Preface

This report documents the work I have done in my final master thesis at the Department of Telematics at NTNU. The assignment has been interesting, but also at times a bit frustrating. I am very satisfied that I have made an prototype in spite of all the problems that had to be overcome. Hopefully this report describes all my work in a satisfactory way.

I would like to thank Stig Frode Mjølsnes for inspiration and guidance, and Randi Flønes at the Department of Telematics for helping me with all practical details.

Thanks to all my friends who always kept faith in me. My education has finally come to an end.

NTNU, Trondheim, 2003-02-28

Vegard Steinsholt
Abstract

With e-business technology continuously being taken more into use and introduced in new markets the transition to m-business will make electronic transactions exceedingly popular. In the near future e-wallets and m-wallets will probably replace today’s leather wallets.

This master thesis proposes a software architecture of an online electronic wallet as described in [Ref: Online E-wallet, Mjølsnes] as well as an implementation prototype of an example application using Java technology.

The software architecture is founded on the requirement to use Java technology and is based on a client-server model. The online e-wallet acts as a client using a credential keeper and services as servers. To access a service the client application retrieves the credentials demanded by the service from its credential keeper and presents them to the service. The architecture is designed at a relatively high level because of the lacking definitions on exactly which services the system should be able to use.

The implementation prototype is adapted to the Personal Java library due to the choice of using the Jeode EVM as Java Virtual Machine. The client application has been deployed and run on an iPAQ 3870 PocketPC with a Jeode EVM, while the credential keeper and example service has been located on standard Windows PCs with regular JVMs. Communication has been done using Java RMI procedure calls over WLAN connections. The implemented prototype was successfully tested and proved to perform satisfactory. Symmetric cryptography has been used between Client and Credential Keeper, but the use of a SSL connection between the Client and the Service failed due to the use of Personal Java. There is also need for further studies on security issues.

However, the work that has been done is a basis for further improvement on design and prototype implementations. All implementations should then aim to use the J2ME library and run on J2ME enabled virtual machines to be able to use updated and improved technology compared to the outdated Personal Java.
Table of contents

1 INTRODUCTION 1
  1.1 ON-LINE E-WALLET WITH DECENTRALIZED KEEPERS 1
  1.2 THE ASSIGNMENT 2
  1.3 READER’S GUIDE 4

2 SYSTEM ARCHITECTURE 5
  2.1 REQUIREMENT SPECIFICATION 5
  2.2 OVERVIEW 7
  2.3 SYSTEM AND SOFTWARE DESIGN 8
    2.3.1 NOTATION 8
    2.3.2 SYSTEM DESIGN 9
    2.3.3 SOFTWARE DESIGN FOR CLIENT TERMINAL 10
    2.3.4 SOFTWARE DESIGN FOR CREDENTIAL KEEPER TERMINAL 13
    2.3.5 SOFTWARE DESIGN FOR SERVICE TERMINAL 14

3 IMPLEMENTATION PROTOTYPE 17
  3.1 HARDWARE AND MIDDLEWARE 17
  3.2 THE CLIENT TERMINAL 18
    3.2.1 WSDD AND J9 18
    3.2.2 JEODE 19
  3.3 COMMUNICATION TECHNOLOGY 20
    3.3.1 WLAN20
    3.3.2 BLUETOOTH 20
    3.3.3 JAVA RMI 21
  3.4 REVISED SYSTEM AND SOFTWARE ARCHITECTURE 22
  3.5 SCENARIO “SHOW CREDENTIALS” 24
    3.5.1 MESSAGE SEQUENCE DIAGRAM 24
  3.6 PACKAGE AND CLASS DIAGRAMS 25
    3.6.1 PACKAGE DIAGRAM 25
    3.6.2 CLASS DIAGRAM KEEPER 26
    3.6.3 CLASS DIAGRAM SERVICE 27
    3.6.4 CLASS DIAGRAM CRYPTO 27
    3.6.5 CLASS DIAGRAM CREDENTIALKEEPER 28
    3.6.6 CLASS DIAGRAM ACCESSSERVICE 29
    3.6.7 CLASS DIAGRAM CLIENT 30
  3.7 IMPLEMENTATION 30
    3.7.1 RUNNING THE APPLICATION 31

4 DISCUSSION OF RESULTS 33
<table>
<thead>
<tr>
<th></th>
<th>SOFTWARE ARCHITECTURE</th>
<th></th>
<th>IMPLEMENTATION PROTOTYPE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td></td>
<td>33</td>
<td></td>
<td>34</td>
</tr>
<tr>
<td>4.2</td>
<td>SOFTWARE ARCHITECTURE</td>
<td></td>
<td>IMPLEMENTATION PROTOTYPE</td>
<td></td>
</tr>
<tr>
<td>4.2.1</td>
<td>DATA SECURITY</td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2.2</td>
<td>PERFORMANCE</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2.3</td>
<td>SIMPLICITY</td>
<td>36</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5 CONCLUSION 38

6 OUTLOOK 40

7 REFERENCE LIBRARY 42

APPENDIX A
SYSTEM AND SOFTWARE DESIGNS

APPENDIX B
CLASS DIAGRAMS

APPENDIX C
IMPLEMENTED JAVA CODE

APPENDIX D
JEODE RUNTIME DOCUMENTATION

APPENDIX E
JAVA SECURITY POLICY FILE

Master Thesis at Department of Telematics, NTNU 2003
Vegard Steinsholt
Introduction

In 2003 e-business is in rapid growth a few years later than originally anticipated by the ICT-industry. The “.com”-breakdown has slowed things down, but the future seems brighter for companies in the e-business industry. Many companies are nowadays working with moving the e-business from wired to wireless communication, and thus introducing m-business (mobile business). The increased flexibility that is introduced is anticipated to boost the market to a new level. Major companies within the finance, creditcard, content providing and ICT industries invest a lot of money in developing new services and features, for example m-payment and m-wallets. Many of them have also seen the need for standards, and have therefore joined organizations such as the Global Mobile Commerce Interoperability Group, GMCIG [GMCIG], Mobile Electronic Transactions, MeT [MeT], and The Mobey Forum [MobeyForum].

**On-line e-wallet with decentralized keepers**

The proposed solutions and standards for m-wallets include mobile terminals with creditcard-slots (one or two), the creation of special, separate accounts on remote servers or using the SIM card in GSM mobile phones as identification and storing of other credentials. Stig Frode Mjølsnes has however seen the disadvantages of the different architectures and proposed an architecture of his own, called an on-line e-wallet with decentralized credential keepers [MjølsnesRong2003].

The on-line e-wallet system includes an e-wallet terminal and a credential keeper. The e-wallet terminal should be a mobile phone, a PDA (Personal Digital Assistant) or a dedicated hardware module. The credential keeper could be located on a standard PC or similar. Services to use can be all sorts of services a user might need, including shopping at web sites or access to a physical or electronic item. When a user needs his credentials, for example when paying for or getting access to a service, the e-wallet terminal is responsible for retrieving the credentials demanded for the service. Instead of having the credentials stored in the e-wallet terminal, the e-wallet retrieves the credentials from its corresponding credential keeper and presents them for the service application. This way the user can keep all his credentials physically safe, but still have access to them anywhere and anytime, as long as the e-wallet can get connected to its credential keeper.

A suggested realization architecture showing a user with a mobile device used as e-wallet and his credential keeper together with a physical and a web service is depicted in figure 1-1. Suggested security is also included in the figure.
The assignment

This thesis will propose a distributed software architecture for the on-line e-wallet with decentralized credentials as well as an implementation prototype for one possible application.

The software architecture should be designed generally so that applications can be implemented using different implementation techniques and languages. However, the assignment text specifies that Java technology should be used. This affects the generality of the architecture and design.

An application prototype will be implemented in Java. The e-wallet functionality will be located on a standard mobile platform while services and credential keeper functionality will run on standard operating systems. Using Java makes the applications portable because of the “Write once, run anywhere” property of Java applications, since any device with a JVM (Java Virtual Machine) installed can run the applications. Adding new applications or new features should also be easy by downloading them from trusted servers.

The application will implement a “Show credentials”-scenario, where a user uses his e-wallet to get access to a resource by showing demanded credentials. The credentials are retrieved by the e-wallet terminal from the credential keeper and presented to the service. If the credentials are
approved the service grants the user access to the resource, e.g. opens a door or similar.

This thesis does not address any design, architecture or methodology behind the service’s interface to the e-wallet. This includes how the service approves the presented credentials, how the service charges any user accounts or uses other resources to complete the service offered to the user.

The assignment text specifies that the proposal will consider available and proposed standards work. All standards found regarding e-wallets, such as in [GMCIG], [MeT] and [MobeyForum] address the communication scheme between the e-wallet and the service. Since the distinctive characteristic in the e-wallet system proposed in [MjølsnesRong2003] is the communication between the e-wallet and the credential keeper, the software architecture and the implementation prototype focus on this part. Considerations regarding these standards are therefore not included in this thesis.

The credential keeper must have some sort of access to all credentials the user’s e-wallet needs. This is not implemented in the prototype, but some suggestions to possible solutions on this part of the architecture are presented in the conclusion.

The security needed for wireless communication supplied by e.g. WLAN is considered to be outside the scope of the assignment. However, two other master theses are written at NTNU spring 2003, one considering the cryptographic protocols needed by the e-wallet and one considering further developments in the communication security.

**Reader’s guide**

The proposed system architecture with system and software design is presented in chapter 2. Chapter 3 gives the story of the equipment being used in the implementation prototype, and presents the prototype. Both system architecture and the implementation prototype are discussed briefly in chapter 4, while chapter 5 concludes on the work that has been done in this master thesis. Chapter 6 gives an outlook on further work on the online e-wallet architecture.
System architecture

Figure 2-1 shows the on-line e-wallet general architecture with multiple applications installed. The three devices are physically distributed.

Requirement specification

To implement the on-line e-wallet, Java technology will be employed. This implies procedure-based communication, which puts some restraints on the system. According to [BrækHaugen1993, 269] the following conditions must hold:

1. The [SDL] (sub) system must behave like a procedure call tree.
2. Only one process, the input process at the root of the tree, may wait to receive signals from the environment or perform time measurements, unless the waiting time is negligible.
3. For each signal sent down the tree, not more than one reply signal is returned to the sender (implemented as a procedure call return value).
4. There is sufficient time between each external input signal to process all signals and outputs that follow from it and to return to the input process. Consequently, there is no need to give pre-emptive priority to the input process.

Defining the e-wallet as a client and the service and credential keeper as servers, conditions 1, 2 and 3 will hold. The e-wallet client is then the root of a procedure call tree. If the service needs interaction with a distributed server of some sort during the transaction, this will reduce the performance. Since the e-wallet has to wait for this communication to finish to complete the service, condition 4 also holds.

Consequently the architecture can be designed using a client-server model and procedure calls for communication. If message-based communication were to be used, Java APIs similar to the Java Message Service API [JMS] for J2EE [J2EE] would have to be included. To keep the size and simplicity of the application and application environment at a minimum, this was not considered an option. From here on a client-
server architecture is chosen, and input and output in the design is considered taken care of by the procedure calls of Java.

