ABSTRACT

KERBEROS SECURE PHONE MESSENGER

by
Nabeel Al-Saber

Security is becoming vital in today's open insecure Internet. While popular Internet enabled mobile devices are spreading widely, the security of such platforms is not maturely addressed. This research extends the popular *Kerberos* authentication protocol to run on mobile phones and builds a novel Kerberos Secure Phone Messenger (KSPM) on top of the protocol. Moreover, the Kerberos network authentication protocol provides user authentication and message privacy with the convenience of secret key cryptography. Such an advantage in mobile phones helps reduce the computational burden and power consumption if compared with public key cryptography. KSPM achieves high standards in terms of security, performance and power consumption. This thesis explains Kerberos authentication and illustrates the software implementation of the protocol and KSPM. Furthermore, it analyzes the performance and power consumption required by KSPM.

KERBEROS PHONE SECURE MESSENGER

by
Nabeel Al-Saber

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Engineering

Department of Electrical and Computer Engineering

January 2008

APPROVAL PAGE

KERBEROS PHONE SECURE MESSENGER

Nabeel Al-Saber

12/10/07

Dr. Sotirios G. Ziavras, Thesis Advisor — Date
Professor of Electrical and Computer Engineering, NJIT

12/10/07

Dr. Roberto Rojas-Cessa, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

12/10/07

Dr. Jie Hu, Committee Member — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Nabeel Al-Saber

Degree: Master of Science

Date: January 2008

Undergraduate and Graduate Education:

- Master of Science in Computer Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2008

- Bachelor of Science in Computer Engineering,
  University of Jordan, Amman, Jordan, 2006

Major: Computer Engineering

Presentations and Publications:

Majid A. Al-Taee, Omar B. Khader and Nabeel A. Al-Saber,
"Remote Monitoring of Vehicle Diagnostics and Location Using a Smart Box with Global Positioning System and General Packet Radio Service,"
IEEE/ACS International Conference on Computer Systems and Applications (AICCSA 2007), Amman, Jordan, pp. 385-388, May 2007.

This project is dedicated to my family, who gave me all the love and support to accomplish this tough mission. It is dedicated also to all my beloved ones who stood behind me and charged me with enough power to run this long rugged journey

ACKNOWLEDGMENT

It is difficult to express my gratitude to Professor Constantine Manikopoulos. With his wisdom, his inspiration, and his dedication, he guided me to fulfill this project. Even when he was going through his disease, he kept helping me until the last moments of his life.

I would like also to express my dearest appreciation and admiration to Professor Sotirios Ziavras. Throughout my second thesis semester, he provided brilliant ideas, encouragement, sound advice, supervision, and good company. I would have been lost without him.

Special thanks are given to Professor Roberto Rojas-Cessa for sharing his knowledge and wisdom with us during the courses I took with him. I also wish to thank Professor Jie Hu for serving on my defense committee. Finally, I would like to express my gratitude to many people who have influenced me and helped in carrying out this project.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

1.1 Motivation

Today, most of popular instant messaging networks are designed for scalability and performance rather than security. The use of instant messaging in the workplace creates potential threats to corporate computer security. Therefore, sensitive data should not be exchanged over insecure instant messengers. The instant messaging architecture does not provide a means for authenticating users or verifying that a message really originated from the sender. Hence, a hacker can not only inject messages into an ongoing chat session, but can also hijack an entire session by impersonating one of the users.

To address these security issues, major instant messaging network providers like Microsoft, AOL, and Yahoo have announced corporate versions of their products. The corporate versions encrypt data transmitted over the network and provide additional functionalities such as central logging, user access controls, and corporate screen names.

As security is being recently added to desktop instant messaging, mobile phones are not yet targeted by these new secure messengers. While smart phones are becoming very popular with WiFi support, more security should be considered for internet applications. Nowadays, many corporations are using these smart phones for shipments handling and other purposes. Hence, in the near future security will become the first priority for these platforms.

The Kerberos Secure Phone Messenger (KSPM) introduced in this thesis is the first mobile phone messenger that provides Kerberos security. Kerberos authentication protocol is known as one of the best secure protocols in the literature. The main

advantage of Kerberos over most of the known security protocols is using symmetric key cryptography. As this messenger is targeted for resource limited platforms (mobile phones), symmetric key cryptography is the best choice for providing security.

1.2 Objectives

This research intends to build a secure messenger for mobile phones using Kerberos protocol. The application should not only be secure but it should also be designed to provide performance efficiency. The resulting application is designed to provide functionalities for home users and corporations users.

The main objectives behind this thesis are the following:

- Implement Kerberos security protocol on mobiles.
- Build KSPM using Kerberos protocol.
- Analyze the performance, memory and power requirements for KSPM.

1.3 Organization

The thesis is organized as follows: Chapter 1 states the motivation and objectives behind this research. Chapter 2 examines the literature review of the Kerberos protocol, it's advantages and benefits. Chapter 3 discusses the software implementation of the project. Chapter 4 analyzes the performance, memory and power requirements for KSPM. Finally, Chapter 5 presents conclusions and future research directions.

CHAPTER 2

KERBEROS PROTOCOL

The Kerberos Network Authentication Service [1, 2, 3, 4, 5, 6, 7] provides the means of verifying the identities of principals on an open, potentially insecure network. It allows individuals communicating over an insecure network to prove their identity to one another in a secure manner. The Kerberos Authentication Service was developed by the Massachusetts Institute of Technology (MIT) to protect the emerging network services provided by Project Athena [2]. Versions 1 through 3 were used internally. Although designed primarily for use by Project Athena, Version 4 of the protocol has achieved wide spread use beyond MIT. Version 5, defined in RFC 1510, of the Kerberos protocol incorporates new features suggested from experience with Version 4, making it useful in more situations. Version 5 was based in part upon input from many contributors familiar with Version 4.

Kerberos prevents eavesdropping or replay attacks, and ensures the integrity of the data. Its designers aimed primarily at a client-server model, and it provides mutual authentication where both the user and the server verify each other's identity [3]. This chapter describes Kerberos model and basic protocol exchanges.

2.1 Kerberos Infrastructure

The Kerberos Key Distribution Center (KDC), tickets, keys, and other terminologies are explained in this section. Kerberos works on the basis of "tickets" which serve to prove the identity of users. The KDC maintains a database of secret keys; each entity on the

network, whether a client or a server, shares a secret key known only to itself and to the KDC. Knowledge of this key serves to prove an entity's identity. For communication between two entities, the KDC generates a session key which they can use to secure their interactions. The three main parts of the Kerberos Physical infrastructure are: KDC, client user and server with the desired service to access.

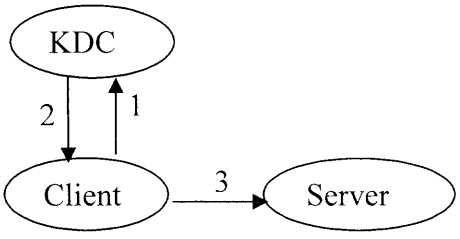


Figure 2.1 Kerberos authentication [3].

2.1.1 Key Distribution Server (KDC)

KDC is the heart and soul of the complete Kerberos infrastructure. The KDC consists of three logical components [4]: a database of all principals and their associated encryption keys, the *Authentication Server*, and the *Ticket Granting Server*. While each of these components is logically separate, they are usually implemented in a single program and run together in a single process space.

In a given Kerberos realm, there must be at least one KDC. Vital data including the secrets for every principal in the realm is located on every KDC in the network. Thus, it is critical that those servers should be as secure as possible. Each KDC contains a database of all of the principals contained in the realm, as well as their associated secrets. Most KDC software also stores additional information for each principal in this database,

such as password lifetimes, last password change, and more. Windows 2000 and 2003 keep this database in the Active Directory, its LDAP store.

2.1.1.1 KDC Architecture. KDC is divided into two parts based on functionality. Each part provides services to the other part. The two parts are the Authentication Server (AS) and Ticket Granting Server (TGS).

Authentication Server (AS) [4] issues an encrypted *Ticket Granting Ticket* (also known as a TGT) to clients who wish to login to the Kerberos realm. The client does not have to prove its identity to the KDC; instead, the TGT that is sent back to the client is encrypted in the user's password. Since only the user and the KDC know the user's password, when the login process attempts to decrypt the ticket using the password supplied by the user, only the correct password will correctly decrypt the ticket. If an incorrect password is used, the ticket will decrypt into garbage, and the user is prompted to try again. The TGT returned by the Authentication Server can then be used, once decrypted by the client, to request individual service tickets. The TGT is the crucial piece that eliminates the requirement for a user to retype their password for each subsequent service they contact.

Ticket-Granting Service (TGS) [4] issues tickets for admission to other services in the TGS's domain. When a client wants access to a service, it must contact the ticket-granting service in the service's account domain, present a TGT, and ask for a ticket. The Ticket Granting Server takes in two pieces of data from the client: a ticket request that includes the principal name representing the service the client wishes to contact, and a Ticket Granting Ticket that has been issued by the Authentication Server. The TGS verifies the TGT is valid and then issues the user the service ticket he requested.

2.1.1.2 KDC Advantages and Disadvantages. The main advantage of using a KDC is that it makes key distribution much easier. If any node wants to join the network, we just need to setup a Key between the node and the KDC. Also in case some node is suspected of being compromised, the setup again needs to be changed for just one place. The alternative to KDC will be for nodes to share keys between themselves depending on what service they may need to access.

Although KDC authentication has its advantages [5], it has disadvantages as well.

- The KDC has enough information to impersonate any one on the network so if it is compromised all the network resources are open to attack.
- The KDC is the single point of failure. If it goes down nobody can access anything [5].
- The KDC can become a bottleneck as far as performance is concerned, because everyone frequently needs to access it. Having multiple KDC's can alleviate this problem, but then again the complexity may be an issue.

After understanding the basics of the Kerberos architecture, the next section goes deeper and takes a look at the logical infrastructure, the key structure, the ticket types, etc. The various physical components of Kerberos contain a number of logical attributes which are of prime importance for Kerberos to work properly.

Kerberos authentication relies on several keys and key types for encryption. Key types can include long-term symmetric keys, long-term asymmetric keys, and short-term symmetric keys. The authentication protocol was designed to use symmetric encryption, meaning that the same shared key is used by the sender and the recipient for encryption and decryption.

2.1.2 Kerberos Keys

To authenticate entities, Kerberos uses symmetric key cryptography. In symmetric key cryptography, the communicating entities use the same key for both encryption and decryption. The basic mathematical equation behind this process is the following:

$$D_K(E_K(M)) = M \tag{2.1}$$

If the encryption (E) and decryption (D) processes are both using the same key K, the decryption of the encrypted text (M) results in the readable text (M). There are two types of keys in Kerberos: long-term symmetric keys and short-term symmetric keys.

The master key (long-term symmetric key) [6] is a secret key that is shared between each entity and the KDC. It must be known to both the entity and the KDC before actual Kerberos protocol communication can take place. The master key is generated as part of the domain enrollment process and is derived from the user, machine, or service's password. The transport of the master key over a communication channel is secured using a secure channel.

The master keys (long-term symmetric keys).

- **User keys** -When a user is created, his/her password is used to create the user key. In the KDC domain, the user key is stored with the user's object in the KDC. At the workstation, the user key is created when the user logs on [6].
- **System keys** - When a workstation or a server joins a Kerberos domain, it receives a password. Similar to a user account, the system account's password is used to create the system key [6].
- **Service keys** - Services use a key based on the account password used to log on. All KDC's in the same realm use the same service key [6].

Short-term symmetric keys are session keys [6]. A session key is a secret key that is shared between two entities for authentication purposes. The session key is generated by the KDC. Because it is a critical part of the Kerberos authentication protocol, it is never sent in the clear over a communication channel: It is encrypted using the master key. The session keys used for ticket-granting tickets (TGTs) and service tickets are short-lived, and used only as long as that session or service ticket is valid.

2.1.3 Kerberos Tickets

The main component of Kerberos authentication is the ticket. A Kerberos ticket [5] is an encrypted data structure issued by the KDC that includes a shared encryption key, unique to each session, along with other fields. Tickets serve two purposes: to confirm the identity of the end participants and to establish a short-lived encryption key that both parties can share for secure communications (called the session key). There are two types of tickets, used in Kerberos authentication as explained below, TGTs and service tickets.

2.1.3.1 Ticket-Granting Ticket (TGT). The KDC responds to a client's authentication service request by returning a service ticket for it. This special service ticket is called a ticket-granting ticket (TGT) [4, 6]. A TGT enables the authentication service to safely transport the requester's credentials to the ticket-granting service.

The TGTs are encrypted with a key shared by the KDC. The client cannot read tickets. Only KDC servers can read TGTs to secure access to user credentials, session keys, and other information. Like an ordinary service ticket, a TGT contains a copy of the session key that the KDC will use in communicating with the client. The TGT is encrypted with the KDC's long-term key.

Clients use the TGT to request a service ticket when accessing a certain service. Before a client attempts to connect to any service, the client first checks its credentials cache for a service ticket to that service. If it does not have one, it checks the cache again for a TGT. If it finds a TGT, the client fetches the corresponding TGS session key from the cache, uses this key to prepare an authenticator, and sends both the authenticator and the TGT to the KDC, along with a request for a service ticket.

The KDC uses the TGT to avoid the performance penalties of looking up a user's long term key every time the user requests a service. The KDC looks up the user's long-term key only once, when it grants an initial TGT. For all other exchanges with this client, the KDC can decrypt the TGT with its own long-term key, extract the session key, and use that to validate the client's authenticator.

2.1.3.2 Service Tickets. A service ticket [4, 6] enables the ticket-granting service (TGS) to safely transport the requester's credentials to the target server or service. The KDC responds to the client's request to connect to a service by sending both copies of the session key to the client. The client's copy of the session key is encrypted with the key that the KDC shares with the client. The service's copy of the session key is embedded, along with information about the client, in a data structure called a service ticket. The entire structure is then encrypted with the key that the KDC shares with the service. The ticket is the client's responsibility to manage until it contacts the service.

A service ticket is used to authenticate with services other than the TGS and is meant only for the target service. A service ticket is encrypted with a service key, which is a long-term key shared by the KDC and the target service. Thus, although the client manages the service ticket, the client cannot read it. Only the KDC and the target service

can read tickets, enabling secure access to user credentials, the session key, and other information. One thing to note here is that the KDC is simply providing a ticket-granting service. It does not keep track of its messages to make sure they reach the intended address. No harm will be done if the KDC's messages fall into the wrong hands. Only someone who knows the client's secret key can decrypt the client's copy of the session key. Only someone who knows the server's secret key can read what is inside the ticket. When the client receives the KDC's reply, it extracts the ticket and the client's copy of the session key, putting both aside in a secure cache (located in volatile memory, not on disk). When the client wants admission to the server, it sends the server a message that consists of the ticket, which is still encrypted with the server's secret key, and an authenticator, which is encrypted with the session key. The ticket and authenticator together are the client's credentials to the server. To guard against the possibility that someone might steal a copy of a ticket, service tickets have an expiration time that is specified by the KDC in the ticket's data structure.

2.1.3.3 Client's Tickets Information. A client needs to have some information about what is inside tickets and TGTs in order to manage its credentials cache. When the KDC returns a ticket and session key as the result of an authentication service (AS) or ticket-granting service (TGS) exchange, it packages the client's copy of the session key in a data structure that includes the information in the following ticket fields: Authentication Time, Start Time, End Time, and Renew Till [6]. In order to reduce the risk that a ticket or the corresponding session key might be compromised, administrators can set the maximum lifetime for tickets. The maximum lifetime for ticket setting is an element of the Kerberos policy.

2.1.4 Realms, Principals, and Instances

Every entity contained within a Kerberos installation, including individual users, computers, and services running on servers, has a *principal* [4] associated with it. Each principal is associated with a long-term key. This key can be, for example, a password or pass phrase. Principals are globally unique names. To accomplish this, the principal is divided into a hierarchical structure.

Every principal starts with a username or service name. The username or service name is then followed by an optional instance. The *instance* [4] is used in two situations: for service principals (which we'll discuss later), and in order to create special principals for administrative use. For example, administrators can have two principals: one for day-to-day usage and another (an "admin" principal) to use only when the administrator needs elevated privileges.

