Implementation of an Instant Messaging Client using the OMA IMPS Protocol

Fredrik Stålnacke

Abstract

This thesis covers the area of instant messaging. Some of the bigger commercial clients are discussed in respect to their protocol and capabilities. One of the goals of the report is to create an instant messaging client that implements the *Instant Messaging and Presence Service*, IMPS, protocol developed by the organization *Open Mobile Alliance*, OMA. The OMA IMPS protocol is released as an open standard, therefore free for everyone to implement.

The client is implemented using Java with the configuration *Connected Limited Device Configuration*, CLDC, and the profile *Mobile Information Device Protocol*, MIDP. Connection to Internet or other data network is made through the, in java built-in, data connections. The client should be able to run on a mobile terminal and therefore it is required to use limited amount of working memory, storage space and processor cycles.

In this thesis, there is also a discussion over security issues in instant messaging environments. One of the major problems with OMA IMPS is that the system may be set up in a way allowing the user to authenticating in a non-secure way, giving an opportunity for a malicious user to gain unauthorized access to the system.

Contents

List of Figures

Summary

Instant messaging is a quick growing way of communication. Today there are many clients available on the market using their own protocol and ways of communications. Until recently, there existed no protocol that defined an open standard that anyone could implement. To fill this hole, an organization called Wireless Village was created. The organization later changed their name when they joined with some other organizations to *Open Mobile Alliance*, OMA. This organization was started by three major telecom companies: Ericsson, Nokia and Motorola.

The standard created was given the name *Instant Messaging and Presence Service*, IMPS. It consist of four parts: Instant Messaging, Presence, Groups and Content Sharing. The standard was made open for everyone to implement and designed so that it could be used on a mobile terminal with limited memory recourses, network access and processor capabilities.

There are several ways in which an instant messaging client can be implemented in order to execute on a mobile terminal. One of these ways is to write a *Mobile Information Device Application*, MIDlet, in Java. With this approach, the MIDlet will be general and should be able to execute on all mobile terminals supporting MIDlets. This was the approach for this thesis. A client, implementing the OMA IMPS protocol was constructed and tested against a simple server.

The client is implemented so that it uses existing methods of creating data connections. Hence the client has no knowledge over which medium the data is transmitted. The advantage with this is that the client can be made more straightforward and that it will be able to run on more types of mobile terminals.

There are many security issues concerning the use of instant messaging clients. Several of them are not always handled with proper respect. One of the major problems with the OMA IMPS protocol is that it offers support for both secure and not so secure methods for logging in to the server and sending messages. This flexibility may result in that only the easier none secure method is implemented causing a security issue.

By designing the instant messaging system in a way so that it makes use of all the security features that the OMA IMPS specification defines, a system can be made relatively secure. However it is important that security issues are kept in mind during the whole design.

Acknowledgements

I would like to express my gratitude to all the people who have helped me in one way or the other to make this thesis possible. A special thanks to my two tutors:

First, I would like to thank my supervisor Thomas Nilsson at Umeå University for his excellent guidance and support throughout the project. Mr. Nilsson has given me many valuable ideas and suggestions during numerous discussions over this report.

Mattias Ohlsson at Teleca Software Solutions also has my complete gratitude. His knowledge and experience in system designing played a very important role in the design of the system.

CHAPTER 1

Introduction

1.1 Background

Instant Messaging is a quickly growing area of communication. As of today, there are many clients on the market all implementing their own protocol. Until recently, the two major problems with most of these protocols are that they are "company-confidential" and that they cannot communicate with each other. Until recently, this has been the case. An organization called *Open Mobile Alliance*, OMA, was created by three major telecom companies to address this problem.

> "The mission of the Open Mobile Alliance is to grow the market for the entire mobile industry by removing the barriers to global user adoption and by ensuring seamless application interoperability while allowing businesses to compete through innovation and differentiation." [Open Mobile Alliance Ltd., 2003]

OMA has created a specification for instant messaging called Wireless Village. The purpose of this specification is to create an instant messaging protocol that should be available to the public and possible to implement on both mobile and stationary devices.

Teleca Software Solutions was interested in this protocol and decided to create a client as a proof of concept, which has resulted in this report.

1.1.1 OMA IMPS

OMA IMPS, *Open Mobile Alliance Instant Messaging and Presence Service*, is base an open standard that was approved to be released in its first version, 1.0, in February 2002. The initiative to create this project was taken by three major telephone companies: Ericsson, Motorola and Nokia. [Cherry, 2002]

In July 2002, the standard was updated to version 1.1. The changes were made to reflect reviews made by the supporting companies. The purpose was that the changes should help to insure interoperability. [Open Mobile Alliance, 2003b]

Since the start, many other companies have joined the project. As of 13^{th} April 2003, 192 companies where involved in the project. Several of those companies are AOL, Orange France, Siemens, Sony Ericsson and Vodafone can be found. [Open Mobile Alliance, 2003]

1.2 Objectives

The main goal of this thesis is to develop an Instant Messaging client that implements the specification from OMA, called IMPS. The client will be written using *Java 2 Micro Edition*, J2ME. The goal is not to create a client that will implement the whole standard and work together with all possible servers, but to create a client that shows a proof of concept - that it is possible to create an instant messaging client that supports this protocol. The test prototype should initially run on a phone emulator on an ordinary computer. Hopefully it should be possible to test on a real mobile terminal.

When designing a program that is meant to execute on a mobile terminal, high demands are set in respect to memory usage, footprint size and the use of common resources. This thesis will show some methods in which these issues can be handled.

Network access is handled on a lower level in the mobile terminal, and has therefore almost no importance to the implementation of the client. Discussion over how the data is transferred through the air interface is done anyway, to give a brief overview before discussing security related issues.

Many problems surround the security in instant messaging. The problems that will be discussed in this report are the risks with replay attack, session hijacking and to some extent, the risk of having users in the system. Possible solutions to the problems presented will also be discussed.

1.3 Instant Messaging

Instant messaging can be divided into four parts. This is also the way that the OMA IMPS specifies the protocol. The four sections are:

Instant messaging The term Instant Messaging refers to the way in which messages are transmitted. To the user it seems as messages are transmitted instantly or in what also could be called ”real time“.

Presence A bread concept that includes the ability for a user to see if other users are currently online and the ability that a user could check the present location of another user.

Shared Content Users can share content with each other. This part is defined by the OMA IMPS specification as a concept, but there is no information on how this should be implemented.

Groups The ability for users to be part of groups where they can send messages or grant permissions to a group of users.

There are many different Instant Messaging systems available on the market today. Some of the more common clients will be described in chapter 3.

CHAPTER 2

Method Outline

The project started with a study of the specification from Open Mobile Alliance concerning the Wireless Villager standard[Open Mobile Alliance, 2003a]. A project plan was constructed with deadlines, tasks and requirements.

The first task was to create a template for this report containing the basic structure and all the main sections to be included. When the report structure was completed, more extensive research was done in respect to security issues in the protocol.

After this more in depth study, the time had come for designing how the system should work. A lot of effort was put into the system design in order to create a system where the classes could interact with each other in a good way.

When system design was completed, the project went into its next phase; system implementation. The conditions for the project took a delay in the beginning of this phase when the new version, 1.2, of the specification was released. The new version was not completely compliant with version 1.1, so some changes had to be made in the design.

The goal was that the client should be tested on both target and host after the implementation phase was over. Unfortunately this could not be completely carried out since there where no server available at that time to test on. However, a simple dumb "server" was constructed for the purpose on making some simple tests.

The documentation was developed in parallel with the rest of the work during the whole project. Since the conditions changed, some of the documentation had to be updated.

CHAPTER 3

Related work

There exist many instant messaging clients on the market. Some of those implement the wireless village specification. However, among the clients that are most frequently used, none use this specification. In this chapter some of the clients with the most users will be presented.

3.1 Trillian

Trillian is a multi-medium instant messaging client. It has no protocol of its own or network to communicate over, but make use of networks from other systems. It can connect to ICQ, AOL Instant Messenger, MSN Messenger, Yahoo Messenger and IRC networks with their protocols. Since it does not have its own network it requires that users of Trillian registers on all networks that will be used. [Cerulean Studios, 2003]

3.2 AOL Instant Messenger, AIM

The AIM uses two different protocols. One called OSCAR, *Open System for Communication in Real-time*, and TOC, *Talk To Oscar*. The first is designed for AOL's own clients. The second, which has limited functionality, is designed for developers who want to develop there own clients. TOC is limited. It only contains the functions for sending messages and almost nothing else to prevent third-party vendors from making a "better" client than the original. The OSCAR server also tries to block clients that are believed not to be original applications. E.g. AOL tries to block Trillian. [Hindocha, 2003b]

Normally an AIM client connects to port 5190 on login.oscar.aol.com but if the client can't establish a connection to that port the client assumes that the port is blocked by a firewall and tries to reconfigure itself to bypass the firewall. That can be done by e.g. using port 80 (HTTP) or 21 (FTP) instead. It can also make use of an HTTP proxy to bypass even networks where packet analyses are used to block traffic. [Hindocha, 2003b]

3.3 ICQ

The software company Mirabilis released the first version of ICQ, pronounced "I seek you", in November 1996. June 1998, AOL bought Mirabilis and thereby also ICQ. The instant messaging client used its own protocol but when AOL took over, they changed the protocol to be the same as in AIM. AIM and ICQ uses the same protocol, they are developed by the same company, and they have about the same functionality and both uses the same techniques to try to bypass blocking firewalls. [Hindocha, 2003b]

In February 2003, ICQ had about 150 million registered users. [ICQ Inc., 2003]

3.4 MSN Messenger

MSN messenger, also known as Windows Messenger, is developed by Microsoft. The protocol used is *Microsoft Partner Network Protocol*, MSNP, an ASCII-text protocol. Information about MSNP can be found on the MSN Messenger Website[Microsoft Corporation, 2003] allowing third-party product interoperability. MSNP are not completely available, the only thing that has been released is an API to use for writing applications in Visual C++, Visual Basic or C#.

The default port for MSN Messenger is 1863 but like most other IM clients there are built-in functions for bypassing firewalls. [Hindocha, 2003b]

CHAPTER 4

Security aspects

4.1 Introduction - The need of security

There is always at least one question concerning security that must be answered before designing a system; how secure should the system be? The answer to this question is unfortunately very hard to answer until after the system has been released on the open market. The goal when designing a system should be to create a system that is secure enough. There is no need for spending more resources on securing the system, than the total worth of the information in the same system.

Instant messaging clients supply, as the name implies, two-way "almost" instant communication. Many users get the feeling that the use of IM clients increases their efficiency. As a result, the use of instant messaging is increased in both private and commercial applications. As with most applications, increasing the number of users increases the security risk. [Hindocha, 2003a]

Most people rely on the fact that their channels of communications are kept secure, and that a message sent can only be viewed by the recipient. This can have devastating consequences in a system that does not use a sufficient level of security. There are many examples of peoples getting in trouble when using instant messaging clients and not considering security. E.g. the CEO of eFront, Sam Jain, had his whole ICQ history stolen and posted on the Internet. This event resulted in that Jain had to resign, and that the company received a bad reputation from its customers. [Festa, 2001]

Some people may say, "Why should I care about security? I do not have any secrets. If someone wants to take part in my communication, let them." This is a very naive point of view. It might be true that most of the time they do not have any secrets. The problem is that people get used to this form of communication, and that after a while they no longer reflect over security issues. Eventually they feel so "safe" with this way of communication that they transmit sensitive information. On that day, someone may be eavesdropping.

When designing a system, one major goal must be to gain and keep the trust of the users. Suppose that you design a system which has security holes. Suppose also that someone takes advantages of some of these flaws, and that they publish the fact that they have done so. Even if this security breach was not significant,

it might still have damaged the trust that the average person puts into the system; a trust that might be extremely hard to regain.

4.2 Theory concerning security

This section will describe concepts important to know when discussing security issues in the instant messaging protocol from Open Mobile Alliance. Many of these concepts may seem trivial, but as we later will find out, they can all contribute to jeopardizing the security of the protocol.

4.2.1 Public key crypto

There exists a cryptographic term called "public key crypto" or "asymmetric crypto". The term was first invented by Diffie and Hellman in 1976. Communication between two users depends on that both parties have knowledge of a public key belonging to the other user. User A, must also have knowledge of one public encryption function E_A and a private decryption function D_A. The encryption function E_A must be created in a way that it is easy to calculate the result from D_A. It is also important that the function D_A is chosen in a way that it is very hard to compute from E_A. [Wätjen, 2002a]

In this section, some calculations will be shown. In those calculations the following symbols have been used to illustrate functions and variables:

Table 4.1: Symbols used to describe functions and methods for this chapter.

Symbol	Meaning
$\mathcal{M}$	The clear text message to encrypt.
C	The encrypted text.
A	The public key of user A.
A'	The private key of user A.
B	The public key of user B.
B'	The private key of user B.
E_k	Function to encrypt a message with the key k.
D_k	Function to decrypt a message with the key k.

If Alice, A, would like to send a private message to Bob, B, she could do it as follows:

1. She fetches a copy of Bob's public key. She must either already have this key, or download it from a trusted source.
2. She calculates $C = E_B(\mathcal{M})$ and sends C to Bob.
3. Bob receives the message and decrypts it through $D_{B'}(C) = D_{B'}(E_B(\mathcal{M})) = \mathcal{M}$

[Wätjen, 2002a]

Through this method, no verification can be done regarding the authenticity of the sender, Alice, because anyone can fetch Bob's public key, use it for encrypting a message, and claim to be Alice. If verification is necessary, a signing method must be used. This can be done as follows:

1. Alice wants to sign the message $\mathcal{M}$. To do this she calculates $C = D_{A'}(\mathcal{M})$.
2. Bob verifies the signing by calculating $E_A(C) = \mathcal{M}$. If the result from the equation is a reasonable plain text, he can be sure that the message originated from Alice.

[Wätjen, 2002a]

This algorithm offers no encryption just the possibility for the receiver to acknowledge the senders identity. To have both encryption and sender verification the two equations that have been described here must be combined into one as shown below:

1. Alice wants to send an encrypted and signed message to Bob. She calculates $C = E_B(D_{A'}(\mathcal{M}))$.
2. Only Bob that knows the private key B can decrypt the message through $D_{B'}(E_B(D_{A'}(\mathcal{M}))) = D_{A'}(\mathcal{M})$. By using Alice's public key, he can then verify the validity of the message by calculating $E_A(D_{A'}(\mathcal{M})) = \mathcal{M}$. If $\mathcal{M}$ is a message that makes sense he is certain that it has originated from Alice.

[Stallings, 2001a]

4.2.2 Hash functions

A hash function is a function h, that follows the following principles.

1. If x is used as input, a constant number of bits are returned regardless of the number of bits in the input from the function $h(x)$.
2. Given x and h, it is easy to compute $h(x)$.

[Wätjen, 2002b]

The above principles are the basic requirements of a hash function. However if the function is to be considered "safe" it must also be designed in a way such that two arbitrary messages, $\mathcal{M}$ and $\mathcal{M}'$ where $\mathcal{M} \neq \mathcal{M}'$, are practically impossible to choose in a way that $h(\mathcal{M}) = h(\mathcal{M}')$.

Besides the above described attributes, hash functions used for authentication in the OMA IMPS protocol must also be one-way functions. One-way functions are functions where given a result from the hash function, z, it is practically impossible to find a message $\mathcal{M}$ such that $h(\mathcal{M}) = z$. [Wätjen, 2002b]

4.2.3 Login procedures

The OMA IMPS specification supports two different login procedures, 2-way login and 4-way login. Both these procedures will be described in more detail in chapter 4.2.3.1 and 4.2.3.2.

4.2.3.1 2-way login

The simplest form of login is the 2-way login. This method is also shown in figure 4.1. The 2-way login is a mandatory function that must be handled both by the server and by the client.

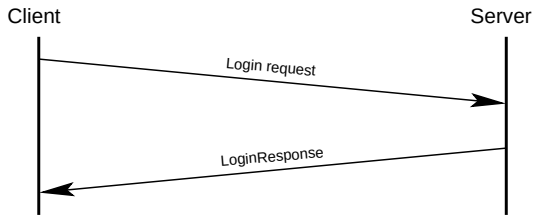

Figure 4.1: The steps needed to perform a 2-way login.

As the name implies two steps are taken in the login procedure. The first is that the client contacts the server and transmits its user identification and password. The server then tries to authenticate the user. If the authentication procedure is successful, a message is returned saying that the operation was successful. If the server is not satisfied with the authentication procedure, it may return a message saying that further authorization is required. In the second case, the client should try to login once more, but this time 4-way login should be used. [Open Mobile Alliance, 2003]

4.2.3.2 4-way login

The protocol for the 4-way login is more complex than the one for 2-way login. As seen in figure 4.2, and as the name implies, four steps are needed to carry out a complete login procedure. OMA's standard for instant messaging specifies six different authentication protocols. Exactly how this login is conducted and what authentication method will be used, depends on the capability of both the server and the client.

The steps performed are:

1. The client sends a login request, telling the server that it wants to login and giving a list of all the different protocols that the client supports. [Open Mobile Alliance, 2003]

2. The server answers the client with instructions of what protocol should be used and gives the client a challenge to prove that the client is who

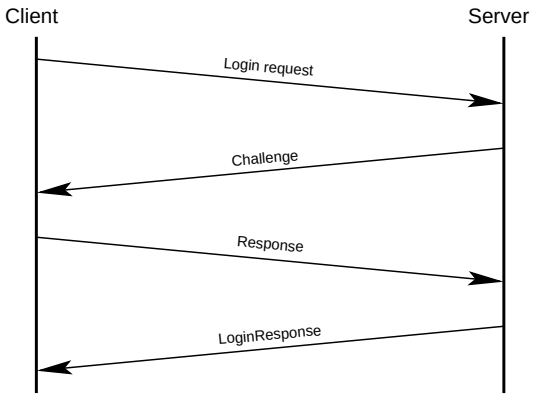

Figure 4.2: The steps needed to perform a 4-way login.

it claims to be. A challenge is a cryptographic term meaning that the client receives a random string that the server expects the client to encode with a secret key. The idea is that only a client that knows this key successfully is able to compute the result. The advantage of sending a challenge and awaiting a response, instead of sending the key itself, is that this method prevents eavesdroppers to record a login attempt and then login themselves with just a copy of the users answers (in cryptography this is normally called a replay-attack).[Open Mobile Alliance, 2003]

3. The client solves the challenge by appending the challenge to the users password and computes a hash value from the resulting string. The result from the hash function is converted into a base64 string and sent to the server. [Open Mobile Alliance, 2003]

4. Since the server knows both the users password and what challenge that has been sent the same calculation as done by the client can be done by the server. By comparing the results the server authenticates the user. The answer sent is only the result of the login. I.e. the client is allowed to connect or not. [Open Mobile Alliance, 2003]

The OMA IMPS specification specifies some different hash functions through which authentication can be preformed. These functions are:

Message Digest 4 and 5 The message digest functions are normally called MD4 and MD5 respectively. MD4 was created in 1990 by Rivest, and in 1991 he presented the new and improved version, MD5. The numbers 4 and 5 stand for the numbers in a row of proposed hash functions. Both functions return a 128 bit number, giving 2^{128} possible values. To find two messages that give the same result, 2^{64} calculations are required. The MD5 algorithm was developed since it was discovered that for MD4 only 2^{20} attempts were required to find a matching pair of messages. [Wätjen, 2002c]

Secure Hash Algorithm, SHA The SHA algorithm was developed in 1993 by The National Institute for Standard and Technologies. The message returned from the algorithm is 160 bytes long. To find a message corresponding to a given hash value about 2^{60} tries would be needed. [Stallings, 2001b]

4.2.4 Wireless Application Protocol, WAP

When data is sent from a mobile terminal to the Internet it is done through the *Wireless Application Protocol*, WAP. The WAP specification is not really a standard but more of a set of standards that together forms WAP. The protocols of WAP lie in the application, session, transaction, security, and transport layers. WAP has been designed to bridge the gap between wireless devices and the Internet, or any other network. WAP is a global standard that was originally created through the foundation of WAP Forum, read more at `http://www.wapforum.org/`.

WAP applications use *Wireless Markup Language*, WML, which is a branch from *eXtensible Markup Language*, XML. The use of WML can be compared to the way that the *Hyper-Text Markup Language*, HTML, is used for stationary computers.

One of the main goals when designing WAP was to create a set of protocols that would be more suited for use on a wireless device, since many of the existing Internet protocols such as *Hyper Text Transfer Protocol*, HTTP, have a great deal of overhead.

The mobile terminal that wants to communicate with a server on the Internet requires a WAP gateway that can translate between HTTP and *Wireless Transaction Protocol*, WTP. The advantage with this approach is that the gateway can reduce the overhead in the HTTP protocol.

The WAP standard exists in two major releases, the old 1.x and the newer, 2.0 release. One of the main improvements that have been added to the 2.0 release is improved security features. One of the changes that led to this was the introduction of the ability for the client to talk directly to a web server without any conversion by the gateway.

4.2.5 Transport bindings

There are many ways for a mobile terminal to communicate with a server. The security aspects vary with the protocols used. This section will cover the security aspects concerning some of the ways in which the client may communicate with the server.

4.2.5.1 HTTP

The use of http as protocol for mobile terminals was introduced in WAP 2.0. Before the changes in the WAP standard, only traffic through a gateway was possible. One big security advantage is that the client is now able to communicate directly with the server that provides the service that the client requests, see figure 4.3. [WAP Forum, 2001]

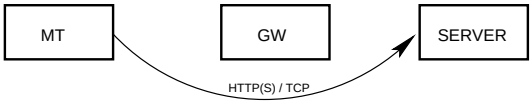

Figure 4.3: MT communicates directly with a server.

No encryption is used for this protocol; everything is transmitted in plain text. Just as discussed in previous chapters, the major security weakness is when the data leaves the wireless media and is transmitted over a public network, such as the Internet.

4.2.5.2 HTTPS

Https is a basically the same protocol as described in section 4.2.5.1. The difference is that some security has been added to the http protocol, where 's' in the abbreviation stands for "secure". This has been done by the use of *Secure Socket Layer*, SSL, encrypting the communication.

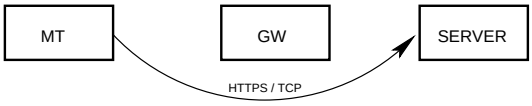

Figure 4.4: MT communicates directly with a server over a secure channel.

This protocol can also be compared with *Wireless Transport Layer Security*, WTLS, protocol, also described in section 4.2.5.5. Both the protocols make use of SSL encryption. The difference is that in https the data sent is not decrypted and then encrypted again by the gateway. This gives the client a possibility to verify that it is communicating with the server it believes to be communicating with. Another advantage is that the client now does not have to trust the gateway. [WAP Forum, 2001]

4.2.5.3 Short Message Service, SMS

OMA IMPS version 1.2 specifies that messages also should be able to be sent through a *Short Message Service Central*, SMSC. As illustrated in figure 4.5, the SMSC acts as a gateway. In figure 4.5 it is shown that the communication between the SMSC and the server is handled by the TCP protocol. This is not completely necessary. Many different protocols could be used for this traffic.

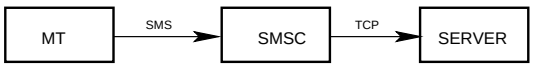

Figure 4.5: MT communicates with a server through SMSC.

The security in this protocol varies depending on the protocol chosen to transmit data between SMSC and server. The possibility to use SMS as a bearer of data has mostly been introduced for older clients, without network support, to be able to connect to a server. It is not very likely that this bearer will be used for a live system. Since this function will most likely not be implemented, it will not be discussed as a security threat.

4.2.5.4 WSP/WTP

The *Wireless Session Protocol*, WSP, and *Wireless Transfer Protocol*, WTP, are both parts of the WAP 1.x standard, which together can be compared with HTTP for stationary terminals. WTP handles the transfer of data while WSP handles session data. The sessions are used over several server queries in order to reduce the overhead for setting up network connections. [WAP Forum, 2001]

The mobile terminal communicates with the server through a gateway. The client has no built in support for communication over HTTP so it has to rely on the gateway to handle this.

Figure 4.6: MT communicates with a server through GW over WTP.

This method described in figure 4.6 is normally used for mobile terminals using WAP 1.x to connect to the Internet. The advantage with this method is that the client does not have to be able to use more than one protocol. The traffic between the client and the gateway should be relatively secure since the data packages are only transmitted in the network of the GPRS supplier. Data packages transmitted over the air interface is encrypted and sophisticated equipments are needed to monitor this traffic. The packages between the client and the gateway are protected but between the gateway and the server, they are sent without any protection. Data is sent in plaintext over a vulnerable network. [WAP Forum, 2001]

4.2.5.5 WSP/WTLS

Wireless Session Protocol, WSP, using *Wireless Transport Layer Security*, WTLS, is the way that the WAP 1.0 standard handles secure connection. It can be compared with the HTTPS protocol, but in this case the encryption is only between the mobile terminal and the gateway.

The encryption method in use differs between different phases during the lifetime of a session. An asymmetric cipher is used to exchange a common key that can be used as key for a symmetric cipher. This key is changed on a regular basis. [Winandy, 2000]

Figure 4.7: MT communicates with a server through GW over a secure channel.

The advantage with this method is that the data is encrypted when transmitted. Even if this protocol adds another level of security to the system, it still lacks some security features. All communication is being decrypted by the gateway and than re-encrypted using a different key. The result of this is that the client has to trust the gateway not to compromise the transmitted data. Another problem is that there is no way for the client to ensure that it is talking to the server it is talking to. [WAP Forum, 2001]

Fortunately, it is not that easy for a malicious user to replace the "real" gateway by one that compromises the security. The gateway is normally the only path through the operator's internal network and the public Internet. To be able to replace the server, the malicious user must get inside the operator network, which is normally strictly controlled.

4.3 Security in OMA IMPS

There are several parts in the standard that work together in order to try to make the system secure. The security in an implementation depends on many factors. Some of those will be discussed here.

4.3.1 Security in the network transport interfaces

There is always a possibility that someone is eavesdropping on the network traffic. This risk has to be taken under consideration. How severe this potential risk is depends on the way the system has been implemented. In section 4.2.5, different transport bearers were presented. They also affects the security of the whole system through the way they have been implemented.

There is no real way to completely protect the communication. There is always a risk. If communication takes part over open air space, there is a possibility that someone over hears the communication sent. This communication is encrypted, but there is unfortunately no guarantee of security. Network scanners for the GSM or GPRS networks can be bought on the Internet[Reuters Ltd., 2003]. Also, if the traffic is bound to copper wiring there is a risk that someone connects to the same wires and listens to the traffic. As long as this traffic is just sent within the phone company network the risk is not that serious, but when data

traffic is sent out onto a public switched network, such as the Internet, the risk increases dramatically.

4.3.2 Encryption

A system that implements the OMA IMPS protocol can use many different levels of cryptology. On the lowest level, encryption of the network traffic is used. This is always done on the air interface, by the GPRS network, but may also be used in other parts of the network.

On the system level of the application, encryption may be implemented by using an encrypted application protocol. E.g., HTTPS could be used to secure the communication with the server.

The highest level in the system where cryptology can be added is the application layer. This is done by ensuring that sensitive information, such as passwords, never are transmitted from the client. The server still needs to verify the client before logging it on to the system. The server can verify the client without any sensitive data is being transmitted by using the 4-way login method described in section 4.2.3.2.

4.4 Threats

As mentioned before, the security of the protocol lies largely on what level of security is used. However there are still some security breaches that can be used to compromise the security. Some of these threats lie in other protocols that the OMA IMPS protocol uses. In this section, some of these threats will be discussed.

4.4.1 Hijacking a session

When a user logs in to the server, the server creates a new session for the user. This is done so that the user should not have to send its password every time a package is sent. The session is normally identified by a session id. E.g., the string could consist of 25 random characters. This session id should be extremely hard to guess. When the user logs out from the system, this session is destroyed. There exists a problem with this procedure, since the only thing a client needs to impersonate another user is an active session id, for the user. However there can be some problems obtaining this session id but once again we come back to the point that it all depends on how the system is set up. If no encryption is in use, a malicious user could user a packet sniffer to obtain this key. After that, he could use a *Denial Of Service*, DOS, attack to shut out the real user and then start using the session.

If the system uses some level of encryption, it would be much harder to make use of different technique. It is nevertheless possible to do. One technique is

the 'Man in the middle attack'. All encryption protocols that the OMA IMPS standard has, all use asynchronous ciphers. By using this attack it might be possible to fool the client into thinking that it is communication directly with the trusted server, when it is really talks to a hostile server.

4.4.2 Replay attack

Replay attack is an attack method where the basic concept is to first record a valid message and then later replaying the same message. This method can be used on all systems that uses constant authentication information, such as passwords or electronically transmitted biometric data.

The risk that some malicious user takes advantages of this weakness against a server running OMA IMPS is rather low, at least as long as the server doesn't completely rely on password authentication.

4.4.3 Allowing the user to verify with 2-way login

By allowing a user to log in using the weaker 2-way login, a weak link has been introduced into the system. "A chain is never stronger than the weakest link." A malicious client does not have to use the more secure 4-way login but can freely choose the weaker 2-way login procedure.

4.5 Recommendations

This chapter will describe some of the issues that have to be considered when implementing a system based on the Wireless Village standard from OMA IMPS. By making sure this issues are covered the risk that some one misuses the system is reduced.

4.5.1 Counter measures against hijacking of sessions

It is not trivial to construct a system that fully guards against having the session hijacked. Some extra security measures can be taken in order to make it harder for whoever tries to hijack the session. Some examples of counter measures are described in the following chapters.

4.5.1.1 Limit the lifetime of a session

The shorter the period of time a session is used the less risk that someone has the time to take advantage of it. OMA IMPS does not specify any method that the server can use to force a client to start using a new session identifier other then sending a server initialized logout. By adding this feature to the protocol,

some of this risk could be reduced. Adding this functionality is unfortunately not completely free from problems.

If a method should be added so the server could enforce a change of the identifier of the session, this must be done in a secure way. The naïve way of just letting the client login once more might not be an improvement. Every time the client's user identity is sent to the server, there is a risk that someone is eavesdropping. A better solution would be to use the old session identifier as an identifier for generating a new one. If this approach is used the clients identification information does not have to be sent to the server more than once, regardless of how long the user stays online.

4.5.1.2 Store more information about the client

As explained earlier, there is a problem that someone can kidnap a session from a valid user through only knowing the session id. The server can however take precautions to avoid this. By storing information about the client together with the session, and the use of this information every time the session identifier is used, some extra security can be gained. Information that can be stored is e.g. IP address, client version, phone number of the client etcetera.

CHAPTER 5

Results

This assignment has consisted of both a theoretical and a practical part. The theoretical results are all described in this report. The practical part, which has been to implement an instant messaging client, can only partially be described since the rights to this program belong to Teleca Software Solutions. All results that can be published are presented in this section.

5.1 Development

The goal was do develop a client that complies with the mandatory functionality of the OMA IMPS specification. In this section, the reader will be presented with information about the basic idea behind the design and implementation of this system. Much of the information has been deliberately left out since the design and source code are the property of Teleca Software Solutions.

The implementation was preformed in three phases: design, implementation and testing. Those phases will be described more in section 5.1.4.

5.1.1 Mobile Information Device Profile, MIDP

Mobile Information Device Profile, MIDP, defines a set of functions to use when creating an application in Java for a mobile terminal. Java hasdifferent branches to cover different needs. In addition to the most basic functionality there are three editions, these are: *Java 2 Enterprise Edition*, J2EE, *Java 2 Standard Edition*, J2SE and *Java 2 Micro Edition*, J2ME. All editions have been developed for different uses. In the case of designing an application for a mobile terminal, J2ME would be the best choice, since it is the edition that was designed for use on mobile terminals.

For each of these editions there is one or many configurations defined. The configuration defines which *Virtual Machine*, VM should be used to execute the application. It is necessary that the mobile terminal have support for the VM for which the application has been written. The configuration *Connected Limited Device Configuration*, CLDC, defines the VM as KVM. The 'K' in KVM refers to kilobyte, because the KVM is designed to run with only 128Kb

of memory, which is very good when running a program on a mobile terminal with only a very limited amount of memory. [Meloan, 2003]

In addition to the CLDC, there should be a profile defining for example what classes and what *Graphical User Interface*, GUI components should be used. In this case, the profile MIDP has been chosen. A graphical representation of how things are connected can be seen in figure 5.1.

MIDP provides the core functionality that the application requires. It specifies interfaces for accessing user interfaces, network access, local data storage etc. There are currently two major versions of MIDP on the market, 1.0 and 2.0. MIDP 1.0 is currently more common in mobile terminals whereas the 2.0 version is still rather new. The major differences between the two versions are that the class library has been extended for MIDP 2.0, and that support for trusted MIDlets, Mobile Information Device Applications, has been added. This means that the MIDlet vendor can digitally sign the MIDlet before shipment and the customer can verify that the MIDlet is produced by the person or company expected.

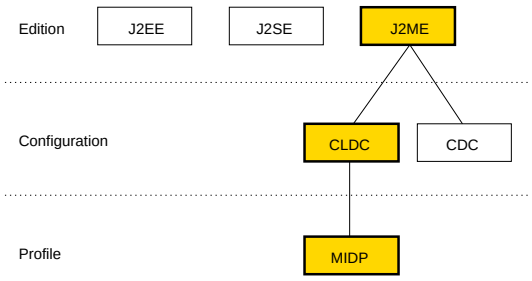

Figure 5.1: Overview of how some of the different parts of java are connected.

The classes specified by the profile are stored in the mobile terminal when shipped. As mentioned before, MIDP is the most common profile for mobile terminals but additions to this standard are becoming more and more common. One of the biggest is the *Vodafone Specific Class Library*, VSCL, developed by Vodafone for their Vodafone live mobile terminals. VSCL adds functionality for a MIDlet manufacturer to set some extra parameters on a MIDlet. These parameters can be copy protection, control of the ability of accessing the Internet, or preventing users to delete the MIDlet they downloaded. Aside from giving the developer better control over how their applications are used VSCL also allows the user to download and run more powerful applications.

5.1.2 Mobile Information Device Application, MIDlet

The Java programs that are normally loaded into a mobile terminal are called MIDlets. A MIDlet is a Java application designed for use on a mobile terminal. It consists of two parts, a jad and a jar file. Both of which will be described below.

5.1.2.1 jad-files

The name jad stands for *Java Application Descriptor*, and is a single text file containing information about the MIDlet. The information that is stored in this file depends on what version of MIDP that is used, commonly both versions contain information such as the vendor and the name of the application. The file also includes the path to where the jar-file can be downloaded.

5.1.2.2 jar-files

Java Archive files or shorter jar-files are a directory that has been compressed as a zip file and given the extension jar. Except for the main class file, the only file that is mandatory in the jar file is a file called *MANIFEST.MF*. This file contains some of the same attributes as the jad-file. The reason for storing this information in both the jad and the jar-file is for verification. The client can verify that the name of the vendor and the name of the MIDlet are the same in both files. By doing so, the client reduces the risk that someone has tampered with the file.

The jar file does not only hold the Manifest file, but also contains all classes of the MIDlet and all resources that the MIDlet needs, such as image and sound objects.

5.1.3 Design choices

The OMA IMPS protocol offers many possibilities of what can be implemented. Many of these choices are optional. Since the protocol stretches over many areas, a choice was made to start with the implementation of the parts “instant messaging” and “presence”, leaving “groups” and “content sharing” for later. To further limit the amount of work that this client would require almost all optional features where left out as well.

The specification defines many choices where it is mandatory for the client to support at least one of two optional features. This is the case with XML. The specification says that it is mandatory for the client to support ordinary XML or binary XML. When encountering choices of this kind, the decision was to choose the path that led to less work. For the specific case of XML, the choice fell on plain XML.

The choice that only mandatory features should be implemented has resulted in the fact that the client lacks some functions that normally would be expected from an instant messaging client. From the client’s interface, there is no way that a user could create a new user account, search and add users to his contact list, authorize or deny users to read away messages etc. Since the client should only be considered as a proof of concept this is acceptable, but these features should be implemented before releasing the client as an instant messaging client for common use.

5.1.4 Design

The first phase after studying IM clients was the design phase. About one week was spent trying to make an object oriented class diagram over how the classes should be connected. Some of the major design goals were to:

1. Create a design where the graphical user interface and the underlying protocol were as separate as possible. The reason for this was that it should later be possible to change the underlying protocol to another protocol if there should be other demands in the future.

2. Try to keep the size of the finished program as small as possible; due to the very limited amount of memory in many mobile terminals.

3. Minimize the use of memory and the use of the in Java built in function garbage collection. RAM memory in a mobile terminal is limited and should therefore not be wasted. The memory is not only limited, it also requires some overhead when allocated or de-allocated. To reduce this overhead functions should be added to enable the reuse of memory instead of always allocating new memory.

To try to achieve the first design goal, an abstract class was created for the protocol class to implement. The reason for this was that it should later be possible to create a new class that implemented the same interface and use the newly created class instead for communication.

In an attempt to honor the second design goal two different methods were used. The first method was to try to minimize the use of classes. An empty class has about 200 bytes overhead when compiled; this extra space could possibly be saved by reducing the amount of classes in the system. The second method for this design goal was a small program called an Obfuscator. When this program is executed on the target program it changes the names of all variables, classes and methods to short names that should take only a very limited amount of space in the jar file.

The last design goal, the reuse of allocated memory, was achieved by adding an extra layer on some of the memory handling functionality that enabled the reuse of memory.

5.1.5 Implementation

Since the system had a well thought-out design, the development part went straight on without any big problems. One goal with the implementation was to try and minimize the use of external programs. Even though attempts were

made to minimize the use of external programs, some were needed in order to make the whole developing process easier. These where:

Netbeans The development environment in which the code for the MIDlet was written. http://www.netbeans.org/

Ant A small program written in Java that allows make-scripts to be carried out. http://ant.apache.org/

A obfuscator A program to obfuscate the jar files created. The main purpose of this program was that it made the size of the binary file to shrink to about half its size.

5.1.6 Testing

At the same time as this project was carried out, another master thesis work was started at Teleca Software Solution in Umeå. The purpose with this second project was to create a server for this client to communicate with. Unfortunately, this project was delayed limiting the possibility to test the clients ability to communicate. In an attempt to test the client at all, a simple server was written. This server did not fulfill the demands of the OMA IMPS specification; neither did it allow real users to login. All communication sent to this server was handled with a simple pattern matching function that then chose a suitable response. This server filled the purpose as a testing environment, but it could not test protocol errors.

Since this client was only meant to be a prototype, this form of testing should be sufficient. The client has been tested on two different mobile terminals to see how it would work out. The first mobile terminal was the Nokia 7650 mobile phone. The virtual machine in this phone was not very well implemented. When the client was loaded, everything went well until the client tried to connect to the Internet, which made the phone reset the MIDlet.

The second mobile terminal that the client was tested on was a Sony-Ericsson P800. This phone had a much better developed virtual machine; the client could be loaded and executed without any problems. Unfortunately, the P800 did not handle soft keys[1] very well. Even though the program was executed without a problem, it is meaningless when there is no way to interact with it.

The last mobile terminal where the instant messaging client was tested, was a Sony-Ericsson T610. This phone is newer than the two phones tried before. Downloading and installing the MIDlet worked all right. The program was started, it connected to the test server, contacts were downloaded and instant messages could be sent. The biggest problem with using this phone was that all soft keys are handled by creating a list of all keys. This list can be accessed

[1]Soft keys is a term that refers to the keys that are placed directly by the display and where the text description of the key are placed on the screen by the key. In this way, the function of the key can be changed over time.

from one place. This way of handling soft keys results in the disadvantage that the user must make more key presses in order to perform actions.

I hope that mobile terminal vendors will present new phones that have a better-developed support for java. From three tested mobile terminals, the client could only run on one of these.

5.1.7 Problems surrounding implementation of MIDlets

Since MIDlets should be downloaded, stored and executed in a mobile terminal, it is important that the developer creates them in a way so the size is as small as possible. Many mobile terminals have a maximum limit for how big MIDlets that can be downloaded are. The size of an application can be reduced in many ways. Some of those are:

Choosing short names for variables and classes In Java, all the names of variables and classes are stored in the jar file. From this, it naturally follows that longer names cause a bigger overhead.

Keep package paths as short as possible The jar file is built according to the zip specifications. Names of files and directories are stored in a part called "central file header". This header is decompressed and the whole path is stored for every file. The path of the package must be stored for every file in the package causing a great overhead.

Use maximum compression The zip format offers different levels of compressions, by using the highest level the size of the jar file can be reduced.

Avoid unnecessary classes and deep inheritance structures An empty Java class requires about 200 bytes of space in the compressed jar file. When inheriting from an object, the whole object is inherited, not just the parts that may be relevant. Deep structures of inheritance often result in allocation of more memory than needed.

Use an obfuscator In this case, an obfuscator is a program that operates on compiled code. It replaces almost all the names of classes and variables with shorter names. This procedure has two functions. First, it alters the classes in a way so that it will be harder for persons trying to reverse engineer the MIDlet. The second function is that the obfuscator reduces the size of the jar file.

Avoid creating unnecessary objects When objects are created, they must later be removed. In Java, the garbage collector does this automatically. In many cases memory allocation and de-allocation of memory on a mobile terminal requires lots of resources. Not allocating more memory than needed, and trying to reuse memory can enhance the performance of the MIDlet.

5.2 Limitations

Since this should not be considered a fully developed product, it is still not free from errors. Known problems with the client and the protocol used are presented in this section.

5.2.1 Hardware limitations

The OMA IMPS specifies a method for reducing the network load that the client produces. This is done by creating a *Communications Initiation Request*, CIR, channel where the server can contact the client and inform it that there is new information available. If this function is not available, the client must poll the server periodically for new data.

This CIR channel should normally be implemented through SMS. I.e., the server sends a special SMS when there is new information available that the client should fetch. MIDP 1.0 only defines a limited amount of interfaces for communication with the hardware. No SMS interface is specified so the client cannot use this method to reduce network load.

5.2.2 Limitations within the system

The MIDlet does not access the screen of the device directly. Instead, it uses the generic GUI components defined by MIDP and supplied by the virtual machine. The program has no real control over how objects will be displayed on the screen, therefore it can not ensure that the program might look the same on all devices. This might be somewhat be confusing for the users.

5.2.3 Incomplete testing

As mentioned in section 5.1.6 the system has not been completely tested. This fact means, in reality there is no guarantee that the system will act or behaves as expected.

5.2.4 Only support for mandatory functions

Since the client only implements a small subset of the communications protocol, many features that a user might require have not been implemented. One of the more obvious functions that are missing is the ability to search for users and add them to the contact list. Today the client retrieves all users in the user lists from the server at start up. Those lists cannot be modified by the client.

5.3 User manual

This section will present a short description to how this program is used. The documentation here is in no way meant to be complete or cover all features that are available on the client. The application has gotten the name "Villager" and will be called so through this whole section. Since in Java all the actual drawings of what is shown on the screen are handled by the KVM, it is not sure that the screen shots presented here will look the same on all systems.

5.3.1 Starting Villager

When the program has been downloaded to the mobile terminal it can be started. The mobile terminal will then show a screen asking the user if the program "Villager" should be started. This is shown in figure 5.2.

Figure 5.2: The last screen shown before the midlet is started.

5.3.2 Connecting to a server

When Villager starts, it will automatically try to connect to a server. A progress meter will show the status of how far the login procedure has gone. A sample of how this may look can be found in figure 5.3.

Figure 5.3: A progress bar showing the progress of the login procedure.

There are some special cases when the client does not connect automatically. E.g., the user has requested that the password should not be stored or that the client is used for the first time and thus no user name or password has ever been entered. Then, instead of showing a progress meter, the program will present the user with a dialog where the user is asked to enter user name and password to be used for logging in on the server.

5.3.3 The idle screen

When the client has connected to the server and successfully logged in, an idle screen is shown, figure 5.4. The main component on this screen is a list of all users in all contact lists belonging to the user of the program.

Figure 5.4: The screen shown when the system is in idle mode.

For every user there is also information about what status mode the user is in. This is represented by different colored circles in front of the user's name. A green circle shows that the user is online and willing to chat. The yellow circle means that the user is in discreet mode, this mode should be read as the discretion is left up to the person that wants to start chatting. When a user is in this mode, he is most likely close to the client but might be in a meeting or involved in some other activity that makes him unavailable for communication. If there is a red circle before the name, the user is not available. It is possible to send messages to this user but it is not very likely that there will be a response to the message. Finally, there might also be a grey circle in front of the name; this means that the user is offline. Messages sent to a user with a grey indication might be discarded by the server without any notice.

5.3.4 A new message is waiting to be read

When in idle mode and a new message arrives, a picture of a letter will start to blink in front of the name of the sender, see figure 5.5. This indicator will be shown until the message is read.

Figure 5.5: A sample of the client when a new message is waiting to be read.

5.3.5 Chatting with a user

If a new message is sent or an incoming message is read, the client will display the chat screen as seen, in figure 5.6. Here the last five messages from that user are also shown. This screen gives the user the ability to write more messages to the person hi are chatting with.

Figure 5.6: The display when a chat session is in place.

5.3.6 Change the language used in the client

The client has support for more than one language. Currently there is only support for English and Swedish, but more languages could easily be added. Figure 5.7 shows the dialog where the user can choose what language that should be used. This setting can be changed at any time by accessing the settings menu. The changes takes effect immediately, are stored in the mobile terminal and automatically loaded the next time the client is started.

5.3.7 Setting a status text message.

The protocol gives the user the ability to set a short message along with the status chosen. This message is meant to be a more explanatory description of

Figure 5.7: Changing the language used by the client.

what the user is doing. The text can for example be, "I'm in a meeting, please do not disturb" for a user in discreet mode or, "I'm ready to chat with you" for an online user, as in figure 5.8. Setting this message is done through the settings menu. First, the user must choose for what mode the message should be changed and then the message can be written.

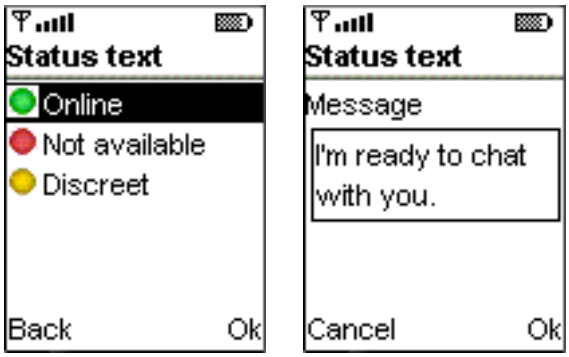

Figure 5.8: The user can change the message that is returned to users that request status messages.

5.3.8 Error handling

When the user is connected, a number of errors can occur. Effort has been taken in order to try to provide the user with meaningful feedback to what error has occurred, and what can be done to repair it. An example of an error message is shown in figure 5.9.

Figure 5.9: An example of how an error message is displayed.

CHAPTER 6

Conclusions

6.1 Conclusions from the implementation

Although Java is designed to be platform independent, work must still be done to be able to run an application on different systems. MIDP offers two different ways in which the screen can be accessed. The first one is a low-level access allowing the drawing of lines, circles and pixels. The other one is far more sophisticated, offering access to a whole API with item lists, event handling, et cetera. Using the more basic API gives the programmer lots of work handling, different screen sizes, color depths and user input possibilities. The second API also has some disadvantages, normally there is only one type of every object. This might sound all right, but it offers no variation and flexibility. E.g., the list class offers a list where text can be presented with or without an icon. For this, everything is fine, but what about when a list is needed where one of the options should be disabled? This is not really something that should be held against the MIDP API, it is hard to create a flexible list of functions that is suited for every need, but it is something a programmer must take into consideration when designing a program.

One goal when designing this system was to create the client in a way so that it should be possible to remove the OMA IMPS protocol and add some other instant messaging protocol. The whole system is built around this idea but many problems have been raised. What functionality can be expected from a general instant messaging protocol? Hopefully, there should exist functionality for sending messages to one another but should it be possible to add new users, should a user be able to see which users are online or not, should a user have to authorize other users before being added to their contact list? The questions are many, and it would have been impossible to implement a graphical user interface, GUI, that would have been suitable for all possible protocols. The solution to this problem has been to implement the GUI so that it would fit the OMA IMPS protocol. For any other protocol that would be used some reconstructing must be done.

6.2 Conclusions regarding security

There exists no system that is completely secure, and most likely, there will never be such a system. This is because the definition of security. The word covers everything from stopping intruders, to malicious system administrators and natural disasters. The most important thing to keep in mind when designing a system is that it must be secure enough, to start defining what there is that must be protected and first thereafter designing the system. The OMA IMPS protocol might have some problems concerning security but most of those can be solved when designing the system.

One thing that must be kept in mind when designing a system is the users. The persons using an instant messaging system might very well be the biggest security threat against the system. A system will never be secure if the people using it don't care about protecting it. Thus, the system must not only be designed with a high level of encryption and other security measures, it must also guard against its own users, preventing them from using simple passwords or any other actions them might take which jeopardizes security.

The OMA IMPS standard gives the developer the opportunity to choose simpler, less secure methods. This may result in that a developer working towards a deadline chooses the simpler method in order to deliver a system on time.

If someone manages to get hold of the session identifier of another user, this user can impersonate that user by stealing the session. The system can try to guard against attack of this kind by saving more information about the session and use this information when validating a user.

Finally, a system that makes use of 4-way login, MD5, sends all traffic by SSL and uses all other security functions that the OMS IMPS specification proposes. A system with a configuration like this should be conceivably to be relatively safe.

APPENDIX A

Terminology

CLDC — Connected Limited Device Configuration, the configuration used for the J2ME edition is Java. This configuration specifies what KVW should be used.

GPRS — General Packet Radio Service, is the packet data transmission or bearer service for GSM networks. It uses the same basic radio station infrastructure as the GSM network.

HTTP — Hyper Text Transfer Protocol, the underlying protocol used by the World Wide Web. The protocol defines how messages are formatted and transmitted, and how browsers should respond to various commands.

HTTPS — Hyper Text Transfer Protocol Secure, this protocol is basically the same as HTTP but with added security features. The encryption used is SSL.

IM — Instant Messaging, a type of service that enables a user to communicate with another user in a way that seems to be instant. Chat-rooms are typically examples of an IM service.

IMPS — Instant Messaging and Presence Service, the instant messaging protocol used for the implementation described in this report.

J2ME — Java 2 Micro Edition, The Java edition that is used when developing applications for mobile terminals.

KVM — Kilobyte Virtual Machine, the virtual machines that are implicitly used by the CLDC configuration. The name kilobyte refers to the fact that KVM is designed to work with only 128Kb of memory.

MD4 — Message-Digest Algorithm 4, creates a 128-bits “fingerprint” or “message digest” from a message.

MD5 Message-Digest Algorithm 5, creates a 128-bits "fingerprint" or "message digest" from a message. MD5 is a further development of MD4.

MIDP Mobile Information Device Profile, the Java profile used together with CLDC to define what classes should be available in the system.

MT Mobile Terminal, a terminal that is mobile.

OMA Open Mobile Alliance, an organization that tries to make the market for mobile products bigger by increasing application interoperability. To accomplish this, the organization has developed a couple of open standards that should help companies develop products that can communicate with each other.

PDP Packet Data Protocol, a PDP context link is used by the GPRS network to establish a wireless network connection between a GPRS enabled device and the GPRS network.

SHA Secure Hash Algorithm, an algorithm that produces a 160-bit output from a given input. The algorithm is called secure because it is competently infeasible to find a message which corresponds to a given digest.

SMS Short Message Service, a service to send small messages to mobile terminals.

SMSC Short Message Service Central, a server that handles SMS.

SSL Secure Socket Layer, a protocol developed by Netscape for transmitting private documents via the Internet. SSL is a public key cryptographic method. This means that two keys exist; one private and one public. What is encrypted with the public key can only be decrypted with the private, and vice-versa. SSL has been approved by the Internet Engineering Task Force (IETF) as a standard.

TCP Transmission Control Protocol, a protocol that assures that a packet is sent accurately.

UDP User Datagram Protocol, a protocol that is used instead of TCP when a reliable delivery is not required. A big advantage of UDP in comparison to TCP is that the overhead is not that big.

URL Universal Resource Locator, the Internet address that defines the route to a file on the Web or any other Internet application.

VM	Virtual Machine, The program that is used to execute Java programs is called Virtual Machine.

References

[Cerulean Studios, 2003] Cerulean Studios (2003). Cerulean studios -> support. Webpage. http://www.trillian.cc/support/manual.php?hchap=1&hsub=1&hsect=1; accessed September 23, 2003.

[Cherry, 2002] Cherry, S. M. (2002). Talk is Cheap, Text is Cheaper. *IEEE Spectrum*, 39(5).

[Festa, 2001] Festa, P. (2001). ICQ logs spark corporate nightmare. Webpage. http://news.com.com/2100-1023-254173.html; accessed March 3, 2003.

[Hindocha, 2003a] Hindocha, N. (2003a). Instant insecurity: Security issues of instant messaging. Webpage. http://www.securityfocus.com/infocus/1657.

[Hindocha, 2003b] Hindocha, N. (2003b). Threats to instant messaging. http://securityresponse.symantec.com/avcenter/reference/threats.to.instant.messaging.pdf.

[ICQ Inc., 2003] ICQ Inc. (2003). About the icq - icq.com. Webpage. http://company.icq.com/info/community.html; accessed September 22, 2003.

[Meloan, 2003] Meloan, S. (2003). Wireless wonders: Java technology for wireless data solutions. Webpage. http://java.sun.com/features/2000/02/wireless2k.html; accessed June 24, 2003.

[Microsoft Corporation, 2003] Microsoft Corporation (2003). .net messenger service – gratis snabbmeddelandetjänst. Webpage. http://messenger.msn.com/; accessed March 5, 2003.

[Open Mobile Alliance, 2003a] Open Mobile Alliance (2003a). OMA IMPS version 1.2. http://www.openmobilealliance.org/omacopyrightNEW.asp?doc=OMA-IMPS-v1_2-20030221-C.zip.

[Open Mobile Alliance, 2003b] Open Mobile Alliance (2003b). The Wireless Village Initiative. Webpage. http://www.openmobilealliance.org/wirelessvillage/pr20020715.html; accessed March 6, 2003.

[Open Mobile Alliance, 2003] Open Mobile Alliance (2003). The wireless village initiative. Webpage. http://www.openmobilealliance.org/wirelessvillage/supporters.html; accessed March 24, 2003.

[Open Mobile Alliance, 2003] Open Mobile Alliance (2003). WV-023 Client-Server Protocol DTD and Examples Version 1.2. Technical report, Open Mobile Alliance Ltd. http://www.openmobilealliance.org/wirelessvillage/.

[Open Mobile Alliance Ltd., 2003] Open Mobile Alliance Ltd. (2003). Open mobile alliance. Webpage. http://www.openmobilealliance.org; accessed September 15, 2003.

[Reuters Ltd., 2003] Reuters Ltd. (2003). Reuters | latest financial news / full news coverage. Webpage. http://www.reuters.com/newsArticle.jhtml?type=technologyNews&storyID=3380775; accessed September 25, 2003.

[Stallings, 2001a] Stallings, W. (2001a). *Sicherheit im Internet*, pages 100–101. Addison-Wesley Verlag. ISBN: 3-8273-1697-9.

[Stallings, 2001b] Stallings, W. (2001b). *Sicherheit im Internet*, pages 80–83. Addison-Wesley Verlag. ISBN: 3-8273-1697-9.

[WAP Forum, 2001] WAP Forum (2001). WAP Architecture. Technical report, Wireless Application Protocol Forum Ltd. http://www1.wapforum.org/tech/documents/WAP-210-WAPArch-20010712-a.pdf.

[Winandy, 2000] Winandy, M. (2000). WTLS - Wireless Transport Layer Security. Technical report, Universität Bonn, Institut für Informatik III. http://www-student.informatik.uni-bonn.de/~winandy/WAP/wtls.pdf.

[Wätjen, 2002a] Wätjen, D. (2002a). *Kryptologie*, pages 71–74. Institut für Theoretische Informatik, Technische Universität Braunschweig.

[Wätjen, 2002b] Wätjen, D. (2002b). *Kryptologie*, pages 91–96. Institut für Theoretische Informatik, Technische Universität Braunschweig.

[Wätjen, 2002c] Wätjen, D. (2002c). *Kryptologie*, pages 100–104. Institut für Theoretische Informatik, Technische Universität Braunschweig.