The client should thus run on a Java enabled mobile phone or PDA with a standard platform for practical reasons. If commercial success is at hand, dedicated hardware modules such as the SmartWear e-wallet [SmartWear] depicted in figure 2-2 could be developed. In any case the device should be equipped with both short range and packet switched communication technology.

![SmartWear e-wallet](image)

**Figure 2-2: SmartWear e-wallet**

The credential keeper should run on a standard operating system, such as Windows, with a JVM to run Java applications. By using a standard operating system the deployment of the software on home computers is possible. Credentials stored on cards or similar credential holders must be accessible to the application. The credential keeper is connected to a packet switched network. The mentioned Residential Gateway in [MjølsnesRong2003] is considered a part of the network topology, and is therefore not included in the design.

The service could run on any operating system with a JVM and packet switched network or short-range radio communication enabled (the same communication technology as the client). How the service validates the credentials sent from the client or uses other resources to offer the complete service is as previously mentioned outside the scope of this thesis.

To protect the communication between the client and the credential keeper, encryption should be used. The client and credential keeper can share a common secret key in tamper resistant hardware (such as a smart card on the client device), and use this to encrypt and decrypt the credentials sent on the network. Public Key Cryptography may also be used.

The communication between the client and the service should use appropriate security mechanisms dependant on which communication technology is used. [MjølsnesRong2003] suggests Bluetooth Security Mode 3 if Bluetooth is used, and SSL/ TLS (Secure Sockets Layer/
Transport Layer Security) connections are well suited when using packet switched communication such as WLAN.

**Overview**

An overview of the suggested system architecture is depicted using blocks in figure 2-3.

![Figure 2-3: Overview of System Architecture](image)

The security between the client and credential keeper is taken care of by symmetric or public key encryption technology handled by Java applications. Security between the client and the service is handled by appropriate security and encryption techniques related to the communication technology used. [MjølsnesRong2003] suggests Security Mode 3 using Bluetooth peer-to-peer and SSL/ TLS using packet switched networks.

**System and software design**

The system and software diagrams are presented in the following with explanations. The diagrams can also be seen all together in appendix A.

**Notation**

The graphical notation used to visualize the system and software design, is the same being used in [BrækHaugen1993, 236-242]. The notation is based on SOON (SISU object-orientated notation), adapted to represent hardware and software structures. Since there is no SDL design developed for the system, the different hardware blocks do not implement any particular SDL system blocks. Still the description of the system in [MjølsnesRong2003] and the assignment should give adequate information to design the hardware and software.
This notation is chosen because it is suitable for distributed system designs and familiar to the author of this thesis. The system design is most likely self-explained, but to help interpret the software designs the graphical objects with explanations are depicted in figure 2-4.

**System design**

The system design is depicted in figure 2-5.
Each Client terminal is connected to its Credential Keeper with a packet switched communication network. The number of Clients and Credential Keepers are thus the same. The Client should be a mobile phone, a PDA or a dedicated hardware module such as the SmartWear electronic wallet. For early test implementations and prototyping, the dedicated hardware module is excluded. Deployment of the application on mobile phones and PDAs that people already use on a daily basis is probably preferred, since these two devices also most likely will be known and used as one device in near future.

Figure 2-5: Hardware Design
The Credential Keeper can be located on a standard PC platform or be an extension hardware module for connecting to a PC. In this way users can install the Credential Keeper at home together with his credentials. If commercial success is at hand, banks or other trusted third parties could keep dedicated hardware racks with many Credential Keepers for its customers in safe locations, much like today’s bank boxes.

The Service computer may be a dedicated hardware module such as a door access controller, a service offered directly on a server or included in a web site on the Internet. The Clients connect to Services using short-range communication or a packet switched network, possibly the same as the network used between the Client and Credential Keeper. This depends on which service is offered. For example will packet switched network be used for payment services on web sites, but for door access control direct peer-to-peer short-range communication will probably be preferred and most effective. The design aims to make no limitations to what kind of services can be offered and how they are offered.

Each of the terminals executes software given by the following software designs.

**Software design for Client terminal**

The software design for the Client terminal is depicted in figure 2-6.
The Client terminal is connected to two networks or communication technologies that may be the same actual networks, different networks with same technology or different networks and technologies. The communication with the Service may also as well be peer-to-peer not using a network.

One or more main applications are connected to Services using short-range communication or packet switched network technology. This communication must be secured with suitable security mechanisms for the used communication technology. SSL/ TLS is an example of a security protocol that is successfully used for e-business applications that may well be an appropriate security mechanism to use. All main applications use a Keeper Accessor to retrieve the home address of the Client’s corresponding Credential Keeper. This functionality is located...
outside the main applications since all implemented applications need it for contacting the Credential Keeper. The address to the credential keeper can be stored unsecured on the Client terminal, since encryption technology will protect the credentials that may be returned to a wrongful Client. This way it will also be easy for the user to modify the home address of his Credential Keeper, if it is moved or he has or needs access to several Credential Keepers.

For applications to communicate with the Credential Keeper, an agent called Keeper’s Agent is connected to a packet switched network. This agent will act as a proxy of the Client’s corresponding Credential Keeper and supply the credentials requested by the main applications. This way the main applications located on the Client should be able to communicate with the Credential Keeper as if they were physically located on the same device. The functionality of the Keeper’s Agent will then be the same as the functionality of the software located on the Credential Keeper.

To encrypt and decrypt the information exchanged between the Client and the Credential Keeper, an Encryptor/ Decryptor is included. The Encryptor/ Decryptor can use one specific crypto standard or give the main application a choice between several standards. Symmetric encryption with common secret key is the easiest method to use, but Public Key Cryptography is also an option. The secret crypto keys are stored in tamper resistant hardware, and can only be accessed by entering the correct PIN code or password. Distribution of keys is considered outside the scope of this design.

The Client only needs one instance of the Encryptor/ Decryptor to execute the encryption and decryption, since this functionality is common for all main applications to use. The system also only needs one Keeper’s Agent, but this agent must be able to retrieve all different credentials the applications demand. Thus the Keeper’s Agent may have to be changed when a new application is deployed to the client, if the application needs a new set of credentials that cannot be fetched by the old Keeper’s Agent. This of course requires that the Credential Keeper has the required credential set at hand. The Keeper’s Agent only needs to be updated if its corresponding Credential Keeper’s functionality is changed. This updating should aim to be accomplished dynamically using the standard communication network.

To help the Client to update the Keeper’s Agent and also to download and install any new applications a Download & Installation Manager is connected to the packet switched network. This feature must be carefully implemented to prevent
malicious applications or code to be deployed on the Client terminal.

All software is supported by a Java enabled operating system for mobile phones or PDAs.

Software design for Credential Keeper terminal

The software design for the Credential Keeper terminal is depicted in figure 2-7.

![Software Design for Credential Keeper terminal](image)

The Keeper Controller is the main application and is connected to a packet switched network. It is responsible for all communication with its corresponding Client and handles the incoming requests using the other entities deployed on the Credential Keeper terminal.

As in the Client software, an Encryptor/Decryptor is responsible for encryption and decryption of the credentials supplied to the client. Of course the same crypto standard and keys have to be used on Client and Credential Keeper to make communication possible. The request from the Client may be passed to the Credential Keeper in clear text, but the returned credentials must
of course be encrypted. If a PIN-code or seed is required to encrypt or decrypt the credentials by the Credential Keeper, this should be accessible all the time without need for any interaction. Access to the credentials on the Client is then accomplished by for example using a common symmetrical crypto key.

To retrieve the credentials from their storage location Credentials Accessors are included. The credentials can be stored in several forms, e.g. smart cards. For each different kind of storage, a specified Credentials Accessor must probably be implemented. The Credentials Accessors must have an interface adapted to the storage method of the credentials, and fetch the correct credentials according to an identifying input that the Keeper Controller has received from the Client. If input is needed to the storage items, e.g. if a smart card needs input to update some data, the Credentials Accessors should also be responsible for this task. The Keeper Controller must be updated according to which credentials and Credentials Accessors are deployed on the Credential Keeper.

A standard operating system with a JVM supports all software.

**Software design for Service terminal**

The software design for the Service terminal is depicted in figure 2-8.
Clients can use packet switched network or short-range communication to contact the Service terminal’s front-end applications called Service Controllers. A Service Controller may for example be located inside a dynamic web page or on dedicated hardware equipment such as door access controllers. To handle the different kinds of services offered, the Service Controllers must be specifically implemented.

A Service Controller must inform the Client requesting its service of which credentials are required for granting the Client access to the service and which communication scheme is necessary. This includes a unique identification of which credential suppliers are accepted. For example if the Service terminal is a payment service, it may inform Clients that it accepts Visa and Master Card, and that it requires card number and expiry date.

The Service Controllers use Validators to check the credentials presented by the Client for validation. Credentials can be
checked using Access Control Lists located on the Service Terminal, or the Validator may have to use some other back-end applications to check them. For this purpose the back-end applications may need to communicate with other distributed software, and are therefore connected to some unspecified network.

To fulfil the offered service, e.g. open doors, give access to some other resource or charge the client for some amount of money, the Service Controllers may use back-end applications. Depending on what kind of service is offered, distributed resources may be used via the unspecified network connection. In addition to fulfilling the service offered the Client is returned with information on how its service request is handled. In its simplest form this is either an acceptance or a non-acceptance. This way the Client is returned the initiative for requesting new services.

A standard operating system with a JVM supports all software. Back-end applications may of course run on other platforms, but this requires an extra communication interface between the different platforms.
Implementation prototype

Much work has been done in this master thesis to make an actual implementation prototype of a part of the system and software architecture. This included getting hold of equipment to use and getting the different equipment to communicate with each other.

**Hardware and middleware**

The online e-wallet is to be run on a wireless terminal, for obvious reasons as small and handy as possible. For an early test implementation like the one presented in this thesis, the assignment specifies a mobile phone or a PDA. This also eliminates the need for yet a device for the user to carry around, since most people today already use either a PDA or a mobile phone. By using Java technology the implementation can easily be moved to other platforms supplying a JVM. For devices with limited resources the JVMs used are small in size and need little computing power. They are called EVMs (Embedded Virtual Machine) or KVMs (Kilo Virtual Machine).

The Credential Keeper can be located on or attached to the user’s home computer, or a computer specially designed for this purpose. Since most home computers use Windows operating system, this is likely to be a suitable operating system for the test implementation. Again, using Java only limits the credential keeper to a computer supplying a JVM.

Services can be offered in many different forms; web pages, payment terminals, door terminals, database servers etc. The wireless communication can be handled directly (peer to peer) or via a network connection point. As earlier mentioned, only the Service’s interface is considered in this thesis. Thus a computer running a JVM on a Windows operating system is sufficient for experimenting with the architecture.

To make the test application available for as many as possible, all middleware and software tools needed to support the application should be freeware.

**The Client terminal**

Having defined the requirements for the Client terminal, the available equipment from the Department of Telematics at NTNU was considered. The only useful terminal available in October 2002 was Compaq’s iPAQ 3650. iPAQ Pocket PCs use Windows CE as operating system by default. Files are deployed to the iPAQ terminal from a PC using a USB (Universal Serial Bus) cradle.
Available freeware JVMs for iPAQ 3650 were then considered. JVMs running on other operating systems than Windows CE were regarded as less interesting, considering the need for wireless communication. Installing a new operating system was expected to possibly complicate the use of WLAN and/or Bluetooth connections from a Java application.