The username and optional instance, taken together, form a unique identity within a given *realm* [4]. Each Kerberos installation defines an administrative realm of control that is distinct from every other Kerberos installation. Kerberos defines this as the realm name. By convention, the Kerberos realm for a given DNS is the domain converted in uppercase.

2.2 Kerberos Operation

Before going into the details of Kerberos message exchange, we look at how basic Kerberos authentication works. In the next section, each and every step is explained in detail.

The Kerberos authentication steps [1] (Figure 2.2):

- A client authenticates itself to the KDC by sending the pre authentication data.
- The KDC sends the TGT which can be used by the client to authenticate itself in the following transactions.
- A client sends a request to the authentication server (AS) for the "credentials" of a given server.
- The AS responds with these credentials, encrypted with the client's key. The credentials consist of a "ticket" for the server and a temporary encryption key or a "session key".
- The client transmits the ticket (which contains the client's identity and a copy of the session key, all encrypted with the server's key) to the server.
- The session key (now shared by the client and server) is used to authenticate the client and may optionally be used to authenticate the server. It may also be used to encrypt further communications between the two parties or to exchange a separate sub-session key to be used to encrypt further communication.

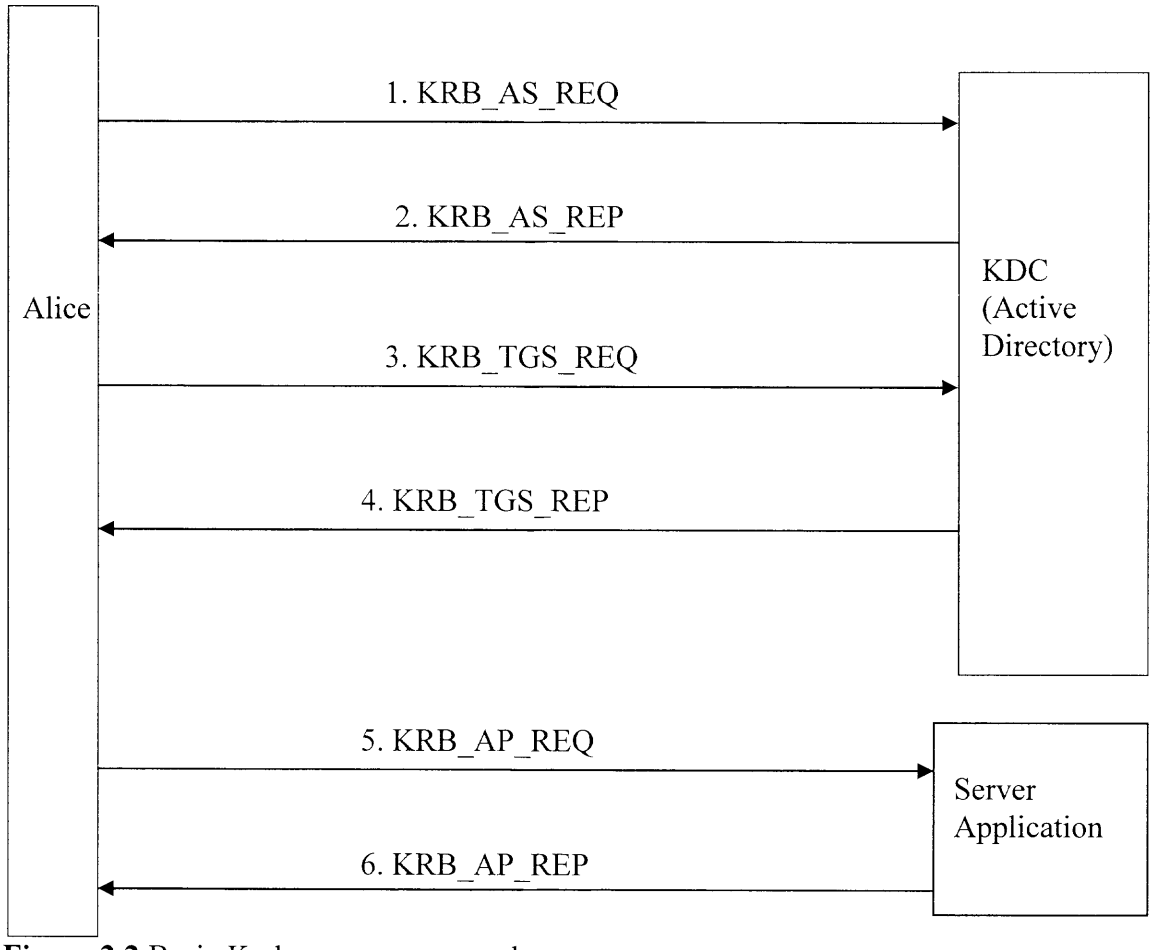


Figure 2.2 Basic Kerberos message exchange.

The Basic Kerberos exchange is divided into 3 parts:

- Authentication Server (AS) exchange
- Ticket Granting Server (TGT) exchange
- Client/ Server Exchange

2.2.1 Authentication Server (AS) Exchange

Kerberos Authentication Service Request (KRB_AS_REQ)

The client contacts the KDC's authentication service for a short-lived ticket (TGT). This is done at login. The Kerberos client on the workstation sends the message KRB_AS_REQ to the KDC.

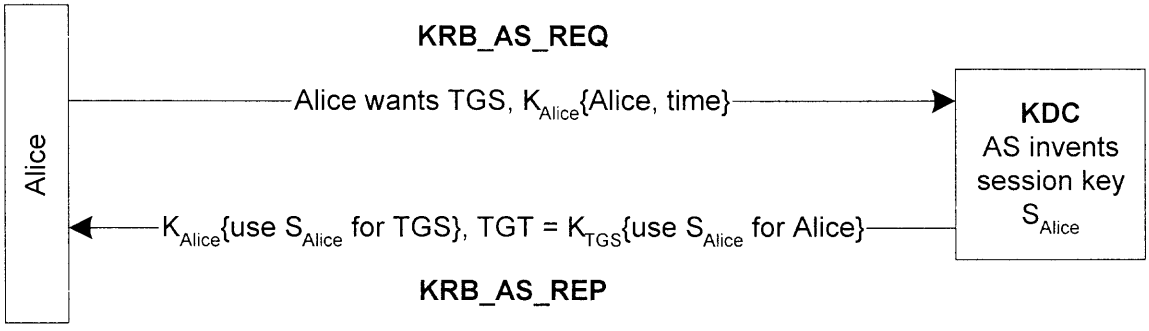


Figure 2.3 Kerberos authentication service [6].

The message includes [1]:

- The user principal name.
- The name of the account domain.
- Pre-authentication data encrypted with the user's key derived from the user's password.

The KDC has a copy of the user's key in its account database (i.e. Microsoft Active Directory). When it receives a request from the Kerberos client on the user's workstation, it takes the user key from a field in the record. This process of computing one copy of the key from a password and fetching another copy of the key from a database actually takes place only once, when a user initially logs on to the network. Immediately after accepting the user's password and deriving the user's long-term key, the Kerberos client on the workstation requests a service ticket and a TGS session key that it can use in subsequent transactions with the KDC during this logon session.

The optional pre-authentication data is used to verify the user during the login session. The KDC decrypts the pre-authentication data and evaluates the embedded timestamp. If the timestamp passes the test, the KDC can be assured that the pre-authentication data was encrypted with the user key and thus it can verify that the user is genuine. After it has verified the user's identity, the KDC creates credentials that the Kerberos client on the workstation can present to the ticket-granting service.

Kerberos Authentication Service Response (KRB_AS_REP)

The AS constructs the TGT and creates a session key that the client can use to encrypt communications with the ticket-granting service (TGS) [1]. The TGT has a limited lifetime. At the point that the client has received the TGT, the client has not been granted access to any resources, even to resources on the local computer.

The KDC replies with KRB_AS_REP [6] containing a service ticket (TGT) for itself. This TGT contains a copy of the session key that the service (KDC) will use in communicating with the user. The message that returns the TGT to the user also includes a copy of the session key that the user can use in communicating with the KDC. The TGT is encrypted with the KDC's long-term key. The user's copy of the session key is encrypted with the user's long-term key.

The message includes [1, 6]:

- A TGS session key for the user to use with the TGS, encrypted with the user key derived from the user's password.
- A TGT for the KDC encrypted with the TGS key (KDC master key). The TGT includes a TGS session key for the KDC to use with the user and authorization data for the user.

When the client receives the KDC's reply to its initial request, the client uses its cached copy of the user key to decrypt its copy of the session key. It can then discard the user key derived from the user's password, for it is no longer needed. In all subsequent exchanges with the KDC, the client uses the TGS session key. Like any other session key, this key is temporary, valid only until the TGT expires or the user logs off. For this reason, the TGS session key is often called a logon session key.

2.2.2 The Ticket-Granting Service Exchange

Kerberos Ticket-Granting Service Request (KRB_TGS_REQ)

When the client wants to access a service, it sends a request to the TGS for a ticket. This ticket is referred to as a service ticket. To get the ticket, the client presents the TGT, an authenticator, and the name of the target server.

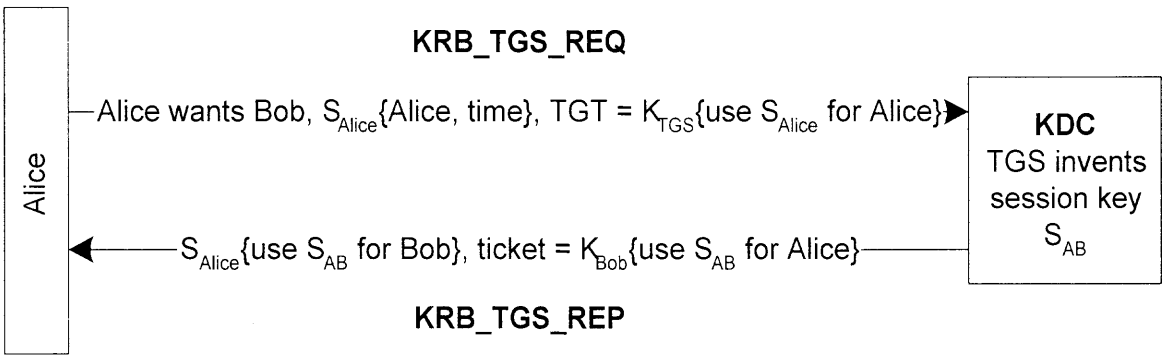


Figure 2.4 Kerberos ticket-granting service [6].

The message includes [1, 6]:

- The name of the target computer.
- The name of the target computer's domain.
- The user's TGT.
- An authenticator encrypted with the session key the user shares with the KDC.

Kerberos Ticket-Granting Service Response (KRB_TGS_REP)

The TGS examines the TGT and the authenticator. Then, the TGS creates a sub-session key for the user to share with the computer encrypted with the session key. The message also includes a service ticket to the computer, encrypted with the computer's secret key. The service ticket includes: A session key for the computer to share with the user and authorization data copied from the user's TGT.

The KRB_TGS_REP [1, 6] message includes:

- A session key for the user to share with the computer encrypted with the session key the user shares with the KDC.
- The user's service ticket to the computer, encrypted with the computer's secret key.
- The service ticket includes: A session key for the computer to share with the user, and authorization data copied from the user's TGT.

2.2.3 The Client/Server Exchange

Kerberos application server request (KRB_AP_REQ)

After the client has the service ticket, the client sends the ticket and a new authenticator to the target server (Bob), requesting access. The server will decrypt the ticket and validate the authenticator.

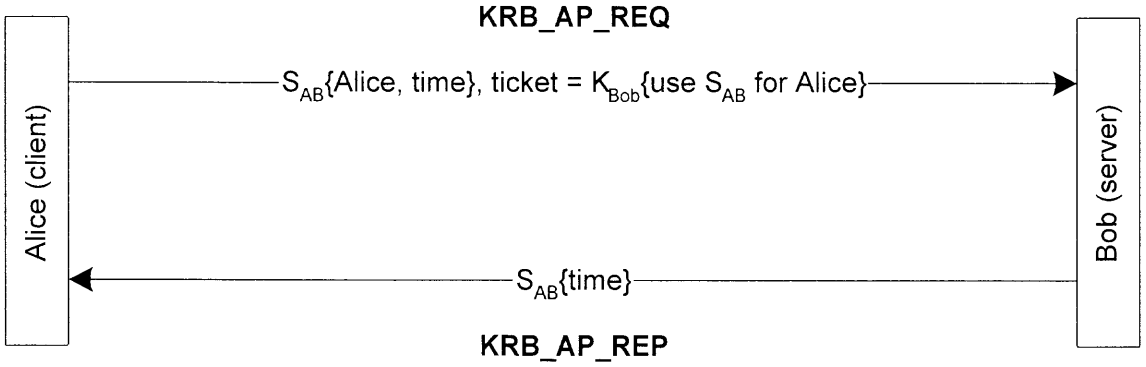


Figure 2.5 Kerberos application server [6].

This message contains [1, 6]:

- An application option flag indicating whether to use the session key.
- An application option flag indicating whether the client wants mutual authentication.
- The service ticket obtained in the TGS exchange.
- An authenticator encrypted with the session key for the service.

Kerberos application server response (optional) (KRB_AP_REP)

Optionally, the client might request that the target server verify its own identity. This is called mutual authentication. If mutual authentication is requested, the target server will take the client computer's timestamp from the authenticator, encrypt it with the session key the TGS provided for client-target server messages, and send it to the client.

If the authenticator passes the test, the service looks for a mutual authentication flag in the client's request. If the flag is set, the service uses the session key to encrypt the time from the user's authenticator and returns the result in a Kerberos application reply (KRB_AP_REP). If the flag is not set, then no response is needed. When the client on the user's workstation receives KRB_AP_REP, it decrypts the service's authenticator with

the session key it shares with the service and compares the time returned by the service with the time in the client's original authenticator. If the times match, the client knows that the service is genuine [6].

2.3 Kerberos Services and Benefits

Kerberos provides various services like authentication, authorization, data integrity and confidentiality. Moreover, there are many benefits of using Kerberos. All of this is discussed below.

2.3.1 Kerberos Services

Authentication [4] is the process of verifying the identity of a particular user. To authenticate a user, the user is asked for information that would prove his identity. This information can fall into one or more of three categories: what he knows, what he has, or what he is. Kerberos provides this using the trusted third party concept. It also provides mutual authentication by which the server trusts the client as well the client trusts the server. At a very simple level, Kerberos uses encryption technology. The user's password is utilized (while still on the user's workstation) to generate an encryption key. The key encrypts certain pieces of information that are exchanged with the KDC. After a few exchanges, the KDC returns information to the user that is usable only by software on the workstation that knows the temporary encryption key derived from the password. Now when users wish to contact a Kerberos-protected service, they first contact the Kerberos ticket-granting service and ask for a ticket to the service. A ticket is a chunk of

information that proves the user's identity to the service; but it's encrypted in the services' long-term key, so it's unintelligible to the user.

Authorization [3] refers to granting or denying access to specific resources based on the requesting user's identity. This step is performed after a user is identified through authentication. Authorization is usually performed through access control lists, which associate user identities with specific rights. Authorization includes information such as a user's group membership, user policies, and other information that determines what level of access that user has to computer or network resources. By default, Kerberos does not provide any authorization services; they are usually implemented as a separate procedure. Still authorization information can be embedded within the TGS. A usual way of doing this in Kerberos is to include access control lists in the ticket that the KDC sends to the client for the server. Once the server decrypts the ticket it has a list of services that this particular client can access and their privilege level. The server uses these ACL's to authorize the servers future requests for resources.

Data integrity [3] ensures the recipient that the message was not tampered with during transit. While encryption as used in Kerberos gives you message integrity for "free," since only the two end points have the required key to encrypt and decrypt messages, there are specialized message-integrity algorithms that can ensure message integrity without the overhead of encryption. There are several different message-integrity algorithms commonly used in Kerberos. Ranging from weaker to stronger, the message-integrity algorithms included in the MIT Kerberos distribution include CRC-32, MD5, and the Secure Hash Algorithm (SHA1).

Confidentiality [3] ensures that certain information is never disclosed to unauthorized entities. Sometimes, you need to know that the conversation is completely private. A more technical term for privacy is "data confidentiality" and once again Kerberos addresses this need. Kerberos provides services that encrypt the entire plaintext message and (optionally) computes a one-way hash of the cipher text. The sender transmits the package to the receiver, who decrypts the cipher text and (optionally) verifies the authenticity of the data. Usually, if we are encrypting the whole message for confidentiality we use the data integrity feature as well.

2.3.2 Kerberos Benefits

Kerberos has a number of advantages as an authentication protocol. This section lists a number of reasons which act as the main driving points for using Kerberos.

1) Faster Authentication

The Kerberos protocol uses a unique ticketing protocol that provides faster authentication [7]. Every authenticated domain entity can request tickets from its local Kerberos KDC to access other domain resources. The tickets are considered as access permits by the resource servers. The ticket can be used more than once and can be cached on the client side. Hence, Kerberos is described as a *Single Sign On* protocol. The use of tickets makes the re-authentication of the client much easier. Once a client has a ticket from the KDC, it can be used again and again for authentication.

2) Mutual Authentication

Kerberos supports mutual authentication where not only the client authenticates itself to the server but the server can also authenticate to the client [7]. So there is no assumption made that the servers are always trustworthy.

3) Open Source

Kerberos is an open source protocol, thus essentially free to use [2]. Because of being open source, there is also much faster development going on all the fronts and it is being used and tested by large user base.

4) Support for Authentication Delegation

Delegation means that user A can give rights to an intermediary machine B to authenticate to an application server C as if machine B was user A. This means that application server C will base its authorization decisions on user A's identity rather than on machine B's account. Delegation is also known as authentication forwarding. In Kerberos terminology, this basically means that user A forwards a ticket to intermediary machine B, and that machine B then uses user A's ticket to authenticate to application server C [7].

5) Support for Public Key Cryptography and One Time Pass Codes

Kerberos can support the use of public key cryptography for authentication of the client to the KDC; this way the password guessing/stealing attacks can be minimized. Another solution that Kerberos provides for such issues is to use one time passcodes or smart

cards where each time to authenticate the user requires a different password so that a password guessing mechanism or a Trojan horse program will never work as the password changes at each login.

CHAPTER 3

SOFTWARE IMPLEMENTATION

This chapter describes the software that empowers the system. Several object-oriented modeling techniques are presented in this chapter. This includes class diagrams, members and methods.

3.1 Kerberos Secure Phone Messenger (KSPM) Architecture

3.1.1 Instant Messaging Architecture

Instant messaging follows the client-server architecture. Communication between clients occurs either via a server, or a server brokers it. This section describes the general architecture of instant messaging networks and clients. There are two main ways messages and files are transferred between clients: server proxy and server broker.

1) Server Proxy

In the server proxy architecture [8], all instant messaging communication is passed through the server. For example, if two users, Alice and Bob, want to exchange messages via instant messaging, they would not send the messages directly to each other. Instead, the messages would first be sent to the server. The server would then forward the message to the intended recipient. In this case, the server acts as a proxy between the users.

The benefit of this method is that both clients initiate outgoing connections to the server and either one does not require the ability to accept incoming connections on a

port that a corporate firewall may block. However, sending messages to the server may incur a time delay. In general, this is the default method that all major instant messaging networks use today.

2) Server Broker

In the server broker architecture [8], the only packets that are sent to the server are packets requesting the server to initiate communication between two clients. The server essentially facilitates the connection between the two clients. The server provides the clients with the connection information; the clients then directly connect to one another.

For example, if Alice wants to send a message to Bob, Alice will send a request to the server to initiate the session. The server will notify Bob that Alice wishes to chat with him. If Bob agrees, he replies to the server with his contact information (typically an IP address and port number) and this information is forwarded to Alice. Then, Alice can directly connect to Bob and messages between the two do not pass via the server.

This method reduces the load on the server and reduces the privacy risk, as potentially confidential messages are no longer sent to the central server. However, this method is often blocked by firewalls, as they are usually configured to not allow incoming connections. Yahoo! Instant Messenger (YIM) is an example of a client that uses server brokering. YIM will send the first message via the server and then attempt a direct connection. If the direct connection fails, YIM will continue to send messages via the server.

3.1.2 Kerberos Secure Phone Messenger (KSPM) Architecture

KSPM looks like a typical phone to phone messenger for the users. The underlying technology for this messenger is based on Kerberos authentication. Kerberos provides a mechanism for mutual authentication between a client and a server on an open insecure network. All communications after the authentication will be encrypted and secure.

KSPM is divided into two main parts. The first part is the messenger running on the phone. The second part is the server messenger application running on the server. All communications from phone to phone go through the messenger server as described in the server proxy architecture. After the clients authenticate with the active directory using Kerberos, the keys are passed to the messenger server. When the client wants to communicate with another client, the message is sent to the messenger server. The messenger server application decrypts the message using the sub-session key for client one. Then, it encrypts the message using the sub-session key for client two and sends it. The server also keeps a log of all communications between the clients. Figure 3.1 shows the authentication with the active directory, and the communications between the clients and server.

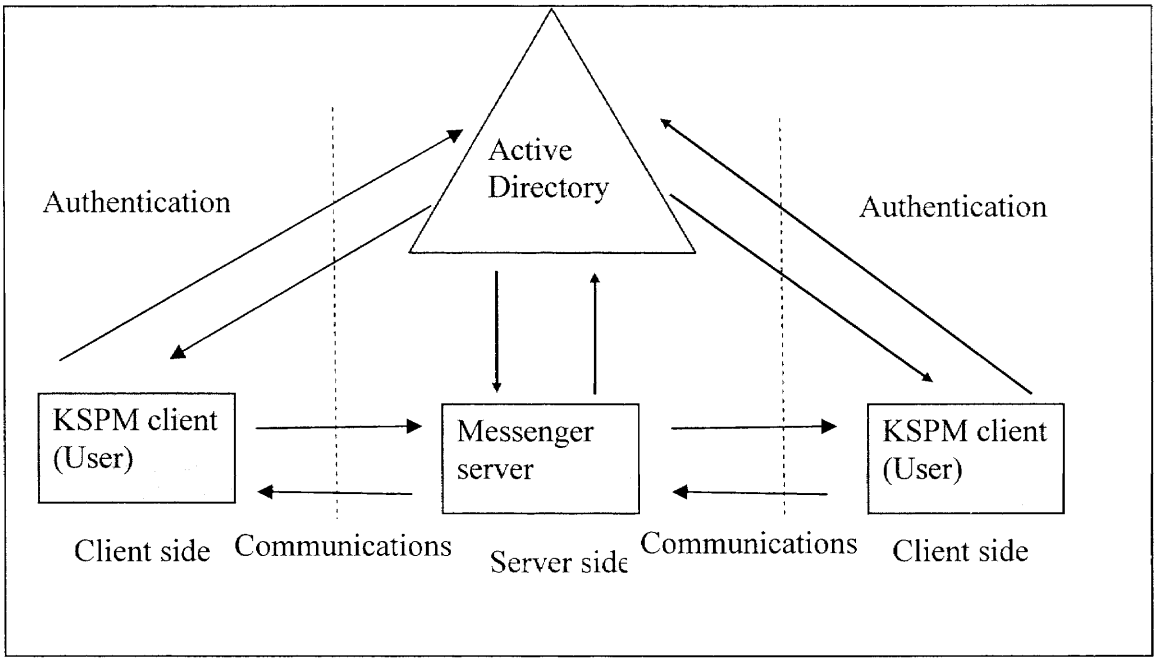


Figure 3.1 Application architecture.

3.2 KSPM Software Implementation

KSPM was written using the Java programming language. The java version for mobile phones platform is called Java 2 Platform Micro Edition (J2ME). It is designed for small, resource-constrained devices such as cell phones. The phone operating system is windows mobile version 5.0. In order to run Java, a virtual machine is required. The only available smartphone JVM in the market is WebSphere Everyplace Micro Edition Java (WEME) by IBM [9]. The Kerberos protocol basic functions were written using Connected Limited Device Configuration (CLDC) [10]. CLDC defines the base set of application programming interfaces and a virtual machine for resource-constrained devices. However, the newer version Connected Device Configuration (CDC) supports more complex libraries. KSPM is written using CDC to provide advanced functionalities and a better user interface while still being able to reuse the Kerberos CLDC version. The

KSPM application has four classes: Kerberos client, Kerberos key, Kerberos ticket, main and communications. The following sections describe the implementation of the software.

3.2.1 KSPM Kerberos Client

- **Kerberos Client** Class

Import: BounctyCastle.crypto

Extends: ASN1DataTypes

This class implements the main functions of the Kerberos authentication protocol. All other sub-functions are used in these main functions. It is responsible for establishing authentication, processing tickets and keys, and encrypting/decrypting messages. Figure 3.2 shows the class diagram and its members.

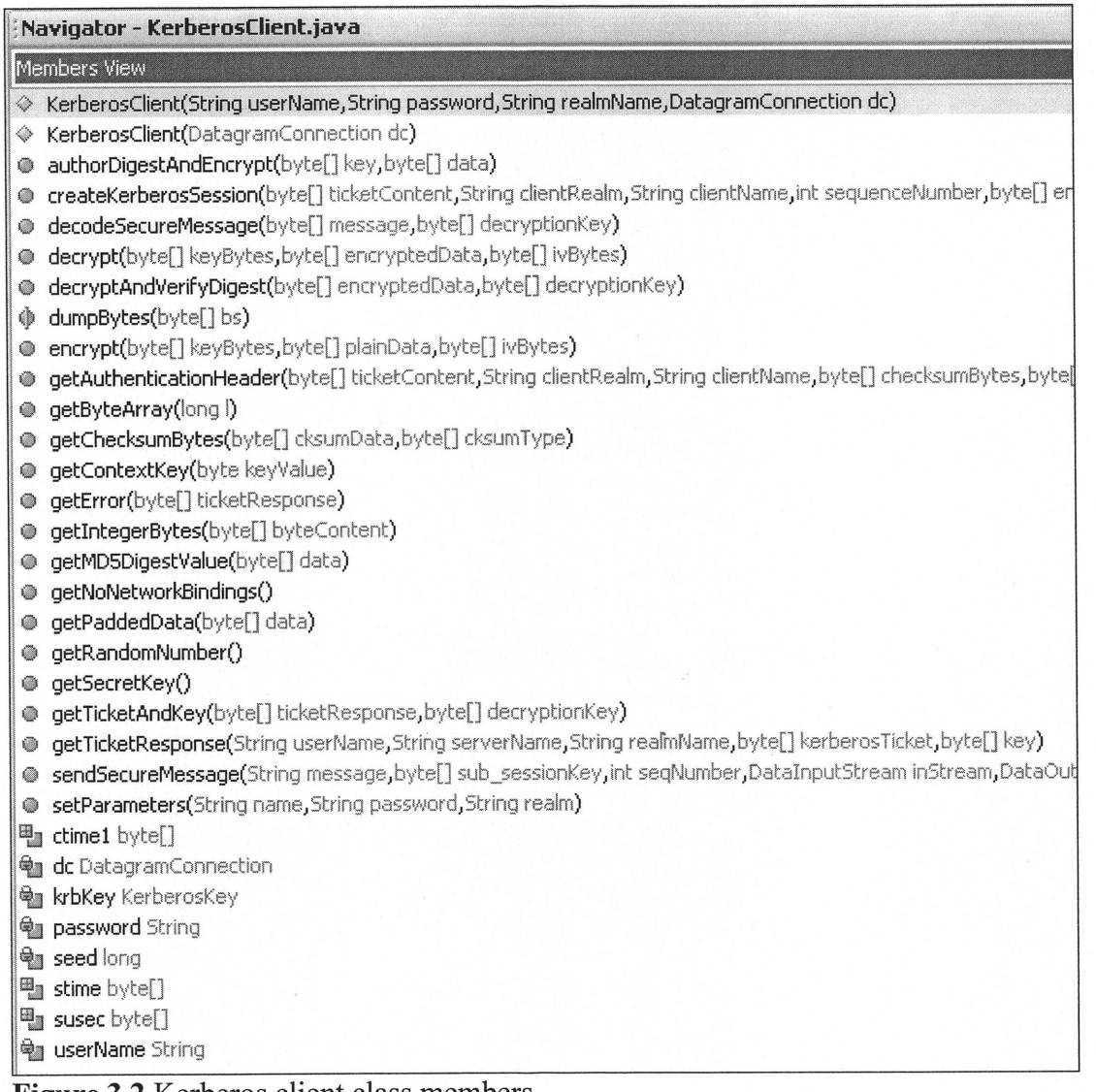


Figure 3.2 Kerberos client class members.

Table 3.1 describes briefly the main functions and the sequence they are used in to establish a connection. These functions implement Kerberos authentication messages described in Chapter 2.

Table 3.1 Main Functions Description

Function name	Description
getTicketResponse	Authoring a TGT request, getting TGT ticket
getTicketAndKey	Getting session key from TGT response
getTicketResponse	Authoring the request body, getting service ticket (TGS)
getTicketAndKey	Getting sub-session key from TGS response
createKerberosSession	Create session
decodeSecureMessage	Decode and decrypt message
sendSecureMessage	Encrypt and send message

Authoring and Processing a TGT Request

The basic purpose of the *getTicketResponse* function is to author a request for a Kerberos ticket (a TGT or a service ticket), send the ticket request to the Kerberos server, get a response from the server, and extract the ticket and the session key from the response. In other words, the first time this method is called, it authors the KRB_AS_REQ message and extracts the ticket and the session key from the KRB_AS_REP message. Notice steps 1 and 2 in Figure 3.3.

Processing and Authoring a Service Ticket

After the TGT response is processed to extract the TGT and the session key, a request for service ticket from the KDC server is made. The same TGT request (*getTicketResponse* method) is used to obtain a service ticket. This time the method is called to author a KRB_TGS_REQ message, and extract the service ticket and sub-session key from the KRB_TGS_REP message. Notice steps 3 and 4 in Figure 3.3.

Processing Error Messages

The method *getError* catches any error messages received by the server. It processes the message, then displays the error description. The Kerberos error codes are well defined in [1]. Using these error codes, the reason of the error can be discovered and resolved easily. For example, the time synchronization error occurs very frequently when authenticating with Windows. The time should be corrected and the rejected message should be sent again stamped with the server's time. Notice the error message between steps 3 and 4 in Figure 3.3

Creating a Kerberos Session

The sub-session key and the service ticket are the two things required to establish a secure communications context with the messenger server. At this point the Kerberos client must author a context establishment request intended for the messenger server. The *createKerberosSession* method handles the following aspects of establishing a secure communication context with the messenger server: authoring the context establishment request, sending the request to the server, fetching a response from the server, parsing the response to check whether the remote server has agreed to the context establishment request, and returning if the session establishment was successful or not. Notice messages KRB_AP_REQ and KRB_AP_REP in steps 5 and 6 in Figure 3.3.

Sending a Secure Message to the Messenger Server

The *createKerberosSession* method returns a true value if the client has successfully established a secure session with the remote Kerberos server. To exchange messages with the messenger server use the *sendSecureMessage* method. This method takes the following parameters: a plain text message, a cryptographic key, a sequence number (which uniquely identifies the message being sent), and input and output stream objects to exchange data with the server. The *sendSecureMessage* method authors a secure message, sends the message to the server over the output stream, listens for a response from the server, and returns the server's response.

The message sent to the server is secured using the sub-session key. This means that only the intended recipient (the messenger server, which has the sub-session key) is capable of decrypting and understanding the message. Moreover, the secure message contains message integrity data, so the messenger server can verify the integrity of the message coming from the client.

Decoding the Server Message

The messenger server reply is secure; only the client possessing the sub-session key can decrypt the message. Method *decodeSecureMessage* takes a secure message along with a decryption key, decrypts the message, checks the data integrity and returns the plain text form of the message.

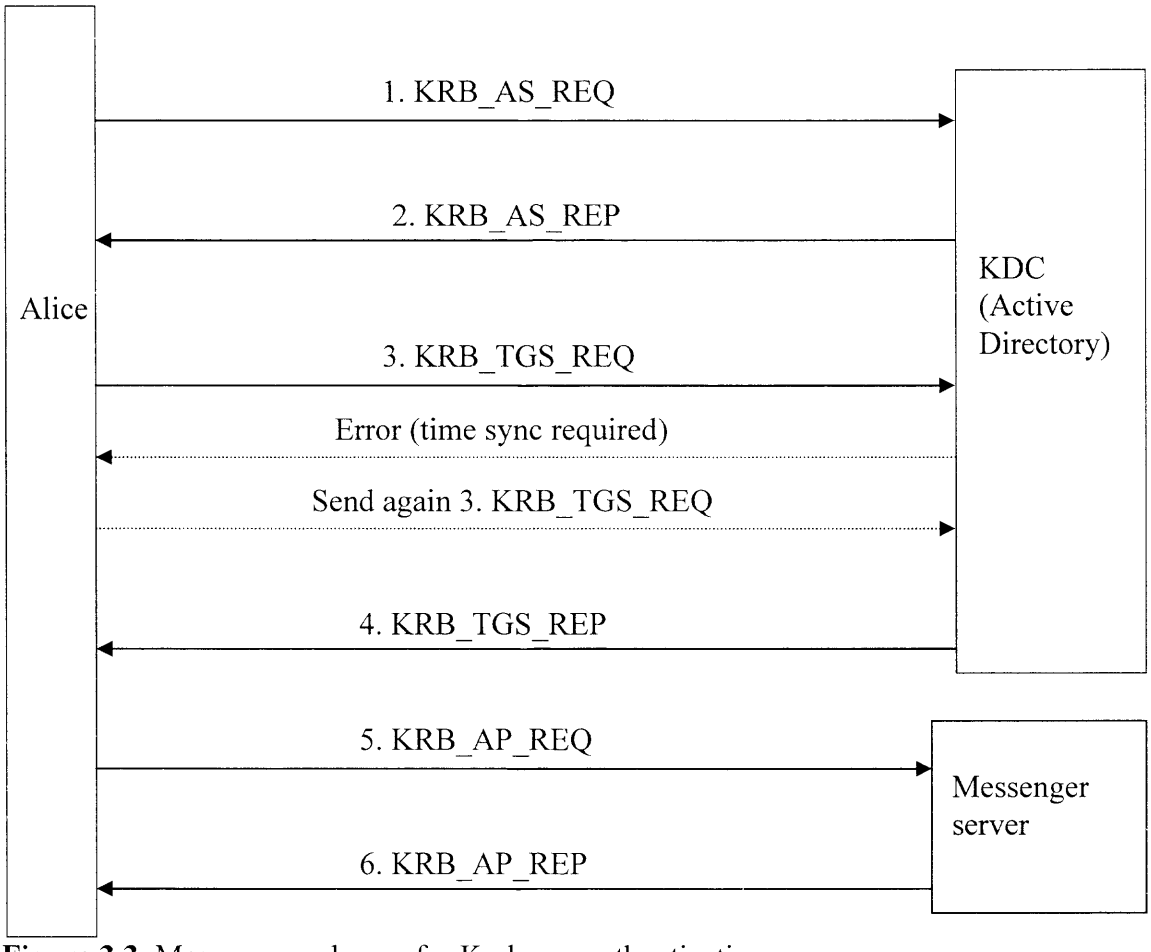


Figure 3.3 Messages exchange for Kerberos authentication.

3.2.2 KSPM Kerberos Key

- **Kerberos Key** Class

Import: BouncyCastle.crypto

The class has all the functions required to handle Kerberos keys. It is responsible for generating the user master key which is derived from the user's password. Figure 3.4 shows the class diagram.

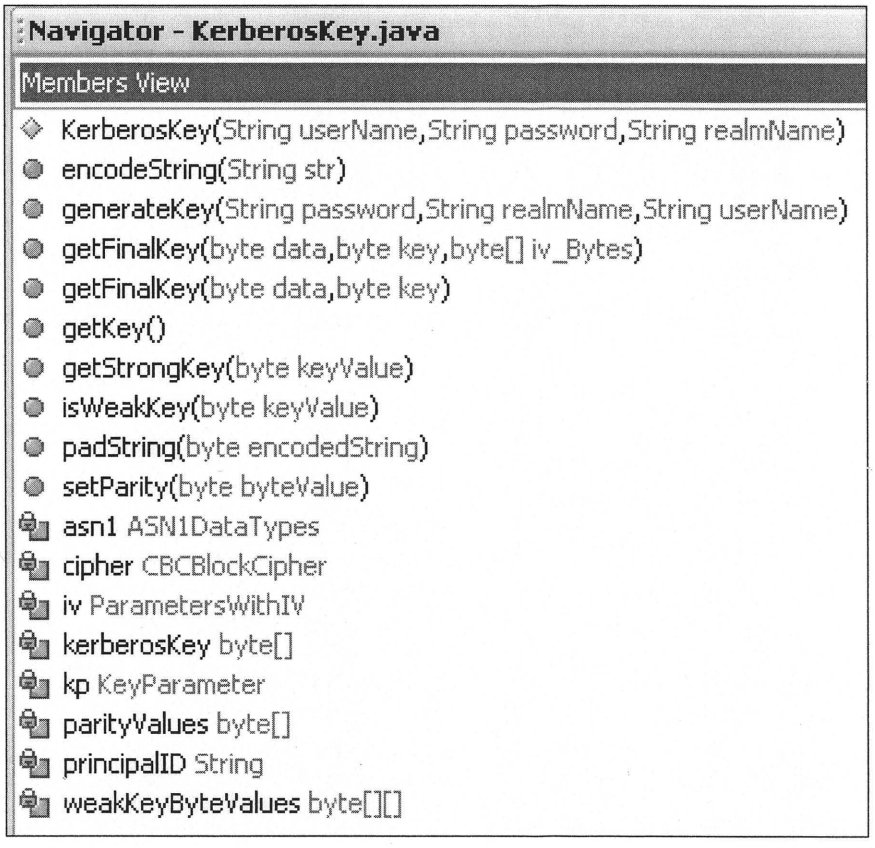


Figure 3.4 Kerberos key class members.

Kerberos defines an algorithm for processing a user's password to produce a secret key. On the Kerberos client, the Bouncy Castle cryptographic library [11] is used for encryption and for generating a secret Kerberos key from the user's password. Bouncy Castle is a collection of APIs (Application Programming Interfaces) used in cryptography.

For the J2ME-based Kerberos client, DES (data encryption standard) is used in the CBC (cipher block chaining) mode. DES is an encryption algorithm where the data to be encrypted (plain text) and the secret key are passed as inputs to the encryption process. The key and the plain text are processed together according to the DES algorithm to produce the encrypted (cipher text) form of the plain text data.

3.2.3 KSPM Kerberos Ticket

- **Kerberos Ticket and Key** Class

This class stores Kerberos tickets and keys. It functions as a cache for the retrieved tickets during authentication. Whenever keys or tickets (TGT and service tickets) are required, they can be retrieved from this cache. Figure 3.5 shows the class diagram.

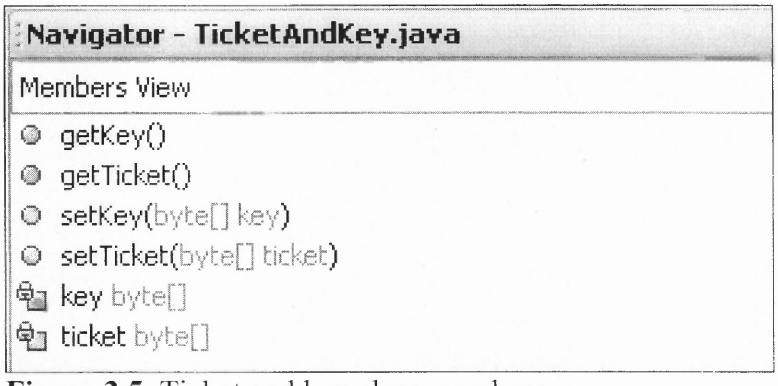


Figure 3.5 Ticket and key class members.

3.2.4 KSPM Main and Communications

- **Kerberos Main** Class
- **Kerberos Receive** Thread
- **Kerberos Send** Thread

The main class runs all other classes, initiates send and receive connection threads and also draws the user interface. The connection parameters, such as the user name, user password, realm, server name, server IP address and service ports, are set in this class. Parameters like the user name, user password, and realm are inserted at run time by the GUI interface. Other parameters like the server IP address and port number are hard-coded.

Connections in this class are separated into two groups:

- The first connection authenticates the client with the server (Active Directory). It establishes a UDP connection to a well known service port number 88 in the Microsoft server. After the authentication is done, the connection is closed but it can be reopened if the client requires any new tickets from the server. In this connection, the first four authentication messages are exchanged as shown in Figure 3.3.

- The second connection is between the client and the messenger server. It establishes a TCP connection to a predefined port number. Then, the connection is migrated to another port to keep the first port free for other clients' connection requests. Further messages between clients are exchanged using this connection.

Before the connection takes place, the user's name and password should be already stored in the Active Directory on the server. The messenger server should also be running to complete the connection. The next section will provide an overview of the messenger server implementation.

3.3 Messenger Server Implementation

```
Import: javax.security.auth
        java.net.ServerSocket
        org.ietf.jgss
```

The messenger server runs on the Windows server as an intermediate point between clients. It was developed using Java 2 Platform Enterprise Edition (J2EE). The application consists of five classes described later in this section. Figure 3.6 shows the class diagram.

Figure 3.6 Messenger server application classes and members.

Figure 3.7 shows the main structure of the application. The application is built using object oriented approach. Each class has a certain function as explained later. Calls between classes are also clear in the figure. The class *MessengerServerAuth* can initiate multiple clients by calling the *RunMessengerClientAuth* class.

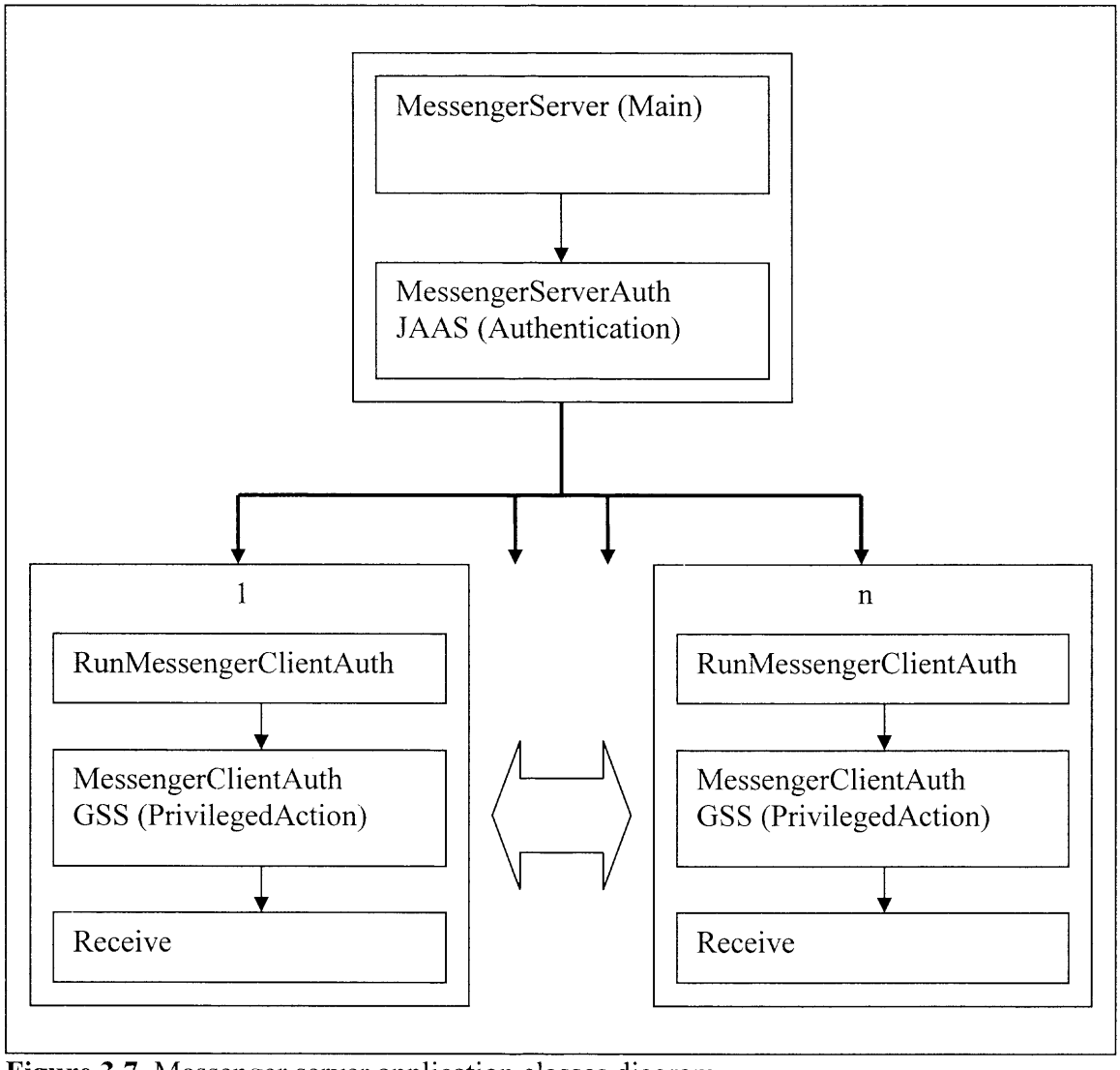


Figure 3.7 Messenger server application classes diagram.

3.3.1 Messenger Server Class

- **MessengerServer** Class

Extends: javax.swing.JFrame

This is the main class. It initializes and runs the JFrame display. It also calls the *MessengerServerAuth* class to start the authentication process.

3.3.2 MessengerServerAuth Class

- **MessengerServerAuth** Class

This class authenticates and authorizes the messenger server with active directory using the Java Authentication and Authorization Service (JAAS) [12]. JAAS can be used for two purposes: for the authentication of users, to reliably and securely determine who is currently executing Java code, and for the authorization of users to ensure they have the access control rights (permissions) required to do security-sensitive operations.

The JAAS uses the LoginContext class to provide the basic methods used to authenticate Subjects. It also allows an application to be independent of the underlying authentication technologies. The LoginContext consults a configuration that determines the authentication services or LoginModules configured for a particular application.

Kerberos Login Module

The class com.sun.security.auth.module.Krb5LoginModule is Sun's implementation of a login module for the Kerberos version 5 protocol. After authentication, the TGT is stored in the Subject's private credentials set and the Kerberos principal is stored in the Subject's principal set. The login modules invoked by JAAS must be able to get information from the caller for authentication using Callback handler. For example, the Kerberos login module may require users to enter their Kerberos password for authentication. After authenticating using JAAS, the class starts the *MessengerClientAuth* class whenever a new client wants to communicate with the messenger server.

3.3.3 MessengerClientAuth Class

- **MessengerClientAuth** Class

Implements: java.security.PrivilegedAction

The Java Generic Security Services Application Program Interface GSS-API [12] defined in RFC 2853 is used for securely exchanging messages between communicating applications. The GSS-API offers application programmers uniform access to security services atop a variety of underlying security mechanisms, including Kerberos.

The GSS-API is used in this class to start a Kerberos server that can accept incoming authentication requests from clients; provided that the client has already authenticated with the active directory and obtained the required credentials to authenticate with the messenger server. When the GSS-API creates the session as described in the steps below, the class starts a Receive class to keep listening for any incoming packets from the client.

Generic Security Services Application Program Interface (GSS-API)

The Java GSS-API framework security related functionality is obtained from GSSManager. The GSSManager can be used to configure new providers [12]. The GSSManager also serves as a factory class for three important interfaces: GSSName, GSSCredential, and GSSContext. These interfaces are described below with the methods to instantiate their implementations.

The GSSName Interface

Sun's implementation of the GSSName interface is a container class. Implementation of GSSName is similar to the principal set stored in a Subject. It may even contain the same elements that are in a Subject's principal set, but its use is restricted in the context of Java GSS-API [12]. For instance a Kerberos V5 mechanism provider might map this name to Bob@SERVER.COM where SERVER.COM is the local Kerberos realm.

The GSSCredential Interface

This interface encapsulates the credentials owned by one entity. The Server's credential requested is one that can accept incoming requests [12]. Moreover, servers are typically long lived and like to request a longer lifetime for the credentials. The secret key of the server is stored as an instance of a subclass of javax.security.auth.kerberos.KerberosKey.

The GSSContext Interface

The GSSContext is an interface whose implementation provides security services to the two peers [12]. This returns an initialized security context on the acceptor's side. At this point it does not know the name of the peer (client) that will send a context establishment request or even the underlying mechanism that will be used. However, if the incoming request is not for service principal represented by the credentials serverCreds, or the underlying mechanism requested by the client side does not have a credential element in serverCreds, then the request will fail.

Before the GSSContext can be used for its security services, it has to be established with an exchange of tokens between the two peers. Each call to the context

establishment methods will generate a token that the application must send to its peer. Once the security context is established, it can be used for message protection.

3.3.4 Receive Class

- Receive Class

Extends: Thread

This class is called as a thread to receive the messages from the server. The messages are encrypted with Kerberos protocol sub-session keys. However, the Java GSS-API provides both message integrity and message confidentiality. The *wrap* method is used to encapsulate a clear text message in a token such that its integrity and privacy are protected. The original clear text is returned by the peer's *unwrap* method when the token is passed to it. The properties object on the *unwrap* side returns information about whether the message was simply integrity protected or whether it was encrypted as well. It also contains sequencing and duplicates token warnings.

CHAPTER 4

SOFTWARE PERFORMANCE AND POWER ANALYSIS

Mobile phones have critical parameters which must be considered carefully when designing applications for this limited computing platform. These parameters mainly fall into three categories: power, processing capabilities, and memory. This chapter studies these parameters and focuses on code performance, space requirements and power consumption.

The Sun Java Wireless Toolkit for Connected Limited Device Configuration (CLDC) [13] provides several tools to monitor the behavior of the applications. These tools are helpful in debugging and optimizing the code:

- The ***profiler*** lists the frequency of use and execution time for every method in the application.
- The ***memory monitor*** shows the usage of memory while your application runs.
- The ***network monitor*** shows network data transmitted and received by the application.
- ***Tracing*** outputs low-level information to the toolkit console.

4.1 Performance

The performance of the application is a very critical parameter. Mobile phones have limited processing power. Kerberos Secure Phone Messenger (KSPM) is designed carefully to perform all the tasks with the best possible performance.

The profiler (Figure 4.1) keeps track of every method in the application. For a particular run, it Figures out how much time was spent in each method and how many

times each method was called. After running the application the profiler pops up allowing to browse through the method call information.

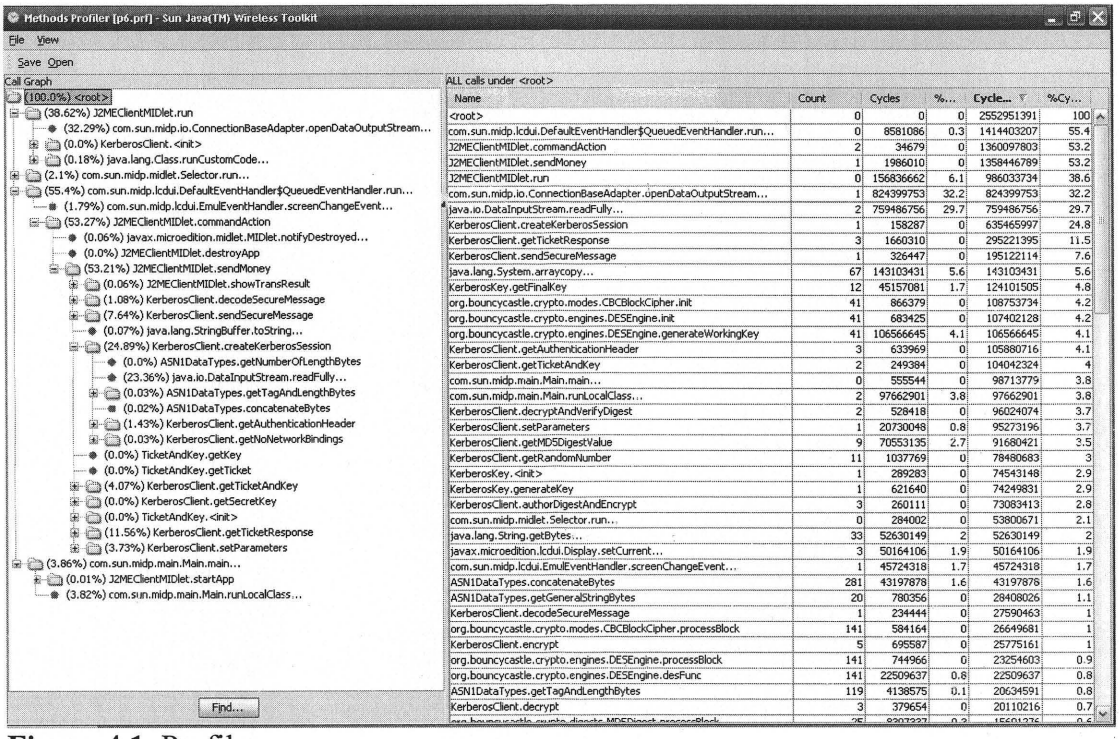


Figure 4.1 Profiler.

The profiler results show that the application executes a total of 9,630,347 VM instructions. For further analysis, the profiler also shows the cycles required for each method. Based on these statistics the analysis is done in the rest of this section.

4.1.1 Distribution of CPU Cycles

Figure 4.2 compares the percentage of CPU cycles used by the main parts in the program. Network's stream connection consumes the largest percentage 62% out of the whole program execution time. Network's socket streams are the most demanding resource in the application. These streams use TCP connection to exchange data. TCP is a complex

protocol; this complexity is a price for the reliability of the protocol. On the other hand, UDP protocol is totally efficient in phone's application because it does not use streams.

Kerberos protocol comes in the second place consuming 23% of the application execution time. The main processing overhead in the protocol comes from the encryption complexity for secure algorithms. Yet symmetric key cryptography complexity is acceptable, if compared to public key cryptography. Kerberos has a big advantage over other protocols because of symmetric key efficiency. The rest of the execution time (15%) is used for loading the application, initiating the classes and displaying the interface.

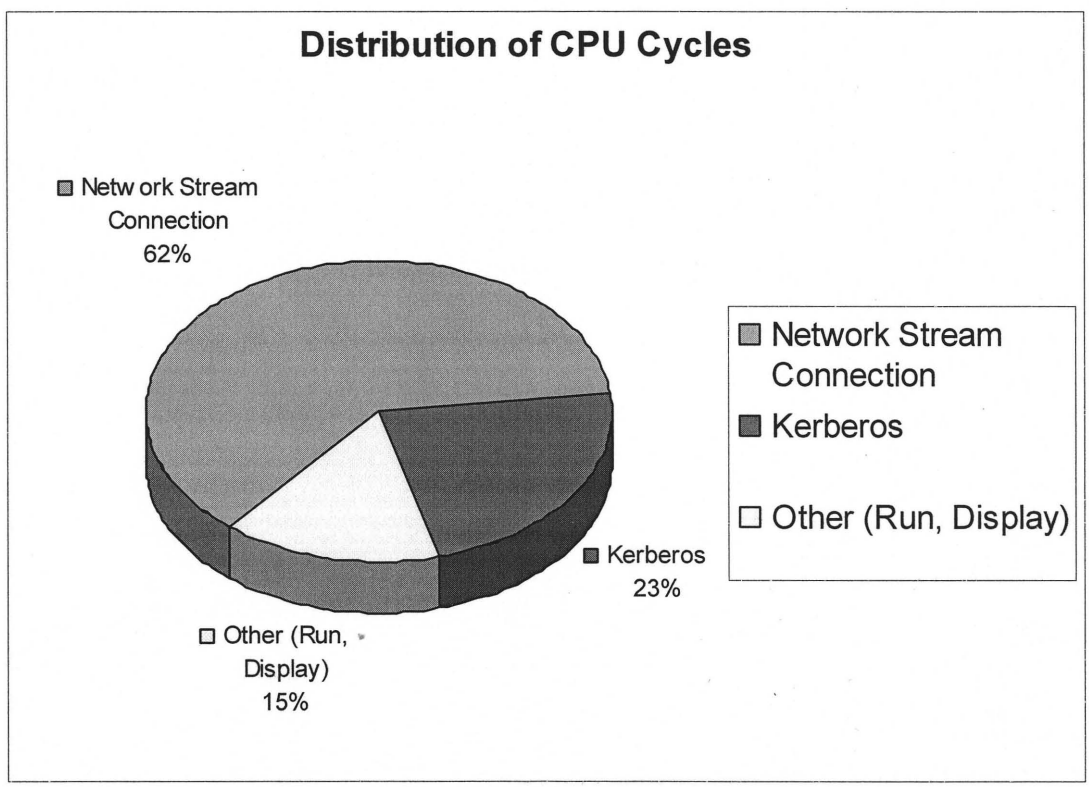


Figure 4.2 Distribution of CPU cycles used by the main parts in the program.

Figure 4.3 studies the percentage of CPU cycles for main routines in Kerberos protocol. These routines represent the high level of interface for Kerberos. Each routine has a specific function explained in the earlier chapter. These routines take up to 23% of the execution time as explained in the Figure above. The Figure shows that six routines require almost 11.5% of the application total execution time. However, the function *getTicketResponse* requires another 11.5% of the application total execution time. Obviously, this function has a large overhead when compared to other functions for two reasons. First, the function is being executed three times while other functions are executed one time each. Second, the function uses the system method java.lang.System.arraycopy which is called 67 times taking 5.6% of the execution time. This function copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array. Thus, this method is called many times because encryption streams use arrays frequently.

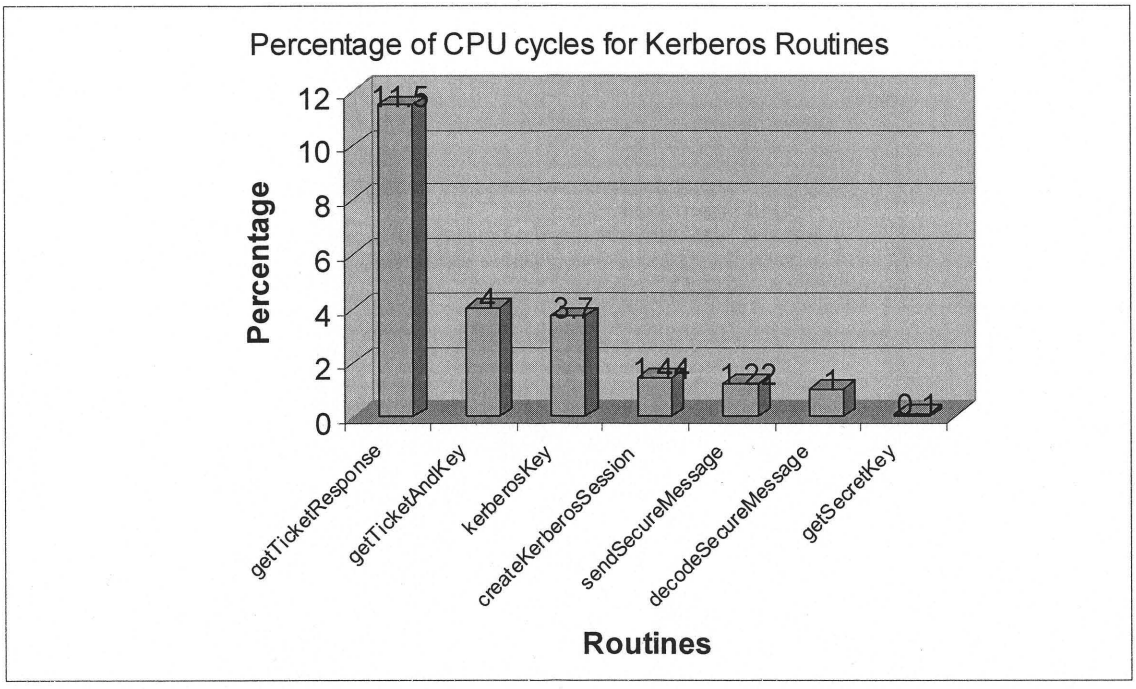


Figure 4.3 Percentage of CPU cycles for main routines in Kerberos protocol.

4.1.2 Execution Time

The application executes a total of 9,630,347 VM instructions. The number of instructions is dependent on the target processor. In this analysis, an assumption made is that the number of VM instructions is equal to the number of instructions actually executed on the target processor. Thus, the analysis is based on estimation and real values can differ slightly. The execution time can be calculated using Equation 4.1, provided that the phone's execution MIPS (Million Instructions per Second) rate is known, which is normally true.

CPU time (in seconds) = Number of Instructions (in millions)/ MIPS (4.1)

Figure 4.4 shows the estimated execution time for the application on three different ARM processors [14] ARM7, ARM9 and XScale. Clearly, phones with high clock rates are faster. The execution time is smallest when executed on ARM XScale (624 MHz) processor. Table 4.1 shows the processors performance details and the estimated execution time for the VM instructions on these processors.

Table 4.1 Estimated Execution Time on Three Different Processors.

Phone	Processor	CPU Clock Rate (MHz)	MIPS	Execution Time (ms)
Nokia	ARM7	60	60	160.50
Sony Ericsson, Siemens	ARM9	200	220	43.77
Dell	XScale	624	800	12.03

Source: Wikipedia, "ARM Architecture," http://en.wikipedia.org/wiki/ARM_architecture.

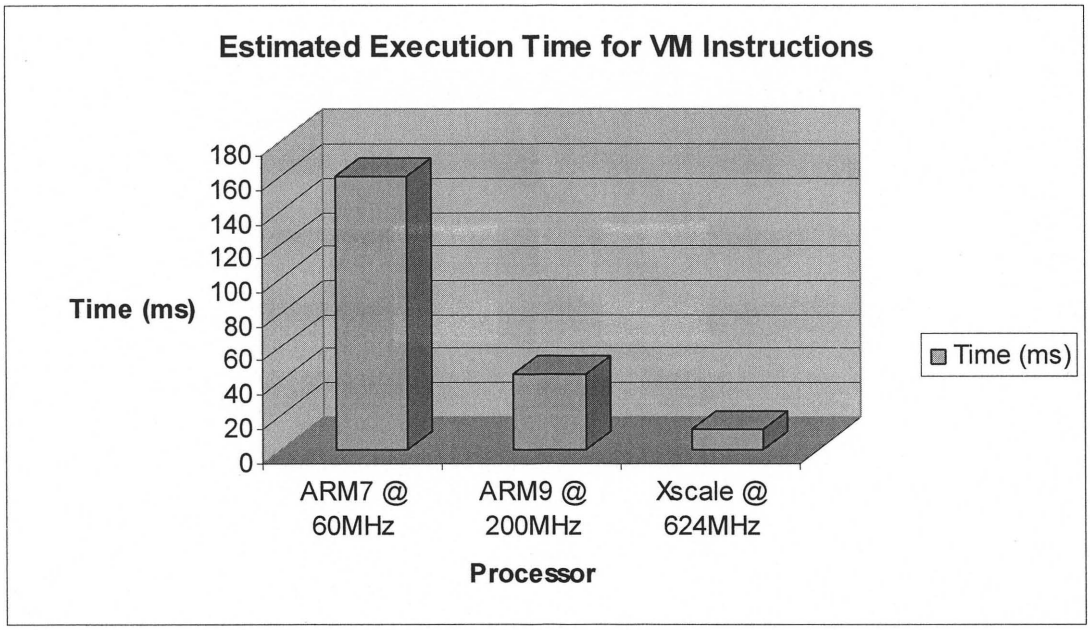


Figure 4.4 Estimated execution time (for VM instructions) on three different processors.

Figure 4.5 shows the estimated execution time for the three major parts in the application using 200MHz processor executing 220MIPS. The longest processing time (27ms) is being spent in network stream connection as explained in the section before.

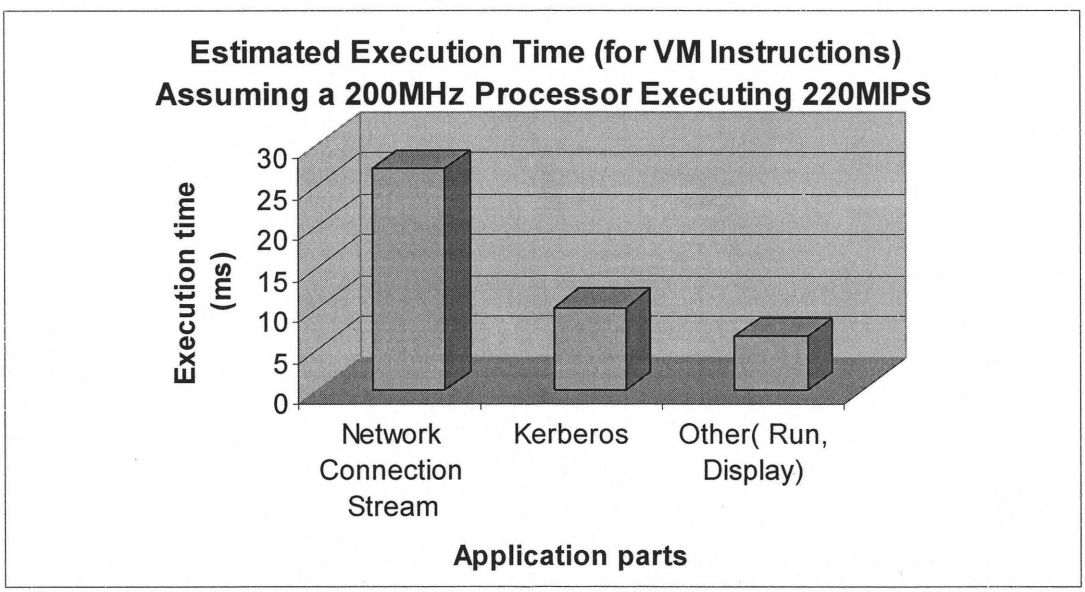
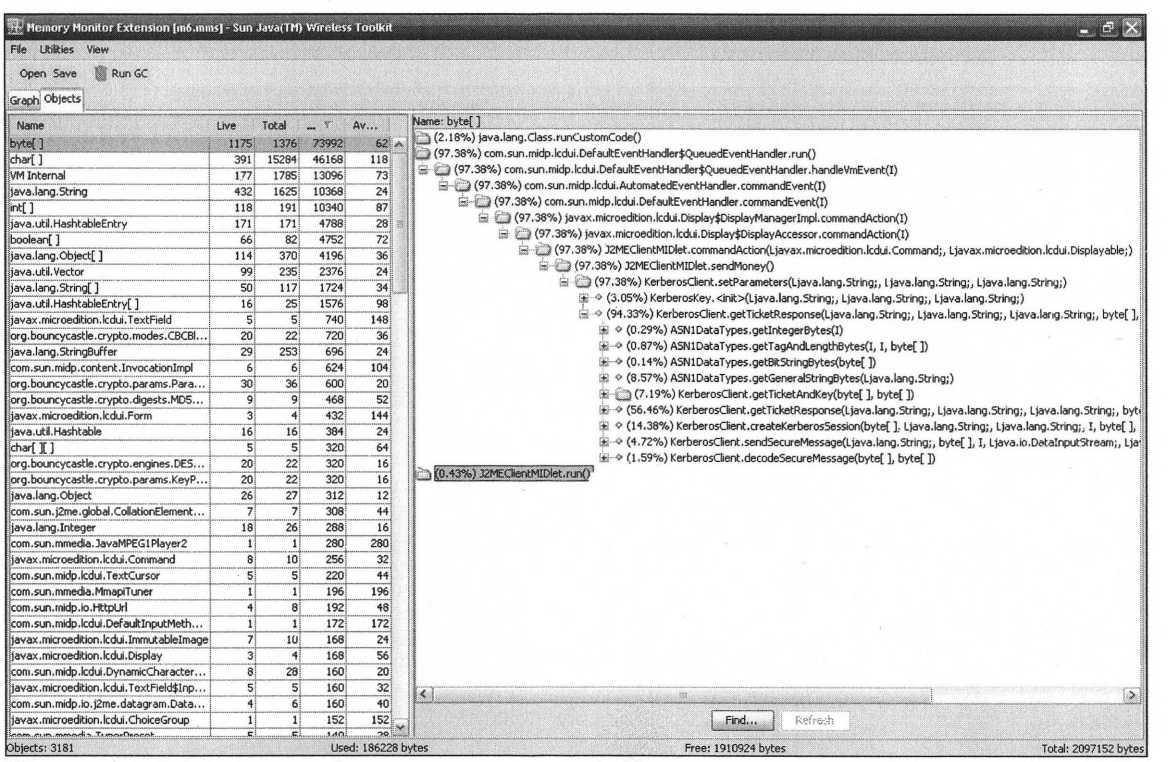


Figure 4.5 Estimated execution time for major application parts assuming a 200MHz processor executing 220MIPS. Direct implementation of VM instructions is assumed.

4.2 Memory Requirements

Memory is scarce on many mobile devices. Usually, the critical memory is the RAM. At the time of writing this document, the average RAM size equals 64MB and the flash memory size can range from 64MB to 1GB or more. The applications are loaded to the RAM at run time from the flash memory. The operating system (Windows Mobile 5.0) requires about 20 MB from the RAM. Thus, only two thirds of the RAM size will be free for user's applications.

The Sun Java Wireless Toolkit includes a memory monitor (Figure 4.6) that makes it easy to examine the memory usage of the application. It shows the total memory used by the application and a detailed listing of the memory usage per object.



Name	Live	Total	...	Av...
byte[ ]	1175	1376	73992	62
char[ ]	391	15284	46168	118
VM Internal	177	1785	13096	73
java.lang.String	432	1625	10368	24
int[ ]	118	191	10340	87
java.util.HashtableEntry	171	171	4788	28
boolean[ ]	66	82	4752	72
java.lang.Object[ ]	114	370	4196	36
java.util.Vector	99	235	2376	24
java.lang.String[ ]	50	117	1724	34
java.util.HashtableEntry[ ]	16	25	1576	98
javax.microedition.lcdui.TextField	5	5	740	148
org.bouncycastle.crypto.modes.CBCBl...	20	22	720	36
java.lang.StringBuffer	29	253	696	24
com.sun.midp.content.InvocationImpl	6	6	624	104
org.bouncycastle.crypto.params.Para...	30	36	600	20
org.bouncycastle.crypto.digests.MD5...	9	9	468	52
javax.microedition.lcdui.Form	3	4	432	144
java.util.Hashtable	16	16	384	24
char[ ][ ]	5	5	320	64
org.bouncycastle.crypto.engines.DES...	20	22	320	16
org.bouncycastle.crypto.params.KeyP...	20	22	320	16
java.lang.Object	26	27	312	12
com.sun.j2me.global.CollationElement...	7	7	308	44
java.lang.Integer	18	26	288	16
com.sun.mmedia.JavaMPEG1Player2	1	1	280	280
javax.microedition.lcdui.Command	8	10	256	32
com.sun.midp.lcdui.TextCursor	5	5	220	44
com.sun.mmedia.MmapiTuner	1	1	196	196
com.sun.midp.io.HttpUrl	4	8	192	48
com.sun.midp.lcdui.DefaultInputMeth...	1	1	172	172
javax.microedition.lcdui.ImmutableImage	7	10	168	24
javax.microedition.lcdui.Display	3	4	168	56
com.sun.midp.lcdui.DynamicCharacter...	8	28	160	20
javax.microedition.lcdui.TextField$Inp...	5	5	160	32
com.sun.midp.io.j2me.datagram.Data...	4	6	160	40
javax.microedition.lcdui.ChoiceGroup	1	1	152	152

Figure 4.6 Memory monitor.

The application size once it is launched is 67,928 Bytes. Then, the application size changes depending on the called functions. Each function requires allocating dynamic memory from the heap at run time. The size becomes 186,228 Bytes when all Kerberos routines are executed. After the authentication is done, the size drops again to 78,340 Bytes. This drop can be achieved only if garbage data is disposed correctly.

Another important fact when running the application on windows mobile operating system is that it uses a Java Virtual Machine (JVM) to convert the code to machine code. The major memory requirement is going to be the JVM size (Figure 4.7). The only available smartphone JVM in the market is WebSphere Everyplace Micro Edition Java (WEME) by IBM [9]. WEME or J9 at run time allocates 4 MB of memory or more depending on the available free memory. On the other hand, when running the application on a java operating system the VM will be already included.

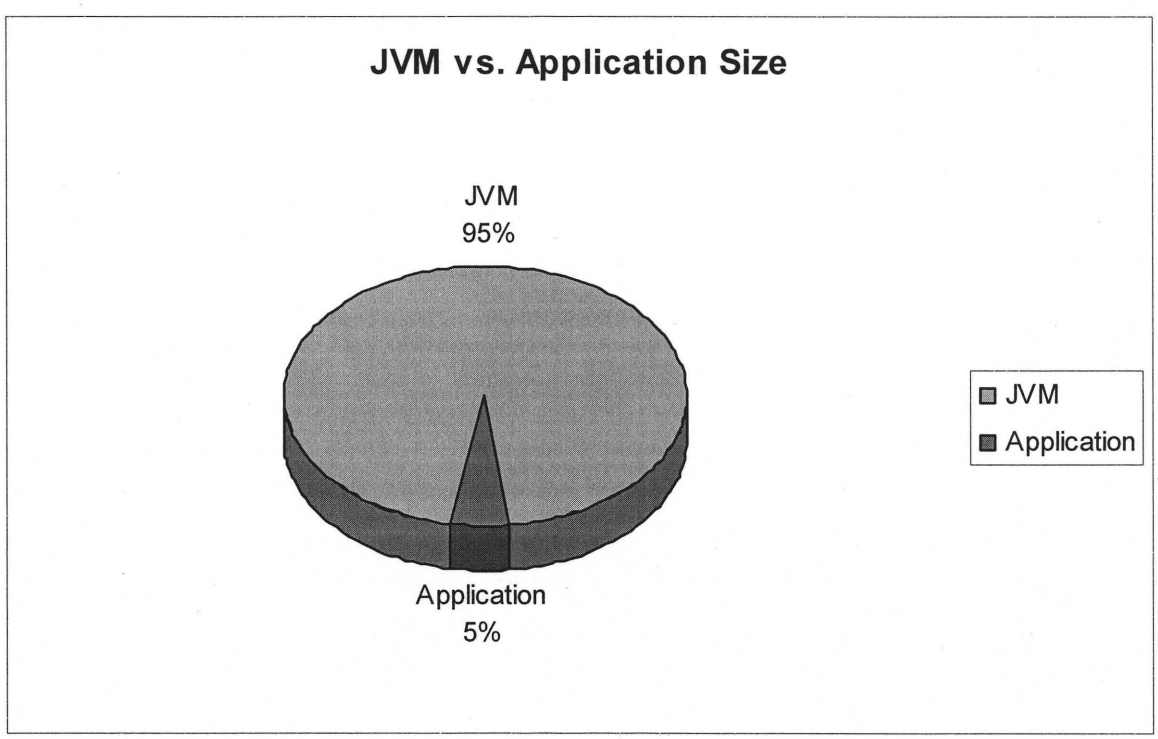


Figure 4.7 JVM vs. application size.

Analyzing the memory requirement for the major routines in Kerberos (Figure 4.8) is important to optimize these main routines. The function *getTicketResponse* requires about 45KB of memory. Obviously this function seizes a large space when compared to other functions. This same function requires the longest execution time as explained before. This is because the function is being executed three times while other functions are executed one time each. It also requires a largest size of byte arrays because encryption streams use arrays frequently.

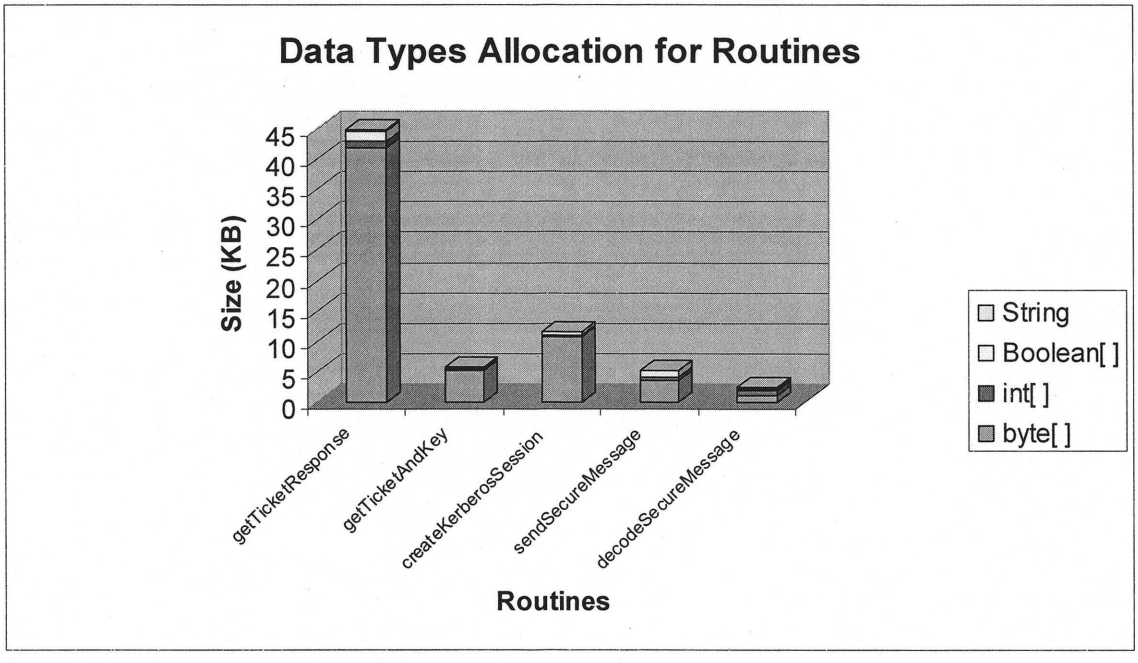


Figure 4.8 Data types allocation for Kerberos routines.

4.3 Power Consumption

ARM CPUs are dominant in the mobile electronics market, where low power consumption is a critical design goal. Analysis shows that the application will consume less than 100mW. The major power consumption in smart phones comes from the wireless connections.

Table 4.2 ARM Processor Power Consumption for the Application

Phone	Processor	CPU Clock Rate (MHz)	MIPS/W	Application Power (mW)
Nokia	ARM7	60	400	11.3
Sony Ericsson, Siemens	ARM9	200	600	16
Dell	XScale	800	850	24

Source: Wikipedia, "ARM Architecture," http://en.wikipedia.org/wiki/ARM_architecture.

Table 4.2 shows the estimated power consumption for the application on three different ARM processors. The MIPS/W ratio for ARM7, ARM9, and XScale processors is obtained respectively from [15, 16, 17]. This ratio gives a clear indication of the processor's performance and power consumption at the same time, independent from the processor's clock rate. Providing that the number of VM generated instructions equals 9630347, the power can be calculated using the MIPS/W ratio. As stated earlier, the number of instructions is dependent on the target processor. An assumption is made that the number of VM instructions is equal to the number of instructions being executed on the real processor. Thus, the analysis is based on estimations and real values can differ slightly. The power values are calculated using Equation 4.2, where the resulting unit is watts per second.

$$\text{Power} = \text{Number of Instructions} / (\text{MIPS/W}) \tag{4.2}$$

Figure 4.9 shows the estimated power consumption for the application on three different ARM processors. Power values are calculated in Table 4.2. Results clearly indicate that the higher the clock rate, the higher the power consumption.

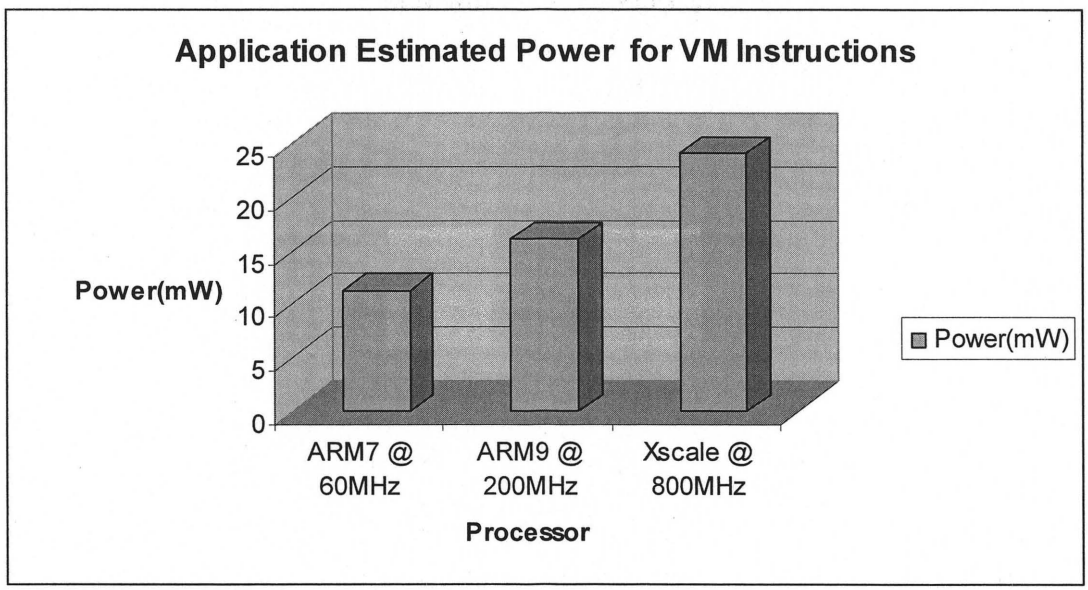


Figure 4.9 Estimated power consumption for the application using VM instructions.

Numerous factors affect the battery lifetime of a smart phone. Despite the fact that each application that runs on the phone contributes differently to its battery drain, there are five fundamental hardware components that consume most of the power: backlight, Bluetooth, CPU, WiFi, and the cell radio. Figure 4.10 illustrates the hardware component distribution of power consumption for an average user [18]. As observed, the processor consumes less power than the WiFi and the GSM radio. However, power analysis in depends on estimations because phones run in different modes. The phone idle time consumes very little power, while making calls is power demanding. Figure 4.10 assumes a rough estimate for the percentages of time when a typical user makes phone calls, browses the Internet, uses Bluetooth headset, plays games, or plays media files.

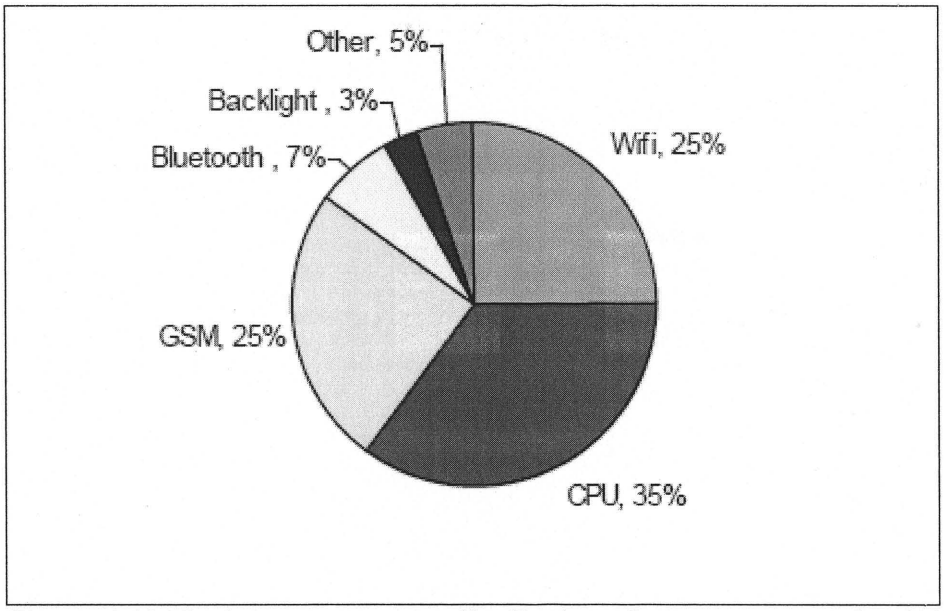


Figure 4.10 Hardware component distribution of power consumption [18].

4.4 Network Performance

The network monitor tool (Figure 4.11) provides a simple way to observe network data exchange. This is helpful in debugging network interactions or looking for ways to optimize network traffic. When the application makes any type of network connection, information about the connection is captured and displayed. The Figure 4.11 shows datagrams requests and responses. The display on the left side shows a hierarchy of messages and message pieces. The details are in the right side of the network monitor. Message bodies are shown as raw hexadecimal values and the equivalent text.

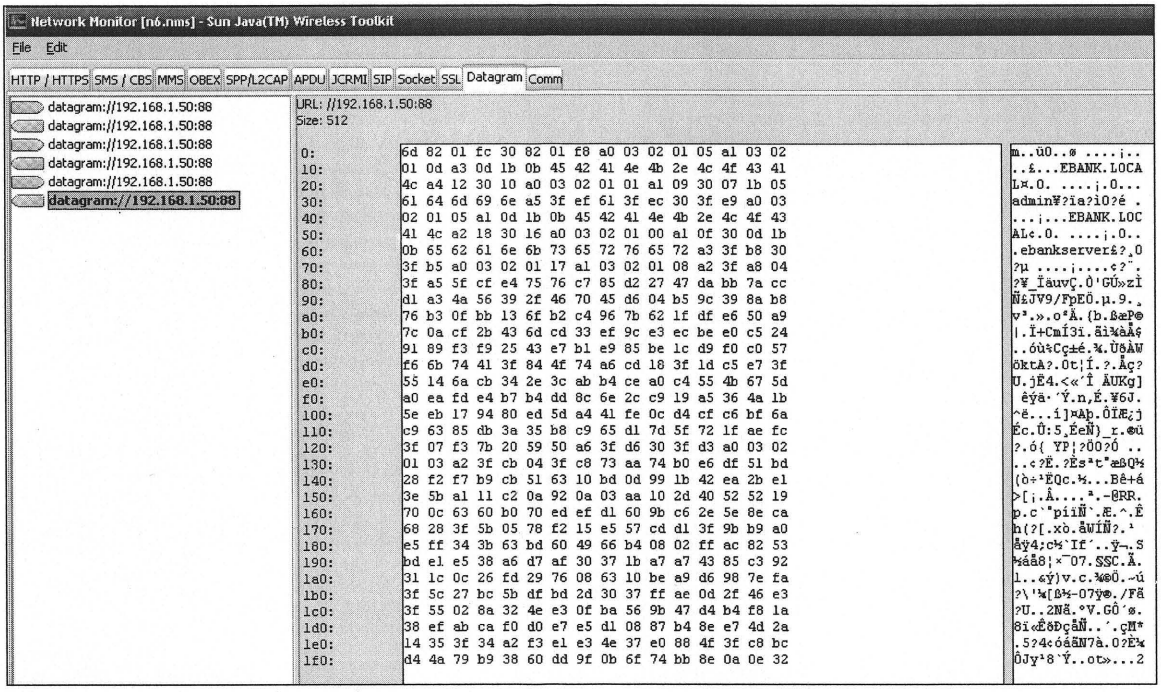


Figure 4.11 Network monitor.

The execution time has network delays to consider. The exchange of messages over the network has a certain delay depending on the network itself. Kerberos authentication is highly affected by this. The authentication process has to exchange with the server about six messages before the secure channel is established. This delay is also dependant on the server speed, as the server needs to decrypt the messages and authenticate the user. The delay by the server can be ignored for now, assuming the server is fast and its processing time is negligible if compared with the phone.

Figure 4.12 shows the sequence of Kerberos authentication messages labeled by their size as noticed by the network monitor. All packets have a small size. The delay for exchanging these packets is negligible. The Round Trip Time (RTT) is less than microseconds in a LAN and less than 11ms on average for 1KB packets in the internet [19]. Hence, the delay of network can be ignored in small networks as in NJIT.

If the size of Kerberos messages is larger than 1500 bytes, fragmentation will occur. Usually, the size is less than 1500 bytes unless long authorization lists are included in the message. If this happens, windows server will replace the UDP connection with a TCP connection to exchange the authentication messages [20]. This will require more resources from the phone. As a result, the messages size should not exceed the limit, otherwise noticeable delays will arise.

Time synchronization restrictions in Kerberos add an extra step in authentication message exchange. This step arises very frequently when authenticating with windows adding more delays [20]. Usually Kerberos allows five minutes skew between the server and the client. This problem can be solved by using network time protocol to synchronize the clock for all computers on the network.

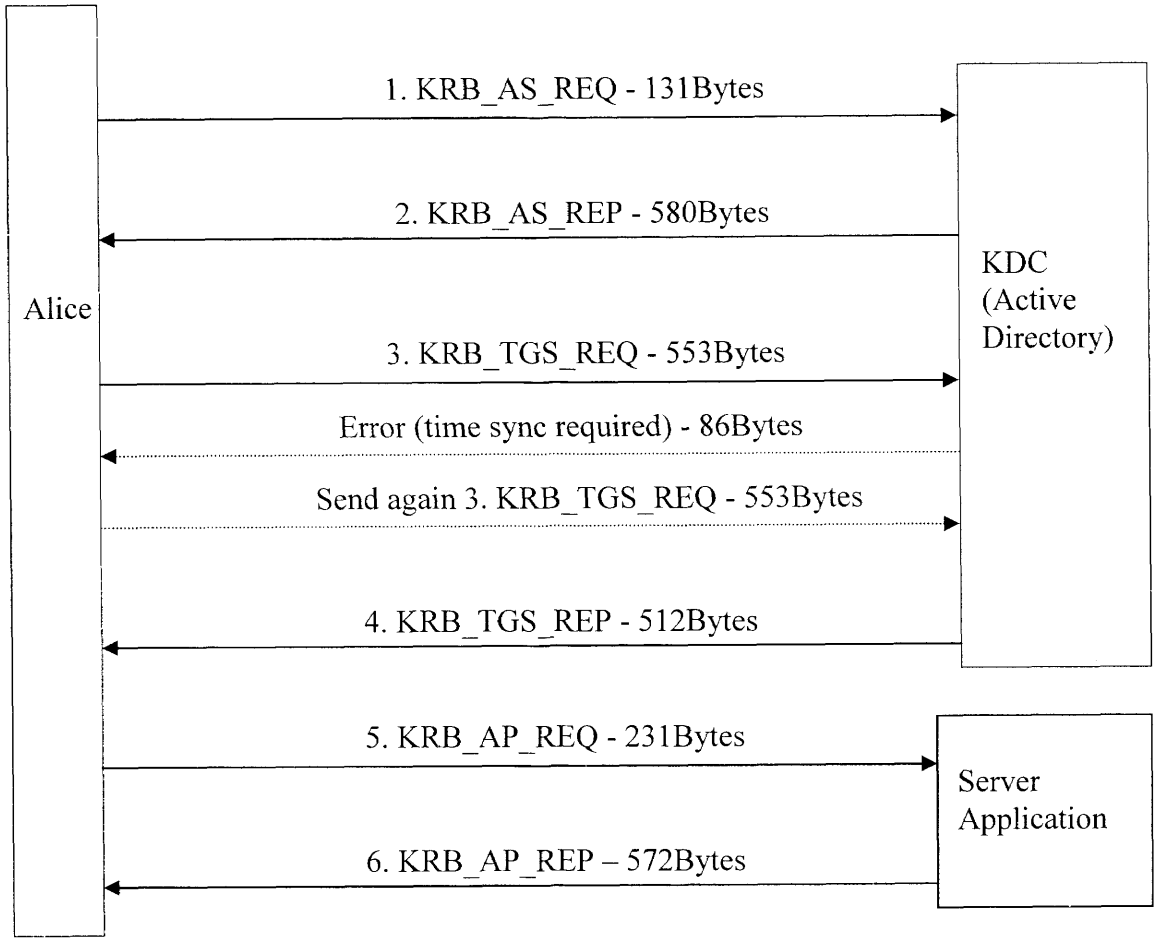


Figure 4.12 Kerberos messages exchange showing messages size.

CHAPTER 5

CONCLUSIONS AND FUTURE RESEARCH

5.1 Conclusions

Internet enabled mobile devices are widely spreading while security is still not maturely addressed for such platforms. Therefore, this research implements a novel Kerberos Secure Phone Messenger (KSPM) extending the popular Kerberos authentication protocol to run on mobile phones. Moreover, the Kerberos network authentication protocol provides user authentication and message privacy with the convenience of secret key cryptography to reduce the computational burden and power consumption if compared with public key cryptography. KSPM analysis shows that the application performance and power consumption is very practical on the commonly used mobiles empowered by ARM processors. Hence, KSPM can provide outstanding secure services for corporate users by protecting their workplace network and all other mobile communications.

5.2 Future Research

Further analysis can be done for the performance of the application using java dedicated hardware processors. Predictions show that major improvements can occur if using such processors to run the application.

5.2.1 Java Hardware Processors

A program written in a language that is designed to run on a virtual machine (VM) is usually compiled to a pseudo-assembler bytecode language. This bytecode is then downloaded and executed on the target device within a virtual execution environment, such as a Java VM. The interpretation of bytecode in software maps individual Java bytecodes to machine instruction sequences. This is efficient in terms of memory footprint requirements, but the interpretation process severely limits performance.

However, direct execution of Java bytecode in hardware, significantly boosts performance, as the bytecode effectively becomes the native instruction set. A key benefit of interpretation techniques is that execution of the application is immediate in other words there is no start-up delay [21]. Such examples of java hardware processors are PicoJava II and ARM Jazelle DBX.

PicoJava II

PicoJava II [22] is a 32-bit stack based Java processor purposed by Sun Microsystems. Microprocessor specifications are dedicated to native execution of Java-based bytecode without the need for an interpreter or JIT compiler, thus speeding bytecode execution many times, compared to standard CPU with a JVM [23]. This approach results in the fastest Java runtime performance with a small memory footprint and competitive performance.

ARM Jazelle DBX

ARM Jazelle DBX (Direct Bytecode eXecution) technology [24] enables hardware direct bytecode execution of Java. Jazelle DBX technology typically increases the performance of a highly optimized commercial JVM by around 2 and 4 times when running benchmarks or complex MIDP 2.0 applications. In addition, all Java bytecodes are restartable, so there is no overhead on real-time performance. When looking at Java on embedded devices, raw speed performance is not the only factor to consider. Power consumption, memory usage (RAM and ROM), ease of integration, system cost and user experience are all equally important and achieving the right balance between each of these constraints is essential.

Application Performance on Java Hardware Processors

For future research, studying the performance of the application on java hardware processors is very important. Such processors can provide major improvements in the application performance and memory requirements. The most important reason of performance improvement is that each bytecode instruction requires one cycle to execute on average. However, mapping the bytecode instructions to the processor's ISA will result in instructions requiring multiple cycles to execute. Furthermore, the size of bytecode instructions is less than the size of the processor's instructions. Hence, the memory requirements are less.

APPENDIX A

AUTHENTICATION SERVICE EXCHANGE MESSAGES

1) KRB_AS_REQ Message Contents (Authentication Service Request)

The initial message sent to the AS requesting a TGT is the KRB_AS_REQ message.

Table A.1 Authentication Service Request Message Fields [6]

Field	Description
Protocol Version	5
Message Type	KRB_AS_REQ
Pre-authentication Data	The intent of these fields is to provide pre-authentication before the KDC sends the client a ticket. This is to prevent brute force or dictionary attacks on the user's password. Note that these fields are used in the KRB_TGS_REQ message for the TGT and the authenticator.
PAData Type	PA-AS-REQ or PA-PK-AS-REQ.
PAData Value	The AS_REQ includes a client timestamp encrypted with the user key. In this case, the data type will be PA-AS-REQ.
Request Body	
KDC Options	The client can request that the TGT have certain optional features. The KDC options are detailed in the table below.
Client Name	Name of the requester.
Realm	Realm (Active Directory domain) of the requester.
Server Name	In the KRB_AS_REQ, this will be the KDC name. The client is specifically requesting a ticket for the TGS.
From	(Optional) If the ticket were postdated (not supported by Windows 2000, but is supported in Windows Server 2003), this field would specify the time from which the ticket would be valid.
Till	This is the requested expiration time. The KDC does not have to honor this request if the requested expiration time violates the domain's Kerberos policy.
Renew Time	(Optional) The requested renewal time.

Field	Description
Nonce	A random number generated by the client. The nonce can be used as the basis for sequence numbering or as an additional authenticator. The nonce supplied here will be returned in the encrypted portion of the KRB_AS_REP message. The client will compare the nonces to make sure they match.
Encryption Type	The desired encryption algorithm to be used.
Addresses	IP addresses from which ticket will be valid.
Encrypt Authorization Data	Not used for KRB_AS_REQ.
Additional Tickets	Not used for KRB_AS_REQ.

2) KRB_AS_REP Message Contents (Authentication Service Response)

This same structure is used for both the KRB_AS_REP message and the KRB_TGS_REP message. Thus, the field contents might change depending on the message type.

Table A.2 Authentication Server Reply Message Fields [6]

Field	Description
Protocol Version	5
Message Type	KRB_AS_REP
Pre-authentication data	For the KRB_AS_REP message, this field will normally be empty.
Client Realm	Realm or domain of the requester.
Client Name	Name of the requester.
Ticket	The encrypted ticket is placed here. For the KRB_AS_REP message, this will be the TGT encrypted with the TGS's secret key.
	The rest of the message fields are encrypted with the user's key.

Field	Description
	The client cannot read the encrypted ticket and might need to verify the ticket information. Furthermore, the client needs to have information about start times and end times so it can determine when to request renewal or replacement tickets. Therefore, the seven fields from Authentication Time to Client Addresses are all included in the ticket and copied here into these fields.
Key	This is the session key the user will use to encrypt and decrypt TGS messages.
Last Requested	The last time a ticket was requested. This is similar to a "last logged on" time. This can be used to track how frequently a client is requesting tickets. A Kerberos policy can be set that limits how frequently ticket requests can be made. Such limits can help prevent brute force attacks.
Nonce	The nonce from the KRB_AS_REQ nonce field will be copied here.
Key Expiration	When the user's key will expire. This is used for password aging.
Flags	These are the flags set in the ticket, based on the flags requested in the KRB_AS_REQ message and the domain's Kerberos policy.
Authentication Time	The time the ticket was issued.
Start Time	(Optional) At what time the ticket is valid.
End Time	At what time the ticket expires (although it can be renewed if the ticket is renewable).
Renew Till	(Optional) At what time the ticket absolutely expires (cannot be renewed past this time).
Server Realm	Requested server's realm (domain).
Server Name	Requested server's name
Client Addresses	(Optional) Addresses from which the ticket will be valid. This is important in several situations. Normally, a ticket is only valid if sent from a specific address.

3) KRB_TGS_REQ Message Contents (The Ticket-Granting Service Request)

The KRB_TGS_REQ message is similar to the KRB_AS_REQ message. The only difference is the pre-authentication field (PAData Type and PAData Value). The TGT and the authenticator are placed in the PAData Value field. The authenticator is encrypted

with the session key. The TGT is encrypted with the TGS's secret key (the key based on the krbtgt user account). A checksum is computed using the contents of fields in the request body, and that checksum is encrypted within the authenticator. The following table lists and describes the fields used in the ticket-granting service request message.

Table A.3 Ticket-Granting Service Request Message Fields [6]

Field	**Description**
Protocol Version	5
Message Type	KRB_TGS_REQ
Pre-authentication Data	The intent of these fields is to provide some authentication before the KDC sends the client a ticket. This is to prevent brute force or dictionary attacks on the user password. Note that this field is used in the KRB_TGS_REQ message for the TGT and the authenticator.
PAData Type	PA-TGS-REQ
PAData Value	TGT and authenticator.
Request Body	
KDC Options	The client can request that the ticket have certain optional features.
Client Name	(Optional) Name of the requester.
Realm	Realm or domain of the requester.
Server Name	(Optional) Requested server name.
From	(Optional) If the ticket were postdated (not supported by Windows 2000, but supported in Windows Server 2003), this field would specify the time from which the ticket would be valid.
Till	This is the requested expiration time. The KDC does not have to honor this request, if the requested expiration time violates the domain's Kerberos policy.
Renew Time	(Optional) The requested renewal time.
Nonce	A random number generated by the client. The nonce can be used as the basis for sequence numbering or as an additional authenticator. The nonce supplied here will be returned in the encrypted portion of the KRB_TGS_REP message. The client will compare the nonces to make sure they match.

Field	Description
Encryption Type	The desired encryption algorithm to be used.
Addresses	(Optional) IP addresses from which ticket will be valid.
Encrypt Authorization Data	(Optional) In the KRB_TGS_REQ message, this will specify a key to use to encrypt any pre-authentication data.
Additional Tickets	(Optional) In a KRB_TGS_REQ for user-to-user authentication, a TGT will be included in this field. The TGS will use the session key from this TGT — instead of the service's secret key — to encrypt the service ticket.

4) KRB_TGS_REP Message Contents (Ticket-Granting Service Response)

The KRB_TGS_REP and KRB_AS_REP messages are similar and the fields are similarly used. The major difference is that the ticket in the Ticket field is not a TGT. Instead, it is the service ticket for the target server.

Table A.4 Ticket-Granting Server Reply Message Fields [6]

Field	Description
Protocol Version	5
Message Type	KRB_TGS_REP
Pre-authentication Data	In a KRB_TGS_REP message, application-specific data might be placed here.
Client Realm	Realm or domain of the requester.
Client Name	Name of the requester
Ticket	The encrypted ticket is placed here. For the KRB_TGS_REP message, this will be the service ticket encrypted with the target server's secret key.
	The message fields below are encrypted with the TGS session key.
Key	This is the session key the user will use with the application server.

Field	Description
Last Requested	The last time a ticket was requested. This is similar to a "last logged on" time. This can be used to track how frequently a client is requesting tickets. A Kerberos policy can be set that limits how frequently ticket requests can be made. Such limits can help prevent brute force attacks.
Nonce	The nonce from the KRB_TGS_REQ nonce field will be copied here.
Key Expiration	(Optional) When the user's key will expire. This is used for password aging.
Flags	These are the flags set in the ticket, based on the flags requested in the KRB_TGS_REQ message and on the domain's Kerberos policy.
Authentication Time	The time the ticket was issued.
Start Time	(Optional) At what time the ticket is valid.
End Time	At what time the ticket expires (although it can be renewed if the ticket is renewable).
Renew Till	(Optional) At what time the ticket absolutely expires (cannot be renewed past this time).
Server Realm	Requested server's realm (domain).
Server Name	Requested server's name.
Client Addresses	(Optional) Addresses from which the ticket will be valid.

5) KRB_AP_REQ Message Contents (Application Server Request)

The KRB_AP_REQ message contains an authenticator encrypted with the session key that the client and target server share, the service ticket encrypted with the target server's secret key, and the optional mutual authentication request.

Table A.5 Application Server Request Message Fields [6]

Field	Description
Protocol Version	5

Field	Description
Message Type	KRB_AP_REQ
Application Options Fields	
Use Session Key	During user-to-user authentication, after the client has obtained a new ticket for the target service, the client will send a new KRB_AP_REQ with this flag set. This tells the target service to use its session key to decrypt the ticket.
Mutual Authentication Required	If this flag is set, the target server will respond with the KRB_AP_REP message, which will authenticate the target server to the client.
Other Message Fields	
Ticket	The service ticket for the target server, encrypted in either the target server's secret key or in its session key, depending on whether user-to-user authentication is required.
Authenticator	The client's timestamp and other data, encrypted with the session key that the client and target server share.

6) KRB_AP_REP Message Contents (Application Server Response)

This message is used when mutual authentication is required (if the client needs to verify the target server's identity). The message consists of a copy of the client's timestamp from the authenticator that was previously included in the KRB_AP_REQ message encrypted with the session key that the client and the target server share.

Table A.6 Application Server Reply Message Fields [6]

Field	Description
Protocol Version	5
Message Type	KRB_AP_REP
	The following message fields are encrypted with the session key

Field	Description
Client Time	Current time on Client (from the authenticator).
CUSEC	Millisecond part of client time (from the authenticator).
Subkey	(Optional) Specifies a key to encrypt client sessions with application server.
Sequence Number	(Optional) Application-specific, so used if sequence number was specified in the authenticator.

APPENDIX B

SOFTWARE OPERATIONS

This appendix describes the operations of the program. It describes briefly how KSPM works and what is required to establish a connection. Snapshots from the program are displayed.

Running KSPM on the phone requires WebSphere Everyplace Micro Edition Java (WEME) to be installed on the phone. On the other hand, the messenger server requires a java runtime compiler to be installed on the server.

KSPM connection parameters such as user name, user password, realm, server name, server IP address and service ports must be known in order to establish the connection. Parameters like user name, user password, and realm are inserted at run time by the GUI interface. Other parameters like the server IP address and port number are hard-coded. Before the connection takes place, the user name and password should be already stored in Active Directory at the server. The messenger server should be running before launching KSPM to complete the connection establishment.

In Figure B.1, two clients (Bob and Alice) are using KSPM to exchanging messages. The program is running using an emulator. Connection information is displayed in the chatting screen for testing purposes. Figure B.2 shows the Messenger Server functioning as an intermediate point between clients. It also logs chat sessions if required.

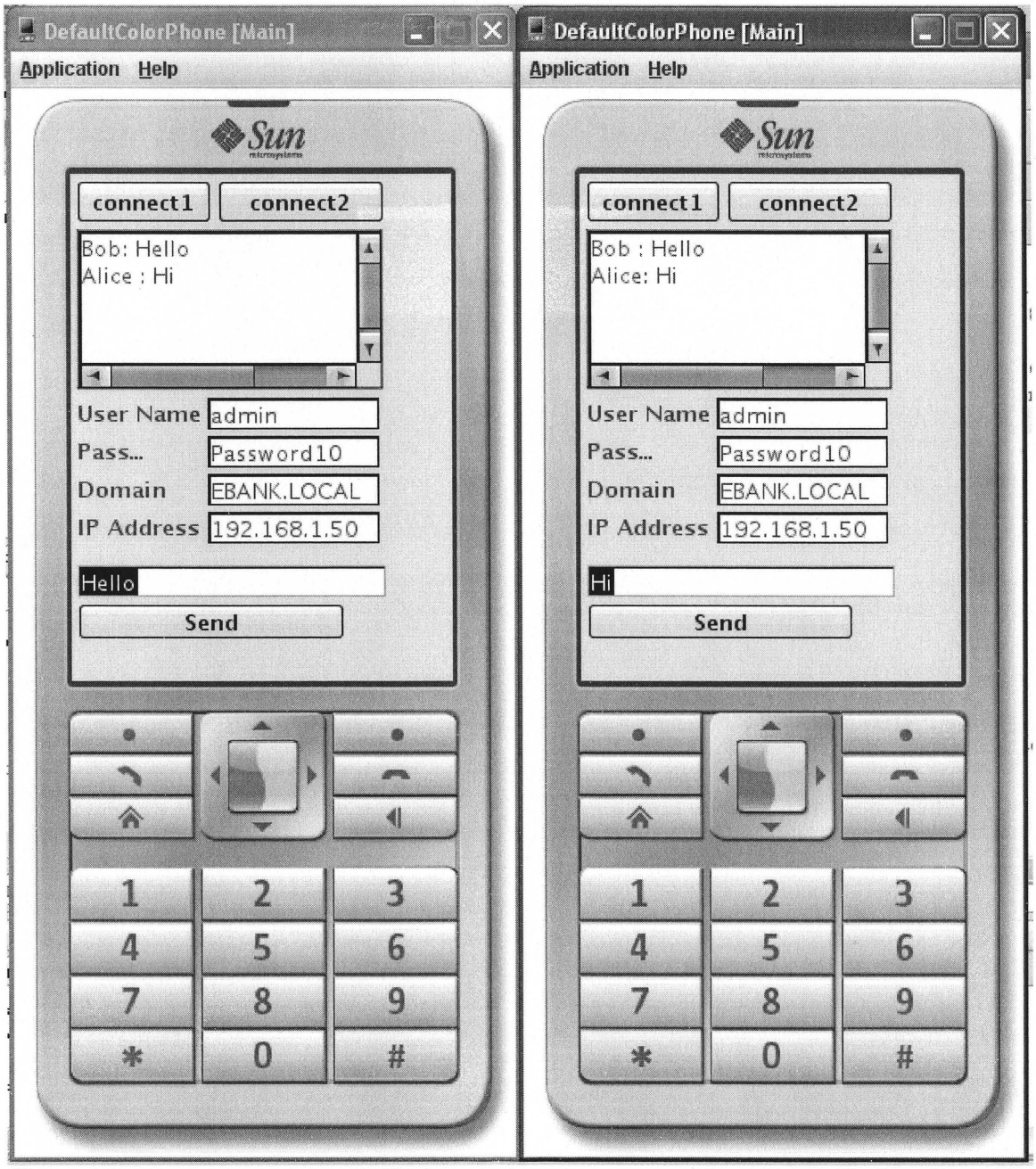


Figure B.1 Two clients using KSPM to exchanging messages.

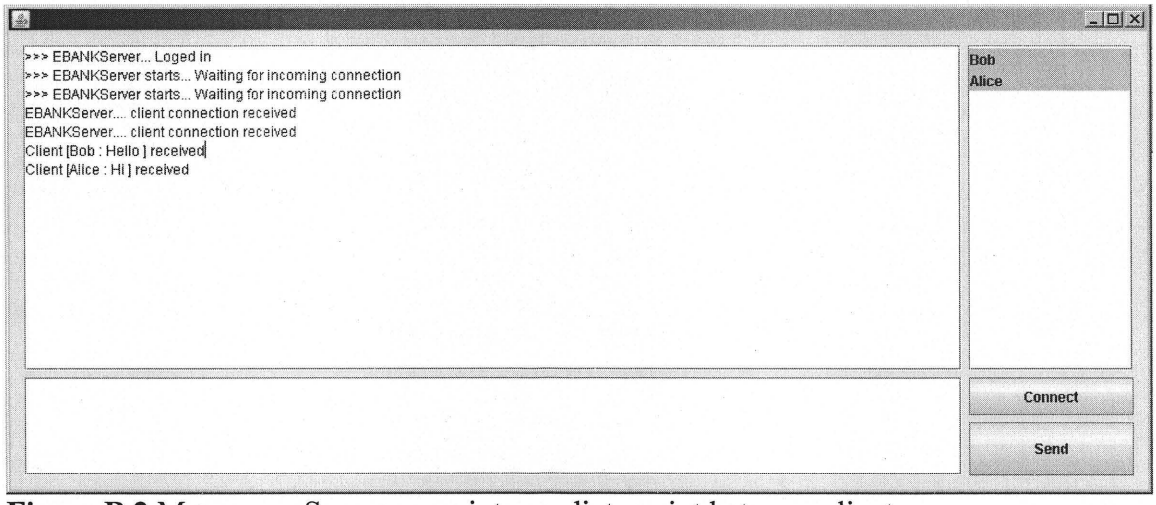


Figure B.2 Messenger Server as an intermediate point between clients.

REFERENCES

1. C. Neuman, T. Yu, S. Hartman, K. Raeburn, "The Kerberos Network Authentication Service (V5)", IETF, RFC 4120, July 2005.

2. MIT, Kerberos. Retrieved October 2, 2007 from the World Wide Web: http://web.mit.edu/Kerberos/.

3. B. Clifford Neuman, and Theodore Y. T'so, "Kerberos: An Authentication Service for Computer Networks", *IEEE Communications Magazine*, vol. 32, no. 9, pp. 33-38, Sept. 1994.

4. J. Garman, "Kerberos: The Definitive Guide", 2nd ed. California: O'Reilly, 2003. [E-book] Available: Safari e-book.

5. C. Kaufman, R. Perlman, M. Speciner, Network Security: Private Communication in a Public World, Second Edition, New Jersey: Prentice Hall, 2002, pp. 307-371.

6. Microsoft, How the Kerberos Version 5 Authentication Protocol Works. Retrieved October 2, 2007 from the World Wide Web: http://technet2.microsoft.com/WindowsServer/en/library/4a1daa3e-b45c-44ea-a0b6-fe8910f92f281033.mspx?mfr=true.

7. Jan De Clerc, "Windows Server 2003 security infrastructures: Core Security Features (HP Technologies)," 1st ed. Digital Press, 2004. [E-book] Available: Safari e-book.

8. N. Hindocha, E. Chien, "Malicious Threats and Vulnerabilities in Instant Messaging Symantec Security Response," *Symantec Security Response, Virus Bulletin International Conference*, September 2003.

9. IBM, WebSphere Everyplace Micro Environment. Retrieved November 1, 2007 from the World Wide Web: http://www-306.ibm.com/software/wireless/weme/

10. Faheem Khan, Lock down J2ME applications with Kerberos Part 1. Retrieved November 1, 2007 from the World Wide Web: http://www.ibm.com/developerworks/wireless/library/wi-kerberos/

11. Bouncy Castle Cryptography. Retrieved November 1, 2007 from the World Wide Web: http://www.bouncycastle.org/

12. M. Upadhyay, R. