIB you can be more certain that the user hasn't hacked the server, since it is much easier to control than if someone has tampered his or her card. The largest advantage when it comes to the possibility to store information on the client side is the possibility to reduce the communication with server side and this is very important at least when such a slow communication medium as SM is used.

As it comes to performance between the different architectures Java on the phone is outstanding when it comes to GUI capabilities since it easily can be programmed as desired, while the SIM cards only can utilize the possibility with SIM toolkit to set up menus and submenus used by the phone. One interesting fact is that the SIM cards outperform Java on the phone when it comes to encryption even though the SIM cards uses 3DES and Java on the phone only single DES. For the WIB it takes around 164 ms to encrypt 64 bytes while it takes 950 ms for Simens M50 and 2533 ms for the Simens SL45i to encrypt 58 i.e. it almost takes 6 and 15 times longer to do the encryption on the phone instead of doing it on the SIM card.

Looking at the different development environments available Java on the phone could be considered a "winner" closely followed by the WIB since both these technologies offer the most flexible and stable environments for the developer. It's also possible to use just freeware tools compared to the development kits required for both Java enabled SIM card and WIB with Java plug-ins. Most important probably is the fact that the tools available are very well documented and there are a lot of them to choose among. The main drawback with Java enabled SIM cards and WIB with Java plug-ins is that the tools have a tendency to crash and are poorly documented. Card manufacturers also tend to promote their proprietary APIs leading to non-portable applets.

The learning threshold depends very much on the background of the developer. One thing that can be said for sure is that Learning threshold for Java Card and WIB with Java plug-in is higher than for the other technologies primarily since the programming is done on a much lower level than the other technologies. For Java Card knowledge of GSM standards is desirable since e.g. the SM header must be known by the developer. If some previous knowledge of Java is known then it goes very quickly to get started to develop J2ME applets for a Java enabled phone. When it comes to the WIB there is also a very low learning threshold and thus goes quick to get started, since practically all that is required is some web server programming.

When it comes to choosing strategy there are a lot issues to consider, if ease of development is the most important aspect then the WIB is ideal. There is also short development time and learning threshold for the WIB. The largest benefit from using the Java Card compared to the WIB is that much more can be done locally and thus limiting the communication needed. If looking at security both the WIB and Java Card can offer a totally secure end-to-end communication including authentication and message integrity. The main threats that can be seen are WAP2.0 and the convergence between J2ME in the phone and the Java Card since then it will be possible to authenticate a user in quite an easy way using the SIM card. This possibility with communication between a midlet and Java Card could be used in many ways e.g. a game that only is allowed to be played for a number of times before more money must be paid here Java Card comes in as suitable place to store such a counter.

It feels like there is no future for SIM based applications at least if the communication is based on sending SMs since they are way to slow and non interactive. One important aspect to not forget is that the mobile operator doesn't just want to give away such a strategic point where he/she is in "full control" of the customer. Most beneficial from an operators perspective is to set up services as menu items in the phone that "lead" the user in the "right" direction. Now a days as mentioned in section *5 Future Work*, large operators are moving some stuff from the SIM to customized menus that reside on the phone this together with preinstalled configurations for GPRS have been made to increase the customers usage of different services.

It seems as the overall winner could be the WIB at least since it's goes very easy and quick to develop a service. One of the largest benefits of the WIB is that there is practically no complexity on the client side so large changes can be made without having to modify the client. In contrary if for example Java on the phone were used there is a large complexity also on the client side, which might make it difficult to do large changes.

To summarize, if the service requires interactivity or more than just text then Java on the phone has to be used today since the SIM cards evaluated doesn't support GPRS as transmission medium yet. On the other hand if it's enough with SM communication or if it's enough with a text based service also in future then it's enough looking at different versions of SIM based applications. A WIB with Java plug-in sounds very attractive since then it would be possible to benefit from both cases i.e. the possibilities to store data locally and the ease of development that is associated with the WIB. One other drawback with Java Card is that an application protocol must be made i.e. how to parse the information sent back and forth in the SMs. With the WIB that is taken care of by the WIG server that transform the SM request to an ordinary http request which relieves the programmer from a lot of extra burden. If just the Java Card and WIB are compared a decision must be made whether the extra work needed to develop for the Java Card is ok compared to the less communication that might be needed. When it comes to security all solutions using SIM cards offer the same thing with encryption and message authentication and the matter of choice doesn't really matter.

7 References

[CEL02] http://www.cellular.co.za/sim_toolkit.htm, June 2002

[CHE00] Zhiqun Chen, *Java Card™ Technology for Smart Cards: Architecture and Programmer's Guide* Addison Wesley, June 2000. Chapter 3 and 9 available online at. http://developer.java.sun.com/developer/Books/-consumerproducts/javacard/

[CON02] http://java.sun.com/docs/books/tutorial/java/concepts/index.html

[CMD00] *Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices – White Paper*; 2000 SUN; http://wireless.java.sun.com/getstart/articles/intro/

[DAY01] *Developing Wireless Applications using the Java 2 Platform, Micro Edition*; Bill Day; http://wireless.java.sun.com/gestart/articles/wirelessdev/wirelessdev.pdf; 2001 SUN

[EFF99] Effing Rankl, *Smart Card Handbook*, Second edition Wiley 1999. ISBN: 0-471-98875-8

[ENC02] http://wireless.java.sun.com/midp/ttips/dataencryp/

[G+02] A training manual for the programming of Java Cards from GemPlus.

[GSM02] http://www.gsmmobile.co.nz/Sim_Toolkit.htm

[GUT02] Guther, Scott & Cronin, Mary J, *Mobile Applications programming with SMS and SIM Toolkit.* McGraw-Hill Telecom, 2002 ISBN: 0-07-1337540-6

[IBM02] *WAP 2.0 Securing the Internet without wires.* Deborah Durham-Vichr & Kimberly Getgen available at: http://www-106.ibm.com/developer-works/library/wi-sectrends/?dwzone=wireless

[JAC02] Site: http://www.javacard.org/ June 2002

[JAV02] http://wireless.java.sun.com/getstart/

[JCS01] *Java Card Platform Security.* Sun Microsystem Technical White Paper available at: http://java.sun.com/products/javacard/JavaCardSecurity-WhitePaper.pdf

[JGU02] http://www.jguru.com/faq/ Java Card FAQ

[PEL02] Lindström, Peter *Evaluation of architectures for the development of secure mobile applications*, thesis report to be published.

[NYT02] Ny Teknik 30 Okt

[STA00] *Data & Computer Communications,* 6ed, William Stallings; 2000 ISBN 0-13-084370-9

[STI02] Smarttrust material for education regarding WIG/WIB

[STW02] http://www.smarttrust.com/

[TEL02] http://privatbutiken.telia.se/butiken/main/main.jsp?name=GPRS &type=Sida

[UST02] http://www.usatoday.com/life/cyber/tech/review/2001-06-05-crypto-phone.htm

[WAG02] *WIG Application Guidelines*, Smarttrust document available for WIG developers