**WSDD and j9**

The first JVM evaluated was IBM’s j9 that is a part of the WebSphere Studio Device Developer, WSDD [WSDD]. WSDD is a relatively large development tool, where J2ME [J2ME] applications for embedded devices such as an iPAQ Pocket PC can be developed in a PC environment and deployed to the mobile device running the j9 JVM. WSDD costs 499$, but a free evaluation copy was downloaded and tested.

WSDD with j9 works fine, but it takes relatively long time to get used to due to its size and many possibilities. Deployment of the application to the mobile device is strongly integrated in the development tool, which makes the deployment process and application on the iPAQ less visible to the developer. Regarding speed the j9 performs satisfactory. To make Java RMI (Remote Method Invocation) communication possible using WSDD, an extra module has to be installed. This is called WebSphere Custom Environment [WCE]. This module adds more functionality, but also complicates the use of the development tool.

**Jeode**

Several web sites and forums where KVMs for iPAQ are considered recommend the Jeode EVM from Insignia Solutions [Jeode]. People working with Java and iPAQ at Telenor FoU in Trondheim also recommended this virtual machine by e-mail. Jeode is shipped together with iPAQ terminals, but only from model 3850 and later. To evaluate this JVM, an iPAQ 3870 terminal already in use was tracked down at the Department of Telematics and exchanged with the 3650.

Jeode supports all Personal Java 1.2 libraries, including optional packages [PersonalJava]. Personal Java was the first Java “micro
edition” technology and is basically Java 1.1.8 without some minor class files. Therefore many devices on market have been and are enabled with Personal Java, but from December 2002 Personal Java is no longer supported by Sun Microsystems. All applications should then be moved over to J2ME using an appropriate configuration and profile [J2ME]. Regarding the application example in this thesis the difference is not very important, as it is compliant with both technologies. For future development however, J2ME devices should be used since this technology is continuously improved.

Jeode is as earlier mentioned shipped together with the iPAQ 3870, and therefore free of charge and easy to install. To run Java applications one simply has to pack the implemented class-files in JAR-files (Java Archive file) [JAR] and copy them to the terminal together with any additional packages. A link file with specific references to needed JAR-files and the main class is deployed to initiate the application.

**Communication technology**

The proposed system architecture in [MjølsnesRong2003] does not decide what communication technologies to use, but suggests WLAN, UMTS or GPRS for connecting the e-wallet to the credential keeper, and WLAN or Bluetooth for connecting the e-wallet to the service. Given the use of the iPAQ terminal as client, UMTS and GPRS were not possible for the test application. The remaining technologies were WLAN and Bluetooth.

**WLAN**

WLANs are emerging almost everywhere these days, and will most likely continue its growth as the security improves. NTNU has installed several WLANs at campus, and the iPAQ is easily expanded with a WLAN adapter using a PC card expansion pack.

The iPAQ was expanded with a Cisco Aironet 352 Wireless LAN Adapter [CiscoAdapter]. To get connected, client adapter software for the iPAQ was also needed. This can be downloaded from Cisco’s website [AdapterSoftware].

With the WLAN card and software installed, the iPAQ should be easy to connect to a WLAN at campus. This actually proved to be a more difficult task than expected, and quite some time was used getting connected. The iPAQ needs a correct setup for connections, and the adapter needs the exact setup of the wanted WLAN to connect to. At most WLAN-zones at the NTNU campus a VPN-client (Virtual Private Network) is needed. Free VPN-clients for Windows CE are presumably not available at the time, and the use of VPN is also not relevant for the architecture in this assignment. A WLAN-zone with no need for VPN-clients was found in a computer lab, and after retrieving the correct
WEP-keys from the student who had set up the network, the iPAQ was connected successfully.

**Bluetooth**

In [MjølsnesRong2003] the suggested technology used between the Client terminal and the service is WLAN or Bluetooth. To evaluate both technologies and also to connect the Client and Service directly without using a network connection, Bluetooth was examined as an option.

To be able to connect the Client directly to a Service using Bluetooth, an implementation of a Java API for Bluetooth is necessary. The Java Community Process [JCP] is responsible for the development of Java technology. To continuously improve the technology Java Specification Requests (JSR) are reviewed and accepted or rejected by this open organization. The specification for the Java API for Bluetooth is named JSR-82 and had its final release in March 2002 [JSR-82].

Implementations of JSR-82 are offered by:

- Zucotto [Zucotto]
- Rococo [Rococo]
- Atinav [Atinav]
- Motorola [Motorola]

Zucotto’s implementation costs $2995, Rococo’s implementation is for Linux or PalmOS and Motorola’s implementation is shipped with Motorola devices. The Atinav implementation seems to be available as of February 2003. Unfortunately, it was not available in October 2002, and has therefore not been examined.

Given these problems developing a Java application using Bluetooth communication peer-to-peer, the decision was made together with the subject teacher to use WLAN for all communication. [JABWT] should give the latest information on the JSR-82 and implementations.

**Java RMI**

For communication between the client and the service and credential keeper, Java RMI was chosen. “RMI enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines” [JavaRMI].

When using Java RMI objects act as client and server applications. The server application must register in an RMI-registry located either on the same computer or another computer.
with an address reference to be available for clients to contact it. The client contacts the RMI-registry and obtains a reference to the server application object. With the reference obtained the client uses a proxy object called a stub to access the distributed server object’s available methods. At server side the server object uses a skeleton as proxy to the client. An overview of the communication scheme is seen in figure 3-2 and a simplified architecture is depicted in figure 3-3.

Figure 3-2: RMI Object Model

Figure 3-3: RMI Architecture

If the RMI client or server needs additional classes to fulfil its desired operations, these can be made available on web servers and automatically downloaded using DCL (Dynamic Code Loading). RMI technology uses serialization of the objects when passing them over a network.

Revised system and software architecture

The superior architecture for the system implementation prototype given the choices made is seen in figure 3-4.
Figure 3-4: Overview of Prototype

The client is a Compaq iPAQ 3870 with Jeode EVM using WLAN to connect to the Internet. The Credential Keeper and the Service is located on standard JVMs on standard Windows operating system connected to the Internet using Ethernet.

The block architecture can also be more specific according to the choices made regarding hardware, middleware and communication technology. The modified architecture is depicted in figure 3-5.

Scenario “Show Credentials”

The test implementation in this thesis simulates a scenario where the Client shows some specified credentials to a service. The imagined scenario is a user requesting to get access to a resource using credentials UserID and/or PINCode/Password.

Any communication with third parties, such as the Acceptor in figure [Ref: Figure MSCChart], is considered outside the scope of this
implementation example. The Credential Keeper’s retrieval of credentials from e.g. smart cards and the Download & Installation manager are also not implemented.

**Message Sequence Diagram**

The message sequence diagram for a simple Show Credentials-scenario is presented in figure 3-6.

![Message Sequence Chart for “Show Credentials”](image)

The diagram in figure 3-6 differs from the suggested message sequence in figure 7 in [MjølsnesRong2003] on some points. The establishment and release of sessions as well as the optional interaction are not included. The Credential Keeper is not contacted before the Client has retrieved which credentials are requested by the Service, but the retrieval and encryption/decryption of the credentials are added.

**Package and Class Diagrams**

In the following the package and class diagrams are introduced and briefly presented. They are also found in appendix B. The diagrams are designed in UML (Unified Modeling Language) using Microsoft Visio.

**Package diagram**

The superior package diagram is depicted in figure 3-7.
The system consists of six packages. Two of them, *keeper* and *service*, only contain an interface each that are implemented in package *credentialKeeper* and *accessService* respectively. This is done to make the Java RMI implementation orderly, since the client needs the RMI-server interface and the RMI-stubs of the implementations of the interface deployed on the terminal. The different packages are deployed to the different terminals according to table 3-1 with packages in columns and terminals in rows.

<table>
<thead>
<tr>
<th>keeper</th>
<th>credentialKeeper</th>
<th>client</th>
<th>crypto</th>
<th>service</th>
<th>accessService</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>Yes</td>
<td>No, but RMI-stub of main application</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>No, but RMI-stub of main application</td>
</tr>
<tr>
<td>Credential Keeper</td>
<td>Yes</td>
<td>Yes, including RMI-stub and skeleton of main application</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Service</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes, including RMI-stub and skeleton of main application</td>
</tr>
</tbody>
</table>

**Table 3-1: Deployment of Packages**

**Class diagram keeper**
The class diagram for the package *keeper* is depicted in figure 3-8.

<table>
<thead>
<tr>
<th>Interface</th>
<th>KeeperController</th>
</tr>
</thead>
<tbody>
<tr>
<td>+getUserID(String strCardID): String</td>
<td></td>
</tr>
<tr>
<td>+getPassword(String strCardID): String</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3-8: Class Diagram for Package keeper**

The interface *KeeperController* defines two methods; *getUserID* and *getPassword* that should return strings with the requested *UserID* and *Password/PINCode* respectively. The name *password* is regarded as a regular password or a PIN-code in this context. Both methods require a uniquely identifying string called *strCardID* as input parameter that identifies which organization or company the credentials must have been issued from.

**Class diagram service**

The class diagram for the package *service* is depicted in figure 3-9.

<table>
<thead>
<tr>
<th>Interface</th>
<th>ServiceController</th>
</tr>
</thead>
<tbody>
<tr>
<td>+getRequiredCredentials(): Vector</td>
<td></td>
</tr>
<tr>
<td>+showCredentials(String strUserID, String strPassword): boolean</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3-9: Class Diagram for Package service**

The interface *ServiceController* also defines two methods; *getRequiredCredentials* and *showCredentials*. Method *getRequiredCredentials* returns a Vector with specifications on which credential supplier is accepted and which credentials are needed for fulfilling the requested service. Method *showCredentials* accept strings *strUserID* and *strPassword* as input parameters. These will be validated and the method returns a boolean value indicating success or not.

**Class diagram crypto**

The class diagram for the package *crypto* is depicted in figure 3-10.
The package `crypto` is used by the Client and the Credential Keeper to encrypt and decrypt the credentials. The interface `Crypto` defines two methods; `encrypt` and `decrypt`. Both methods require a string as input parameter, in clear text and cipher text respectively, and return strings that are turned the other way around with the defined crypto standard. They also require a string `strPINCode` as input parameter to get access to the crypto key stored in tamper resistant hardware.

The class `DesCryp` implements the interface `Crypto` and uses DES (Data Encryption Standard) encryption. The constructor requires a boolean value as input parameter that indicates whether the object instance will be used for encryption (value `true`) or decryption (value `false`). The methods `encrypt` and `decrypt` defined in the interface are implemented, and attributes needed by the class to perform encryption/decryption, such as a `KeyParameter`, are defined.

**Class diagram credentialKeeper**

The class diagram for the package `credentialKeeper` is depicted in figure 3-11.
The package `credentialKeeper` consists of the class `MyKeeperController` that implements the interface `KeeperController` from package `keeper` and a simulated implementation of a `CredentialAccessor`.

The methods `getUserID` and `getPassword` in `CredentialController` are implementations of the previously explained methods in package `keeper`. These two methods use the identically named methods in class `FakeCredentialAccessor` to retrieve credentials. The method `main` is the executable method that is run when the application is started on the Credential Keeper terminal. For added flexibility when running the application this method should prompt for input to be set up properly.

The interface `CredentialAccessor` only defines the two methods `getUserID` and `getPassword`. This interface should be improved and extended as needed when other credentials are “installed” on the Credential Keeper terminal. The class `FakeCredentialAccessor` is only implemented to return some fake credentials, as the name implies.

**Class diagram accessService**

The class diagram for the package `accessService` is depicted in figure 3-12.

![Class Diagram for Package accessService](image)

In package `accessService` the class `AccessController` implements the interface `ServiceController` from package `service`. The implemented class is responsible for offering the service. The two methods defined in the interface, `getRequiredCredentials` and `showCredentials`, are implemented, but the `showCredentials` does not perform any validation on the credentials presented from the Client terminal.
The executable method `main` is run when the application is started. The Service terminal is then set up with the uniquely identifying `strCardID` to decide which issuer of credentials are valid, and a choice of requiring credentials `UserID` or `Password/PINCode` or both to grant access to the service offered. These values are stored in the defined attributes that have to be declared static.

**Class diagram client**

The class diagram for the package `client` is depicted in figure 3-13.

![Class Diagram for Package client](image)

The class `ShowCredentials` is the main class that initiates the whole system when executing the `main` method. The user is prompted to write the IP-address of the desired Service causing the Client to contact the given address. The Service returns its credential requirements, and the Client contacts its corresponding Credential Keeper for retrieving the required credentials. When the credentials are returned, the user is prompted to input a PIN-code to decrypt the credentials and present them to the Service thus completing the service request.

A simulated implementation of the interface `KeeperAccessor` is included to retrieve the home address of the Client’s Credential Keeper in form of a string using the method `getHomeIPAddress`.

**Implementation**

The implemented code of the Java classes and interfaces is found in appendix C and should speak for itself.

The Encryptor/ Decryptor is implemented using a lightweight cryptography API for Java supplied by BouncyCastle [BouncyCastle]. DES (Data Encryption Standard) cryptography is used in the prototype. This can easily be changed to any other desired encryption standard using the same API.

Unfortunately, Personal Java does not support using SSL on top of TCP/IP. This feature was introduced in Java™ 2 SDK, Standard Edition,
version 1.2 [RMISSL]. The communication between the Client and the Service is therefore not secured in this prototype.

**Running the application**

To test the implementation, the Java class files were deployed to the different terminals. The Credential Keeper Terminal needs the class files in packages `keeper` and `credentialKeeper`. Equally the Service Terminal needs the class files in packages `service` and `accessService`. On each terminal, an RMI-registry was started and the main classes run. It is important that the classpath on the computer is deleted when starting the RMI-registry.

The Service is initiated with input that identifies the kind of credentials required. A uniquely identifying `cardID` is mandatory to identify which supplier the required credentials must have been issued from, e.g. NTNU. The Service may then be set up demanding `userID` or `PINCode/Password` or both of them.

The Credential Keeper and the Client need access to a common DES crypto key stored on each terminal. This key must be instantiated and deployed before running the applications.

The client terminal needs the Java class files contained in the package `client`, RMI stub-files of the two interfaces `KeeperController` and `ServiceController` and packages `keeper` and `service`. All files are packed in JAR-files and deployed onto the client terminal. Additional classes needed from the BouncyCastle API also must be packed and deployed. To run the application using Jeode, a link-file was written and deployed to the iPAQ. The link contains an executable command with command-line parameters specifying classpath and main class to run. The file `ShowCredentials.lnk` contained the following:

```
18#"\Windows\evm.exe" -Djeode.evm.console.local.keep=TRUE -cp
\jeode\keeper.jar;\jeode\credentialKeeper.jar;\jeode\service.jar;\jeode\\
accessService.jar;\jeode\crypto.jar;\jeode\orgCrypto.jar;\jeode\client.jar client.ShowCredentials
```

`18#"\Windows\evm.exe"` specifies where the executable exe-file is located on the terminal. `-Djeode.evm.console.local.keep=TRUE` prevents the Jeode console window to shut down after application is finished. The `–cp` command is followed by the JAR-files needed in the classpath and `client.ShowCredentials` tells which class to run. Further information on syntax for the command-line parameters can be found in the Jeode Runtime documentation in appendix D.
With both Credential Keeper and Service terminals with RMI registries initiated with needed parameters, the running Client terminal can contact the Service by entering the Service’s IP-address. The Client then retrieves its corresponding Credential Keeper’s IP-address and requests the demanded credentials. The required credentials are returned encrypted and the user is prompted to enter an access code to decrypt them and complete the Service. When the Service receives the demanded credentials, they are checked for approval (not implemented) and the Client is approved or disapproved.
Discussion of results

This chapter discusses the proposed software architecture and the implementation prototype.

Software architecture

The proposed software architecture is designed with the requirement of implementing applications in Java. This is not the optimal way to design a system, since choice of implementation method and language rather should be made regarding a complete software design. The architecture may therefore have been influenced in a disproportionate way.

Still the architecture should be useful, regarding that Java technology probably will be used in the development of mobile e-wallets. J2ME will probably be a widely used platform for mobile devices and a master thesis written at the IT University of Copenhagen also states that J2ME technology is suitable for developing mobile payment systems if the manufacturers are willing to support it [Cervera2002].

The architecture is also designed at a rather high level of abstraction. This is due to the fact that services that should be offered in the system are not finally decided. Different services most likely require different credentials and different scenarios of communication between the different entities. For every defined service the client should be able to use, the set of credentials and communication scenario must be defined to see if a new kind of application on the client terminal is required. This complicates the generality and reuse of implementations.

The Service terminal is vaguely defined in the design, since most services probably will be implemented by many different organizations and companies in many different ways. This should be accounted for by modifying the applications on the Client terminal, and not the other way around.

An area that has not been taken care of in the architecture is a scenario where a third party used by the Service needs to interact with the Client to fulfil the requested service. The choice of designing the system as a client-server model unfortunately makes this integration a bit challenging.

The software architecture and design has been developed simultaneously with the implementation work. This may have influenced the design by taking implementation issues too much into account when designing the system. Making an actual implementation prototype work on a distributed platform has been significantly prioritized and taken a lot of time, causing the design to be less quality assured.
Implementation prototype

Quite a few practical obstacles were overcome to make the implementation work. An indication of the amount of problems that can be met using the technology in this thesis can be given from reading postings at Sun’s Java developers’ forums on J2ME, Personal Java and RMI [JavaForum]. Combining the technologies adds up the difficulty and introduces new troublesome areas.

Data security

An application handling credentials requires the highest possible level of security. The aspects considering the e-wallet implementation will now be discussed.

Communication

The communication between the Client and its corresponding Credential Keeper using symmetric cryptography should be easy to secure at an adequate level. Though the DES standard is proved to be cracked by the Electronic Frontier Foundation [EFFDES], using other stronger standards such as the AES (Advanced Encryption Standard) will strengthen the security sufficiently. A new cryptoprotocol already has been proposed by Mjølsnes inspired by the solution of the Kerberos authentication system. This can be further examined reading [Mjølsnes2003].

As previously mentioned SSL is not used for communication between the client and service in the prototype due to Personal Java. Using a KVM supporting J2ME however will probably make communication with SSL easy to implement using a custom RMI Socket Factory [RMISocket].

Java Security Policy

Using Java RMI, a Security Manager must be created and installed. This Security Manager protects access to system resources from untrusted downloaded code running within the virtual machine. All JVMs are equipped with a security policy, which is defined in a java.policy file. The policy file is normally to be found in the folder {jdk-root}/jre/lib/security. However using the Jeode EVM installed on an iPAQ 3870, the java.policy file is located in the folder Windows/lib/security.

To allow Java RMI clients and servers to connect to each other and the RMI-registry and possibly download code, these policy files must be altered. The altered policy file for the terminals can be found in appendix E.

Granting all the new permissions in these policy-files imply new security concerns. Allowing sockets to
connect to all unprivileged ports (from 1024 and up) may cause the terminal to open for malicious attacks. Access to reading files located on the terminal is not a relevant concern to a completed system, since these files are only faked substitutes for real storage of access codes etc. RMI-communication is handled at port 1099 as default, but can be altered to any other port. Because of these possible security flaws further studies should be made to optimize the policy files. The focus on making the communication work has unfortunately limited the considerations on possible security weaknesses introduced in the process.

Performance
The first WLAN where the application was executed was set up behind a relatively strict firewall using SSID (Service Set Identifier) and 128 bits WEP-keys (Wired Equivalent Privacy). When running the application in this WLAN, each RMI lookup took about 40 seconds to finish. Given the use of the application this was not satisfactory at all.

A workaround to the problem was found at the Sun’s Java Developer web pages [RMILookup]. When executing a RMI lookup contacting a given address, the Java class InetAddress in package java.net uses the networks DHCP-server (Dynamic Host Configuration Protocol) to perform a cross-reference between the IP-address and host name. This is done every time a new IP-address or host name is presented to the InetAddress class to prevent spoofing. The solution to this problem is to include host names and corresponding IP-addresses in the operating system’s hosts file.

The hosts file is not included in Windows CE, but an application giving the same functionality was found on the Internet [CEStuff]. Installing the application Pocket Hosts and entering the corresponding host names and IP-addresses used by the RMI-application improved the performance from 40 seconds to less than 2 seconds. This should be considered a satisfactory performance.

However, trying out the application on another similar WLAN that was set up quite open without SSID and WEP, the performance did not differ noticeably using Pocket Hosts. The RMI lookups took less than 2 seconds in both cases. The exact reason for the significant delay in the first WLAN is not found, but the degree of security is most likely the reason in some way.

The total performance of the application will most likely be improved by introducing short-range communication between the Client and the Service. The communication will then be
handled directly between the two terminals, abolishing the use of a network connection and DHCP-servers.

**Simplicity**

An implementation of an m-wallet needs simple functionality if commercial success is to be achieved. This prototype has no GUI (Graphical User Interface), since the user interface has been down prioritized due to time constraints. If the lack of GUI is disregarded, the most important issue related to functionality is how to connect to the desired service.

**Connecting to a service**

A consideration that came up in an early stage was how to initiate the contact with a desired Service. The user has to indicate which service he wants to use, especially when using short-range communication in an area offering several services such as a hallway with many doors. The service also has to indicate to the user that he in fact is connected to the service and not some other service or third party.

In the implementation prototype the user must enter the Service’s IP-address. This is time-consuming and not very user friendly. One suggestion is to offer the physical services via a web site where all services offered in a particular area can be contacted by only clicking on a link. Using applications to pay for services, web pages will presumably be used in any case. To increase the degree of user friendliness and present all services the same way, using web pages for all applications would be preferred. This also solves the problem regarding response from the Service, since the web page will confirm that the service is in fact contacted.
Conclusion

This master thesis only concludes on part of the total scope on the suggested architecture in [MjølsnesRong2003] but will be a foundation for further work by Mjølsnes and the other previously mentioned master theses.

The software architecture and implementation design is completed. The design is general and is a foundation for further development and test implementations on different parts of it. Since the major difference in this architecture from other e-wallets is the communication between the client and the credential keeper, integration into any proposed standard for handling communication between the Client and the Service should be easy to implement. It is also likely to believe that Java applications and SSL/TLS connections will be suggested by many standards and implemented by services offered to e-wallets.

The implemented prototype application works fine and has good performance. That implies that the technology used is an option for further development of the design. The lack of GUI and general user friendliness makes a rather poor first impression, but this is easily improved and integrated.

The application is implemented considering a service requesting access to some resource using credentials UserID and PINCode/Password. By adding a parameter from the service describing an amount of money to charge the user, the application can be used for payment transactions. The parameters UserID and PINCode/Password can easily be used to present e.g. Visa card number and expiry date.

The limited access to equipment to use at the university and the skilled personnel on the accessible equipment has limited the implementation part of the work. Making the terminals and software work and communicate with each other consumed time that could have been spent on other areas. The time spent on these problems is hard to communicate, as it is integrated and hidden in Java code and setup of equipment.

Using Jeode EVM as virtual machine on the client terminal is most likely not optimal, due to the use of Personal Java instead of J2ME. Several problems could have been avoided using a KVM supporting J2ME, since this technology is continuously improved and the choice of most developers. J2ME is always compatible with Java applications running on other computers, and supports the use of SSL and “state-of-the-art” technology.
Outlook

The first major improvement of the implementation is to move the communication between the client and the service from WLAN to Bluetooth. This will improve the mobility since no network is needed and probably also improve the performance. Bluetooth technology is in rapid growth and most mobile devices sold today are Bluetooth enabled. Also by deploying the Client application to a terminal (PDA or mobile phone) with built-in support for J2ME, Bluetooth and some packet switched technology, the development will probably be easier and the usage more realistic.

To further improve performance of the application, caching of retrieved credentials on the Client terminal should be examined. Concurrent communication or at least lookups on both Credential Keeper and Service simultaneously can also be tried out. This can possibly be implemented using Java threads.

As mentioned earlier there is ongoing work considering the cryptographic protocols for the online e-wallet. This is clearly an area that must be looked into to make the architecture mature for any real implementation. Moving from Personal Java to J2ME solves the problem of using SSL/TLS for the RMI procedure calls, and should be tried out. The security concerns regarding the use of Java application as an m-wallet in any given virtual machine is also important, as this is not thoroughly investigated in this thesis.

Another area to investigate is trying out SD (Secure Digital) memory card for tamper resistant storage of protected information on the iPAQ, such as crypto-key. This would then be similar to the use of SIM cards in GSM mobile phones today.

The credentials in this thesis are all simulated. The storage of the credentials and the Credential Keeper application’s retrieval of them are important issues. The immediate thought is credit cards with magnetic stripes or smart cards that are placed in a card reader with many slots. Java also has technology for this kind of applications, called JavaCard [JavaCard]. Another suggestion is to locate the credentials on small USB memory devices connected to the computer in an USB-rack.
Reference library

[AdapterSoftware] Cisco Client Adapter Software;
http://www.cisco.com/cgi-bin/Software/Tablebuild/doftp.pl?
ftpfile=pub/wireless/aironet/utilities/ce/30/WinCE30-PCMCIA-LMC-v222.exe

[Atinav] aveLink Bluetooth Protocol Stack in JAVA for J2ME;
http://www.avelink.com/modules/bt_javaj2me.htm

[BouncyCastle] Legions of the Bouncy Castle;
http://www.bouncycastle.org

[BrækHaugen1993] Bræk, Rolv; Haugen, Øystein;
Engineering Real Time Systems;
Prentice Hall Europe, 1993

[Cervera2002] Cervera, Anders;
Analysis of J2ME for developing Mobile Payment Systems;
IT University of Copenhagen, 01.08.2002

[CEStuff] Marc Zimmermann’s Windows CE Software;
http://www.zimac.de/cestuff.htm

[CiscoAdapter] Cisco Aironet 352 Wireless LAN Adapter;
products_data_sheet09186a0080088828.html

[EFFDES] EFF DES Cracker Project;
http://www.eff.org/descracker.html

[GMCIG] Global Mobile Commerce Interoperability Group;
http://www.gmcig.org

http://java.sun.com/j2ee/

http://java.sun.com/j2me/

[JABWT] Java APIs for Bluetooth Wireless Technology;
http://www.jabwt.com/

[JAR] The Java Archive (JAR) File Format;
http://developer.java.sun.com/developer/Books/
javaprogramming/JAR/

[JavaCard] Java Card Technology;  
http://java.sun.com/products/javacard/

[JavaForum] Java Technology Forums;  
http://forum.java.sun.com/

[JavaRMI] Java Remote Method Invocation (RMI);  
http://java.sun.com/products/jdk/rmi/

[JCP] Java Community Process;  
http://www.jcp.org

[Jeode] Insignia Jeode PDA Edition VM;  
http://www.insignia.com/content/products/pda.shtml

[JMS] Java Message Service API;  
http://java.sun.com/products/jms/

[JSR-82] Java APIs for Bluetooth;  

[MeT] Mobile Electronic Transactions;  
http://www.mobiletransactions.org

[Mjølsnes2003] Mjølsnes, Stig Frode;  
A cryptoprotocol securing keeper communication or The removal of an Agent;  
NTNU, 13.02.2003

[MjølsnesRong2003] Mjølsnes, Stig Frode; Rong, Chunming;  
On-Line E-Wallet System with Decentralized Credential Keepers; Mobile Networks and Applications 8, 2003

[MobeyForum] The Mobey Forum;  
http://www.mobeyforum.org

[Motorola] Java APIs for Bluetooth;  
http://e-www.motorola.com/webapp/sps/site/taxonomy.jsp? 
nodeId=03GGqrsDZxwK1dm1Y6

[PersonalJava] Personal Java Application Environment;  
http://java.sun.com/products/personaljava/

[RMILookup] Advanced Programming for the Java 2 Platform, Chapter 4:  
Lookup Services;  
http://developer.java.sun.com/developer/onlineTraining/ 
Programming/ JDCBook/lookup.html
[RMISocket] Using a custom RMI Socket Factory;
http://java.sun.com/j2se/1.4.1/docs/guide/rmi/socketfactory/index.html

[RMISSL] Using RMI with SSL;
http://java.sun.com/j2se/1.4/docs/guide/rmi/socketfactory/SSLInfo.html

[Rococo] Impronto Developer Kit;
http://www.rococosoft.com/devcorner/index.html

[SmartWear] SmartWear, home of the electronic wallet;
http://www.smartwear.com/

[WCE] WebSphere Custom Environment;

[WSDD] WebSphere Studio Device Developer;

Appendix A: System and Software designs

HS TYPE On-Line M-wallet System

1. Client
   - PDA or Mobile Phone
   - (1:n) Client

2. Credential Keeper
   - Keeper Computer
   - (1:n) Credential Keeper

3. Service
   - Service Computer
   - (+) Service

Packet Switched Network or Short Range Communication

<1> = Executes SS Type Client Terminal
<2> = Executes SS Type Credential Keeper Terminal
<3> = Executes SS Type Service Terminal

Unspecified Network
Software Architecture for On-line Electronic Wallet

SS Type Service Terminal

(+) : Validator

(+) : Service Controller

Packet Switched Network or Short Range Communication

(+) : Access Control List

(+) : Back End Applications

< 1 >

Unspecified Network

Operating System with JVM

< 1 > = All necessary back-end applications with network connections the service needs to fulfill the service offered
Appendix B: Class Diagrams

Package Diagram

Class Diagram for package keeper

```
public interface KeeperController {
    String getUserID(String strCardID); // String
    String getPassword(String strCardID); // String
}
```

Class Diagram for package service

```
public interface ServiceController {
    Vector getRequiredCredentials(); // Vector
    boolean showCredentials(String strUserID, String strPassword); // boolean
}
```
### Class Diagram for package crypto

**Crypto**

- **Encrypted**
  - `String` `strClearText`, `String` `strPINCode`
  - `encrypt(String strClearText, String strPINCode) : String`
  - `decrypt(String strCipherText, String strPINCode) : String`

  - implemented by
  - implements

**DesCryp**

- `cipher : PaddedBufferedBlockCipher`
- `key : KeyParameter`
- `inText : byte[]`
- `outText : byte[]`
- `type : boolean`

  + `DesCryp(boolean type)()`
  + `encrypt(String strClearText, String strPINCode) : String`
  + `decrypt(String strCipherText, String strPINCode) : String`

### Class Diagram for package credentialKeeper

**keeper: KeeperController**

- `interface` `keeper: KeeperController`

  - implemented by
  - implements

**MyKeeperController**

- `void` `getUserID(String strCardID) : String`
- `void` `getPassword(String strCardID) : String`

**FakeCredentialAccessor**

- `void` `getUserID(String strCardID) : String`
- `void` `getPassword(String strCardID) : String`
**Class Diagram for package accessService**

```
<<interface>>
service.ServiceController

1 -implemented by
1 -implements

AccessController
-arCardID : String
-binUserIDRequired : boolean
-binPasswordRequired : boolean
+getRequiredCredentials() : Vector
+showCredentials() : boolean
+main() : void
```

**Class Diagram for package client**

```
<<interface>>
KeeperAccessor
+getHomeIPAddress() : String

1 -is implemented by
1 -implements

ShowCredentials
+main() : void

FakeKeeperAccessor
+FakeHomeIPAddress() : String

ShowCredentials -uses -is used by

```
Appendix C: Implemented Java code

```java
package keeper;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 * Interface that must be implemented when making a Keeper Controller
 * @author Vegard Steinsholt
 */
public interface KeeperController extends Remote {

    /**
     * For retrieval of User ID
     * @param strCardID the ID uniquely identifying the card
     * @return a String containing the user's user ID for the given card ID
     */
    String getUserID(String strCardID) throws RemoteException;

    /**
     * For retrieval of password or PIN Code
     * @param strCardID the ID uniquely identifying the card
     * @return a String containing the password or PIN code for the given card ID
     */
    String getPassword(String strCardID) throws RemoteException;
}
```
Software Architecture for On-line Electronic Wallet

**Package service**

**ServiceController.java**

```java
package service;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Vector;

/**
 * @author Vegard Steinsholt
 */
public interface ServiceController extends Remote {

    /**
     * Used by client to get required credentials needed for
     * access
     * Card ID is mandatory to identify what card to fetch
     * credentials
     * from
     * @return a Vector defining the demanded credentials
     */
    public Vector getRequiredCredentials() throws RemoteException;

    /**
     * Used by client to show required credentials to obtain
     * access
     * @param strUserID a String with the user's ID
     * @param strPassword a String with the user's password or
     * PINCode
     * @return a boolean indication approval
     */
    public boolean showCredentials(String strUserID, String strPassword) throws RemoteException;
}
```

Master Thesis at Department of Telematics, NTNU 2003
Vegard Steinsholt
package crypto

package crypto;

/**
 * An interface to implement for Encryption/ Decryption
classes
 *
 * @author Vegard Steinsholt
 *
 */
public interface Crypto {

/**
 * Encrypts a string
 *
 * @param strClearText a String to encrypt
 * @param strPINcode the code to access the crypto key
 * @return an encrypted String
 *
 */
public String enCrypt(String strClearText, String

strPINCode);

/**
 * Decrypts a string
 *
 * @param strCipherText a String to decrypt
 * @param strPINCode the code to access the crypto key
 * @return a decrypted String
 *
 */
public String deCrypt(String strCipherText, String

strPINCode);
}

DesCryp.java

package crypto;

import java.io.*;
import org.bouncycastle.util.encoders.*;
import org.bouncycastle.crypto.*;
import org.bouncycastle.crypto.engines.*;
import org.bouncycastle.crypto.modes.*;
import org.bouncycastle.crypto.paddings.PaddedBufferedBlockCipher;
import org.bouncycastle.crypto.params.*;

/**
 * Responsible for encryption and decryption of credentials using
 DES
 *
 * @author Vegard Steinsholt
 */
public class DesCryp implements Crypto{

private PaddedBufferedBlockCipher cipher;
private KeyParameter key;
private byte[] inText;
private byte[] outText;
private boolean type;

/**
 * Constructor for instantiating a DES encryptor or
decryptor
 *
 * @param blnType true for encryption; false for decryption
 *
 */
public DesCryp (boolean blnType) {

try {
    // read the key, and decode from hex
    encoding
    BufferedInputStream keystream = new
    FileInputStream("\\\resources\\deskey.dat");
    int len = keystream.available();
    byte[] keyhex = new byte[len];

    //...
Software Architecture for On-line Electronic Wallet

```java
keystream.read(keyhex, 0, len);
byte[] inkey = Hex.decode(keyhex);
this.key = new KeyParameter(inkey);
}
catch (IOException ioe)
{
    System.err.println("Decryption key file not found, "+
    "or not valid keyfile");
    System.exit(1);
}
cipher = new PaddedBufferedBlockCipher
{new CBCBlockCipher(new DESEngine());
this.type = blnType;
}
/**
 * Encrypts a string
 *
 * @param strClearText a String to encrypt
 * @param strPINCode the code to access the crypto key
 * @return an encrypted String
 */
public String enCrypt(String strClearText, String strPINCode)
    throws SecurityException {
    String strAccessCode = new String();
    try {
        BufferedReader reader = new BufferedReader
        (new FileReader("\resources\AccessCode.txt");
        strAccessCode = reader.readLine();
    } catch (IOException e) {
        System.err.println("Valid Access Code not readable");
        System.exit(0);
    } if (!strAccessCode.equals(strPINCode)){
        SecurityException e = new SecurityException();
        throw e;
    }
    cipher.init(this.type, this.key);
    this.inText = strClearText.getBytes();
    int outSize = cipher.getOutputSize(this.inText.length);
    byte[] outTemp = new byte[outSize];
    int oLen = cipher.processBytes
    (this.inText, 0, this.inText.length,
    outTemp, 0);
    try {
        cipher.doFinal(outTemp, oLen);
    } catch (CryptoException ce) {
        System.out.println("Encrypt exception: "+
        ce.toString());
    }
    outText = Hex.encode(outTemp);
    return new String(outText);
}
/**
 * Decrypts a string
 *
 * @param strCipherText a String to decrypt
 * @param strPINCode the code to access the crypto key
 * @return a decrypted String
 */
public String deCrypt(String strCipherText, String strPINCode)
    throws SecurityException {
    String strAccessCode = new String();
    try {
        BufferedReader reader = new BufferedReader
```
new FileReader("\resources\AccessCode.txt");
strAccessCode = reader.readLine();
} catch (IOException e) {
    System.err.println("Valid Access Code not readable");
    System.exit(0);
} if (!strAccessCode.equals(strPINCode)) {
    SecurityException e = new SecurityException();
    throw e;
}
cipher.init(this.type, this.key);
this.inText = Hex.decode(strCipherText.getBytes());
int outSize = cipher.getOutputSize(this.inText.length);
byte[] outTemp = new byte[outSize];
int oLen = cipher.processBytes(
    this.inText, 0,
    this.inText.length,
    outTemp, 0);
try {
    cipher.doFinal(outTemp, oLen);
} catch (CryptoException ce) {
    System.err.println("Decrypt exception: "+ ce.toString());
} return new String(outTemp);
Software Architecture for On-line Electronic Wallet

Package credentialKeeper

MyKeeperController.java

```java
package credentialKeeper;

import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
import crypto.*;
import keeper.*;

/**
 * The class controlling the credential keeper
 * @author Vegard Steinsholt
 */
public class MyKeeperController extends UnicastRemoteObject implements KeeperController{
    private static String strAccessCode;

    /**
     * Default constructor
     */
    public MyKeeperController() throws RemoteException{
        super();
    }

    /**
     * For retrieval of User ID
     * @param strCardID the ID uniquely identifying the card
     * @return a String containing the user's user ID for the given card ID
     */
    public String getUserID(String strCardID){
        System.out.println( "Inside getUserID" );
        //Must be changed to other instance of CardKeeper, if implemented
        CredentialAccessor cardKeeper =
            new FakeCredentialAccessor();
        String strUserID = cardKeeper.getUserID(strCardID);
        System.out.println( "User ID is " + strUserID);
        //encrypt the string before returning it
        DesCryp encrypter =
            new DesCryp(true);
        String strEncryptedUserID =
            new String();
        try {
            strEncryptedUserID = encrypter.enCrypt(strUserID,
                        strAccessCode);
        }catch (SecurityException e){
            System.out.println("Code not accepted!");
            System.exit(0);
        }
        System.out.println( "Encrypted user ID is " + strEncryptedUserID);
        System.out.println( "getUserID finished" );
        return strEncryptedUserID;
    }

    /**
     * For retrieval of password or PIN Code
     * @param strCardID the ID uniquely identifying the card
     * @return an encrypted String containing the password or PIN code for the given card ID
     */
    public String getPassword(String strCardID){
        System.out.println("Inside getPassword");
    }
```
Software Architecture for On-line Electronic Wallet

// Must be changed to other instance of CardKeeper
// if implemented
CredentialAccessor cardKeeper = new FakeCredentialAccessor();
String strPassword = cardKeeper.getPassword(strCardID);

// encrypt the string before returning it
DesCrypt encrypter = new DesCrypt(true);
System.out.println("Password is " + strPassword);
String strEncryptedPassword = new String();
try {
    strEncryptedPassword = encrypter.encrypt(strPassword,
strAccessCode);
} catch (SecurityException e) {
    System.out.println("Code not accepted!");
    System.exit(0);
}

System.out.println("Encrypted password is " + strEncryptedPassword);
System.out.println("getPassword finished");
return strEncryptedPassword;

/**
 *  Main  method  for  running  the  credential  keeper
 */
public static void main(String[] args) {
    if (System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }

    System.out.println("Please enter IP-address of this
machine:");
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    String strIPaddress = null;
    try {
        strIPaddress = reader.readLine();
    } catch (IOException e) {
        System.out.println("Could not read input; please start
application again");
e.printStackTrace();
System.exit(0);
    }

    System.out.println("IP to bind is : " + strIPaddress);
    try {
        KeeperController credentialControl = new
MyKeeperController();
    } catch (IOException e) {
        System.out.println("Could not read input; please start
application again");
e.printStackTrace();
System.exit(0);
    }
}

String name = "/// + strIPaddress + "/KeeperController";
System.out.println("IP to bind is : " + strIPaddress);
try {
    KeeperController credentialControl = new
MyKeeperController();
}
Naming.rebind(name, credentialControl);  
System.out.println("Credential Controller bound to registry");  
}  
catch (Exception e) {  
  System.err.println("Exception is: " + e.getMessage());  
e.printStackTrace();  
}
}

CredentialAccessor.java

package credentialKeeper;

/**  
* Interface that must be implemented when creating a Credential Accessor  
*  
* @author Vegard Steinsholt  
*/
public interface CredentialAccessor {

    /**  
     * For retrieval of User ID  
     *  
     * @param strCardID the ID uniquely identifying the card  
     * @return a String containing the user's user ID for the given card ID  
     */
    public String getUserID(String strCardID);

    /**  
     * For retrieval of password or PIN Code  
     *  
     * @param strCardID the ID uniquely identifying the card  
     * @return a String containing the password or PIN code for the given card ID  
     */
    public String getPassword(String strCardID);
}

FakeCredentialAccessor.java

package credentialKeeper;

/**  
* Fake implementation of a Credential Accessor  
*  
* @author Vegard Steinsholt  
*/
public class FakeCredentialAccessor implements CredentialAccessor {

    /**  
     * Default constructor  
     */
    public FakeCredentialAccessor() {
        super();
    }

    /**  
     * For retrieval of User ID  
     *  
     * @param strCardID the ID uniquely identifying the card  
     * @return a String containing the user's user ID for the given card ID  
     */
    public String getUserID(String strCardID) {
        String strUserName = null;
        if (strCardID != null) {
            strUserName = new String();
            strUserName = "steinsho";
        }  
        //obtain the string for the correct card  
        return strUserName;  
    }
}


public String getPassword(String strCardID) {
    String strPassword = new String();
    if (strCardID != null) {
        strPassword = "5678";
    }
    //obtain the string for the correct card
    return strPassword;
}
**Package accessService**

**AccessController.java**

```java
package accessService;

import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;
import service. *

/**
 * Implementation of an Access Controller.
 * Offers front end methods for client to use to get access
 * Back end is not considered
 *
 * @author Vegard Steinsholt
 */
public class AccessController extends UnicastRemoteObject implements ServiceController{

/**
 * The identificator of the card, e.g. VISA, NTNU
 */
private static String strCardID = new String();

/**
 * Decides whether User ID is required
 */
private static boolean blnUserIDRequired = true;

/**
 * Decides whether Password/ PIN Code is required
 */
private static boolean blnPasswordRequired = true;

/**
 * Default constructor
 * @author Vegard Steinsholt
 */
public AccessController() throws RemoteException{
  super();
}

/**
 * Used by client to get required credentials needed
 * for access
 * Card ID is mandatory to identify what card to fetch
 * credentials from
 *
 * @return a Vector containing strCardID (position 0),
 * blnUserIDRequired (position 1) and
 * blnPasswordRequired (position 2)
 */
public Vector getRequiredCredentials(){
  System.out.println("Inside getRequiredCredentials");
  Vector vtrDemands = new Vector();
  vtrDemands.insertElementAt(this.strCardID,0);
  vtrDemands.insertElementAt(new Boolean(this.blnUserIDRequired),1);
  vtrDemands.insertElementAt(new Boolean(this.blnPasswordRequired),2);
  System.out.println("Contact finished");
  return vtrDemands;
}

/**
 * Used by client to show required credentials to
 * obtain access
 *
 * @param strUserID a String with the user's ID
 * @param strPassword a String with the user's
 * password or PINCode
 * @return a boolean indication approval
 */
public boolean showCredentials(String strUserID, String strPassword){
  System.out.println("Inside showCredentials");
  return false;
}
}
```
boolean blnApproved = false;

//Check if user is approved
if (blnUserIDRequired && blnPasswordRequired) {
  if (strUserID != null && strPassword != null)
    blnApproved = true;
}
if (blnUserIDRequired && !blnPasswordRequired) {
  if (strUserID != null)
    blnApproved = true;
}
if (!blnUserIDRequired && blnPasswordRequired) {
  if (strPassword != null)
    blnApproved = true;
}

//Grant physical access
System.out.println("ShowCredentials finished");
return blnApproved;

/**
* The main method run to start an AccessController
* Requires input from the user to set up the controller
* correctly
*/

public static void main(String[] args) {
  if (System.getSecurityManager() == null) {
    System.setSecurityManager(new RMISecurityManager());
  }
  System.out.println("Please enter IP-address of this machine:");
  BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
  String strIPaddress = new String();
  try {
    strIPaddress = reader.readLine();
  } catch (IOException e) {
    System.out.println("Could not read input; please start application again");
    e.printStackTrace();
    System.exit(0);
  }
  System.out.println("Input was " + strIPaddress);
  System.out.println("Please enter the card ID for this access controller:");
  reader = new BufferedReader(new InputStreamReader(System.in));
  try {
    strCardID = reader.readLine();
  } catch (IOException e) {
    System.out.println("Could not read input; please start application again");
    e.printStackTrace();
    System.exit(0);
  }
  System.out.println("Is user ID required for valid entry? (y/n)");
  reader = new BufferedReader(new InputStreamReader(System.in));
String strYorN = new String();
try {
    strYorN = reader.readLine();
} catch (IOException e) {
    System.out.println("Could not read input; please start application again");
    e.printStackTrace();
    System.exit(0);
}

if (strYorN.equals("n")) {
    blnUserIDRequired = false;
}

System.out.println("Is PIN code required for valid entry? (y/n)");
reader = new BufferedReader(new InputStreamReader(System.in));
strYorN = new String();
try {
    strYorN = reader.readLine();
} catch (IOException e) {
    System.out.println("Could not read input; please start application again");
    e.printStackTrace();
    System.exit(0);
}

if (strYorN.equals("n")) {
    blnPasswordRequired = false;
}

//Bind to registry
String name = "/" + strIPaddress + "/ServiceController";
System.out.println("IP to bind is: " + strIPaddress);
try {
    ServiceController accessControl = new AccessController();
    Naming.rebind(name, accessControl);
    System.out.println("Access Controller bound to registry");
} catch (Exception e) {
    System.err.println("Exception is: " + e.getMessage());
    e.printStackTrace();
    }
}
ShowCredentials.java

```java
package client;

import java.rmi.*;
import java.util.Vector;
import java.io.*;
import crypto.*;
import service.*;
import keeper.*;

/**
 * Main class for running the client application
 * User must enter IP address of the desired service
 * and access code to decrypt
 * the credentials demanded from the service
 * 
 * @author Vegard Steinsholt
 */

public class ShowCredentials {

    /**
     * The main method running the application
     */
    public static void main(String[] args) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        System.out.println("Please enter IP-address of service:");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String strIPaddress = null;
        try {
            strIPaddress = reader.readLine();
        } catch (IOException e) {
            System.out.println("Could not read input; please start application again");
            e.printStackTrace();
            System.exit(0);
        }
        System.out.println("Input IP address was "+strIPaddress);
        String strServiceName = "/" + strIPaddress + 
        "/ServiceController";
        Vector vtrDemands = new Vector();
        ServiceController serviceController = null;
        try {
            System.out.println("Contacting service... Please wait...");
            serviceController = (ServiceController) Naming.lookup(strServiceName);
            System.out.println("Service contacted, getting required credentials");
            vtrDemands = serviceController.getRequiredCredentials();
        } catch (Exception e) {
            System.err.println("RMI exception: " + e.getMessage());
            e.printStackTrace();
        }
        if (vtrDemands == null) {
            System.out.println("The service didn't return any demands");
            System.exit(0);
        }
        System.out.println("Required credentials for CardID = " + vtrDemands.elementAt(0));
    }
}
```
Software Architecture for On-line Electronic Wallet

+ "\n UserID required : " + vtrDemands.elementAt(1) + "\n Password/PINcode required : " + vtrDemands.elementAt(2));

// Retrieve demands
// CardID is mandatory
String strCardID = new String("");
Boolean blnUserIDRequired = new Boolean(false);
Boolean blnPasswordRequired = new Boolean(false);
strCardID = (String) vtrDemands.elementAt(0);
blnUserIDRequired = (Boolean) vtrDemands.elementAt(1);
blnPasswordRequired = (Boolean) vtrDemands.elementAt(2);

// Check that mandatory card ID is returned
// Must be checked in this way to avoid
// NullPointerException when strCardID=""
if (strCardID == null){
    System.err.println("Keeper didn't return CardID");
    System.exit(0);
}
else {

    if (strCardID.equals("")){
        System.err.println("Keeper didn't return CardID");
        System.exit(0);
    }

    KeeperAccessor kpaKeeperAccess = new FakeKeeperAccessor();
    //Get home IP-address
    String strHomeIPAddress = new String();
    try{
        strHomeIPAddress = kpaKeeperAccess.getHomeIPAddress();
    } catch (FileNotFoundException e){
        System.out.println("Home IP address info not found" + e.getMessage());
        e.printStackTrace();
    } catch (IOException e){
        System.out.println("IO exception" + e.getMessage());
        e.printStackTrace();
    }
    System.out.println("Home IPAddress found - " + strHomeIPAddress);

    //Contact credential keeper
    String strHomeName = "" + strHomeIPAddress + "/KeeperController";
    String strEncryptedUserID = "";
    String strEncryptedPassword = "";
    try{
        System.out.println("Contacting credential keeper");
        KeeperController myKeeper = (KeeperController) Naming.lookup(strHomeName);
        System.out.println("Keeper contacted");

        //Get user ID if demanded by service
        if (blnUserIDRequired.booleanValue()){
            strEncryptedUserID = myKeeper.getUserID(strCardID);
        }

        //Get Password if demanded by service
        if (blnPasswordRequired.booleanValue()){
            strEncryptedPassword = myKeeper.getPassword(strCardID);
        }
    }
}
```java
    catch (Exception e) {
        System.err.println("RMI exception: "+ e.getMessage());
        e.printStackTrace();
    }

    if (strEncryptedUserID == null || strEncryptedPassword == null) {
        System.err.println("Keeper didn't return required credentials");
        System.exit(0);
    }
    else {
        if (blnUserIDRequired.booleanValue() &&
            strEncryptedUserID.equals("")
            ) { System.err.println("Keeper didn't return required user ID");
            System.exit(0);
        } if (blnPasswordRequired.booleanValue() &&
            strEncryptedPassword.equals("")
            ) { System.err.println("Keeper didn't return required PIN Code");
            System.exit(0);
        }
    }

    //Get Access code for decryption and executing the service
    System.out.println("Please enter access code for decryption" +
        "and completing the service:");
    reader = new BufferedReader(new InputStreamReader(System.in));
    String strInputCode = null;
    try {
        strInputCode = reader.readLine();
    }
    catch (IOException e) {
        System.out.println("Could not read input; please start application again");
        e.printStackTrace();
        System.exit(0);
    }

    DesCryp decrypter = new DesCryp(false);
    String strDecryptedUserID = new String();
    String strDecryptedPassword = new String();
    System.out.println("Encrypted userID = " +
        strEncryptedUserID);
    System.out.println("Encrypted password = " +
        strEncryptedPassword);

    //Decrypt credentials
    try {
        strDecryptedUserID =
        decrypter.deCrypt(strEncryptedUserID,
            strInputCode);

        strDecryptedPassword =
        decrypter.deCrypt(strEncryptedPassword, strInputCode);
    }
    catch (SecurityException e) {
        System.out.println("Code not accepted!");
        System.exit(0);
    }

    System.out.println("Decrypted userID = " +
        strDecryptedUserID);
    System.out.println("Decrypted password = " +
        strDecryptedPassword);

    //Show credentials
    boolean binSuccess = false;
    try {
```
if (serviceController == null) {
    System.out.println("Service out of time, contacting again...");
    serviceController = (ServiceController) Naming.lookup(strServiceName);
    System.out.println("Service contacted");
    System.out.println("Showing credentials...");
    blnSuccess = serviceController.showCredentials(strDecryptedUserID, strDecryptedPassword);
}
} //end main

KeeperAccessor.java
package client;
import java.io.*;
/**
 * Interface that must be implemented when constructing a Keeper Accessor
 * @author Vegard Steinsholt
 */
public interface KeeperAccessor {
    /**
     * Obtains the IP address of the client's credential keeper
     * @return a String containing the credential keeper's IP address
     */
    public String getHomeIPAddress() throws IOException, FileNotFoundException;
}

FakeKeeperAccessor.java
package client;
import java.io.*;
/**
 * Implementation of KeeperAccessor to test the design
 * @author Vegard Steinsholt
 */
public class FakeKeeperAccessor implements KeeperAccessor {
    /**
     * Default constructor
     */
    public FakeKeeperAccessor() {
        super();
    }
    /**
     * Retrieves the IP address from a file on the client
     * @return a String containing the IP address
     */
    public String getHomeIPAddress() throws IOException, FileNotFoundException{
BufferedReader reader = new BufferedReader(new FileReader("\resources\IPaddress.txt");
    String strIPAddress = reader.readLine();
    return strIPAddress;
}
Appendix D: Jeode Runtime Documentation

Jeode™ documentation

- Introduction
- Installation
- Demonstrations supplied
- Running Java applets
- Running Java applications
- Command-line options
- FAQ
- Further information
- Disclaimer

Introduction
This document describes how to run JeodeRuntime on your Compaq iPAQ Pocket PC device. JeodeRuntime is a fully-certified implementation of Sun's PersonalJava 1.2 specification. JeodeRuntime can be used:
- As a plug-in to your iPAQ's Pocket Internet Explorer, to run Java applets from a web page.
- As a stand-alone Java VM (Virtual Machine), to run Java applications on your iPAQ.

Note: Some Java applets on web pages are not designed for the memory constraints of your iPAQ. Please see the FAQ section for guidelines on what types of applets you can expect to run.

Installation
JeodeRuntime is provided as an executable file. After you copy JeodeRuntime onto your PC, simply click on the executable file to install JeodeRuntime onto your iPAQ. By default, JeodeRuntime files are installed in \Windows and subdirectories of \Windows on your iPAQ.

Notes:
- You must have a working ActiveSync connection between your Windows PC and your iPAQ (for ActiveSync trouble-shooting information, please see the Microsoft web site).
- For Windows 2000 PCs, you must have administrator privileges set.

Uninstallation

Master Thesis at Department of Telematics, NTNU 2003
Vegard Steinsholt
To uninstall JeodeRuntime on your iPAQ, select **Start > Settings**, select the **System** tab, select **Remove Programs**, then remove **Insignia Solutions JeodeRuntime**.  
**Note:** if uninstallation fails, please reset your iPAQ, then re-try.

---

**Demonstrations supplied**

We provide a Java applet and a simple Java application, to demonstrate JeodeRuntime functionality:

- **Quasar**: a Java applet, invoked by opening a HTML file.
- **PrimTest**: a Java application which uses the AWT graphics library to demonstrate some Java graphics capabilities.

To run the demonstration applications or applet, select **Start > Programs > Jeode > Examples**, then select the demonstration application or applet to run. For example, to run the Quasar demonstration applet, select **Start > Programs > Jeode > Examples > Quasar**.

---

**Running Java applets**

The Jeode plug-in for Pocket Internet Explorer is included when you install JeodeRuntime.

When your browser next opens a web page which contains a Java applet, the plug-in runs the applet. This all happens automatically - no user intervention is needed.

For more information about running applets (including limitations), please refer to the **FAQ** section.

---

**Running Java applications**

To run a Java application using JeodeRuntime, you will need to copy the application classes, libraries and properties to your iPAQ.

You can then launch JeodeRuntime in two ways:

- Tap on the EVM icon (**Start > Programs > Jeode**), then enter options at the **Enter args** prompt as follows:
  
  ![Command-line options](image)

  where:

  - **[command-line options]** are **command-line options** to pass to the EVM (for example, **-cp**).
  - **class-name** is the class name for your Java application (for example, **PrimTest**).
  - **[class-arguments]** are any optional arguments for the Java application to run.

  For example:
  
  ```
  -cp \Windows\lib\jeodedemos.jar PrimTest
  ```

- Run JeodeRuntime as a shortcut.

Create then tap on a shortcut (a .lnk file) to **evm.exe** in the **Start** menu. This allows you to store EVM arguments (class name and command-line options) in the **Properties** of the shortcut, avoiding the need to re-type them every time you launch JeodeRuntime.

See the **Creating a shortcut** section below for more information.

---

Master Thesis at Department of Telematics, NTNU 2003  
Vegard Steinsholt
Creating a shortcut

To create a shortcut, use File Explorer to go to the Mobile Devices > My Pocket PC > Windows window, select evm.exe, select Edit menu > Copy, then select Paste Shortcut.

This creates a file Shortcut to evm.exe.lnk. Copy this file to a directory on your PC, and open it with a text editor such as Notepad. The contents of the file will look like,

18"\Windows\evm.exe"

You can now edit this file to add command-line parameters. For example, to run the supplied PrimTest example, you could edit the file to be:

18"\Windows\evm.exe" -cp \Windows\lib\jeodedemos.jar PrimTest

You can add other command-line options to the shortcut as required, up to 255 characters.

You can now copy the shortcut back to the iPAQ, then tap on it to run the Primitives Test application.

Command-line options

This section describes some common JeodeRuntime command-line options you can pass to the Jeode EVM, when running via the command line or setting up a .lnk file.

-?
-classpath or -cp
-D
-v or -verbose
-version
-Xnowinceconsole
-?, -h or -help

Displays help for all JeodeRuntime command-line options.

-classpath <pathnames> or -cp <pathnames>
Specify the path(s) used for loading application classes. The pathnames are separated by semi-colons.

For example, to include classes contained in jeodedemos.jar, you could use the command:

-cp \Windows\lib\jeodedemos.jar

-D<propertyName>=<value>
Supplies the value for a JeodeRuntime or standard Java system property.

For example, to keep the display console open, use the command:

-Djeode.evm.console.local.keep=TRUE

-v, -verbose
Causes messages to be displayed on the JeodeRuntime console when:

- A class file or dynamic library is successfully loaded.
- A garbage collection cycle is performed.
-version
Displays JeodeRuntime version information on the EVM console.

-Xnowinceconsole
Use the -Xnowinceconsole option to disable the EVM console if you do not want to see it - for example, when running graphical applications.

FAQ
This section answers frequently-asked technical questions about JeodeRuntime.

- Why does my applet not work?
- Why do I get a "Failed to run applet" message?
- Does JeodeRuntime work with JavaScript?
- Can I discard some JeodeRuntime files to save memory?
- How do I define classpath settings?
- Is Swing supported?
- How do I prevent the console from closing when the EVM terminates?
- Why can't my program find a data file?

Why does my applet not work?
There are two common reasons why a Java applet may not be able to run:

- Applets in CAB files
JeodeRuntime does not currently support loading applets from Microsoft CAB files, as CAB files are not part of standard Java technology. So if the web site makes assumptions (based on the type of browser) about the behavior of the VM being used, this may result in problems loading the applet.

- Applets in JAR or ZIP files
JeodeRuntime does support use of applets contained in JAR or ZIP files. However, because Pocket Internet Explorer is compatible with the HTML 3.2 Specification, Pocket Internet Explorer does not support using the ARCHIVE attribute of the APPLET tag in the HTML page calling the applet, and so Pocket Internet Explorer will not support loading applets from JAR or ZIP files under these circumstances. If you are able to rewrite the HTML in the web page calling the applet, you can use JAR and ZIP files, by specifying the archive as a parameter name as well as an ARCHIVE attribute (using <param name=archive>). For example:
<p param name=archive value=myclasses.jar>

Why do I get a "Failed to run applet" message?
You may see a message of the type:
Failed to run applet. Please reload
This means that the plug-in has detected a condition that prevents the applet from being run. There are various

Master Thesis at Department of Telematics, NTNU 2003
Vegard Steinsholt
possibilities - one common problem is lack of memory on an iPAQ, as some applets are written assuming unlimited amounts of memory to be available.

The Jeode plug-in has no control over the memory requirements of the applet. If there is insufficient memory remaining for the plug-in to continue running, it will issue a message to the user and close down.

**Does JeodeRuntime work with JavaScript?**

JavaScript is an object scripting language for the creation and customization of applications. While Java is used by programmers to create new objects and applets, JavaScript is designed for use by HTML page authors to dynamically script the behavior of objects running on either the client or the server. JavaScript and Java are completely different programming languages; therefore, Jeode is not used to run JavaScript.

Also, the Jeode plug-in does not currently support communications between JavaScript and Java applets. This means that applets which require this interaction will not work with JeodeRuntime.

**Is Swing supported?**

Unfortunately, Swing 1.1.1 is not compatible with PersonalJava 1.2, so JeodeRuntime will not support Swing 1.1.1. The problem is due to an incorrect test in Swing 1.1.1, as described in bugID 4309057 on Sun's Java Developer Connection web site.

**How do I define classpath settings?**

You can set the classpath via the command-line -cp or -classpath option. See the Running Java applications section for more information.

**Can I discard some JeodeRuntime files to save memory?**

The minimum set of files depends on the application. If your applet or application does not require internationalization support, (that is, non-US English language support), you can discard the i18n.jar file. Similarly, if your application does not use graphics (AWT), you can discard the evmawt.dll and awt.jar files.

**Note:** if you then try to run an applet or application that requires either of these pieces of functionality, the applet or application will not run correctly, and may generate an appropriate error message.

**How do I prevent the console from closing when the EVM terminates?**

Use the -Djeode.evm.console.local.keep=TRUE command-line option. For example, to run the PrimTest...
demo application, and stop the console from closing, you could use a command-line invocation such as:
-Djdeoe.evm.console.local.keep=TRUE -cp \Windows\lib\jeodedemos.jar PrimTest

You can also use the -
Djdeoe.evm.console.local.paging=TRUE command-line option in a similar manner, to enable console paging - this is useful if your application sends more than one screen of information to the console.

Why can't my program find a data file?

Some Java applications look for files (data, preferences, and so on), in an assumed current working directory, rather than specifying an absolute pathname. On iPAQs, the current working directory is always assumed to be the root directory (\), so this may cause problems for some Java applications.

Further information

• JeodeRuntime on other platforms
• Jeode development environment

JeodeRuntime on other platforms

This version of JeodeRuntime is targeted at the Compaq iPAQ.
JeodeRuntime is also available on a number of other platforms - for more information, please contact Insignia Solutions.

Jeode development environment

This version of JeodeRuntime is provided for your personal use, to run Java applications and applets. Insignia Solutions also offers a version of the Jeode product suitable for a development environment, which includes extra tools, enhanced documentation, premier technical support, and other variants of JeodeRuntime. If you are interested in purchasing development seats of the Jeode platform for your development project, please contact Insignia Solutions.

Disclaimer

This documentation describes how to install and use software relevant to the JeodeRuntime product. Reasonable efforts have been made to ensure the information in this documentation is accurate and up to date. However, Insignia will not be responsible for any errors or omissions.

Copyright

© Copyright 1998 - 2001, Insignia Solutions plc. Insignia Solutions specifically retains title to and interest in all Insignia software.
The Jeode\textit{Runtime} software is furnished under a license agreement and may only be installed, used, or copied in accordance with the terms of that agreement. Some or all of this work is covered by patent. All use of this product is strictly limited by the terms of the associated license agreement. Some of the functions described here may not be applicable to your system, depending on how your system is configured.

\textbf{Trademarks and acknowledgements}

Insignia acknowledges that some proprietary programs, products or services may be mentioned in this documentation. Names and logos related to these programs, products or services are trademarks or registered trademarks of their respective vendors and/or distributors.

Insignia and Insignia Solutions are registered trademarks and Jeode, EVM, Embedded Virtual Machine, and Jeode\textit{Runtime} trademarks of Insignia Solutions, Inc.

Sun and Java are registered trademarks of Sun Microsystems. Windows is a registered trademark of Microsoft Corporation.

iPAQ is a registered trademark of Compaq Computer Corporation.
Appendix E: Java Security Policy File

//
// This file provides an example security policy for use with the
// Personal Java Application Environment (PJAE) version 1.2.
// The policy described below is identical to that used by default
// by Jeode/EVM, however by default this file itself is NOT read. To
// make changes to this default security policy uncomment the line ...
//
// policy.url.1=file:${java.home}/lib/security/java.policy
// ...
// in the java.security file in this directory and then apply the
// required changes to the policy given here.
//
//
//=
// (c) Copyright Insignia Solutions plc, 2001
//=
//=
// Standard extensions get all permissions by default
grant codeBase "file:${java.home}/lib/ext/" {
    permission java.security.AllPermission;
};
// default permissions granted to all domains
grant {
    // allows anyone to listen on un-privileged ports
    // changed by Vegard Steinsholt
    permission java.net.SocketPermission ":*:1024-",
    "listen,connect,accept";

    // added by Vegard Steinsholt
    permission java.util.PropertyPermission
    "java.rmi.registry.packagePrefix", "read";
    permission java.lang.RuntimePermission "modifyThreadGroup";

Software Architecture for On-line Electronic Wallet

permission java.util.PropertyPermission
"java.rmi.server.codebase","read";
permission java.lang.RuntimePermission "createClassLoader";
permission java.net.SocketPermission ":80","connect";
permission java.io.FilePermission "\resources\IPaddress.txt","read";
permission java.io.FilePermission "\resources\AccessCode.txt","read";
permission java.io.FilePermission "\resources\deskey.dat","read";

// "standard" properties that can be read by anyone
permission java.util.PropertyPermission "java.version","read";
permission java.util.PropertyPermission "java.vendor","read";
permission java.util.PropertyPermission "java.vendor.url","read";
permission java.util.PropertyPermission "java.class.version","read";
permission java.util.PropertyPermission "os.name","read";
permission java.util.PropertyPermission "os.version","read";
permission java.util.PropertyPermission "os.arch","read";
permission java.util.PropertyPermission "file.separator","read";
permission java.util.PropertyPermission "path.separator","read";
permission java.util.PropertyPermission "line.separator","read";
permission java.util.PropertyPermission "java.specification.version","read";
permission java.util.PropertyPermission "java.specification.vendor","read";
permission java.util.PropertyPermission "java.specification.name","read";
permission java.util.PropertyPermission "java.vm.specification.version","read";
permission java.util.PropertyPermission "java.vm.specification.vendor","read";
permission java.util.PropertyPermission "java.vm.specification.name","read";
permission java.util.PropertyPermission "java.vm.version","read";
permission java.util.PropertyPermission "java.vm.vendor","read";
permission java.util.PropertyPermission "java.vm.name","read";
}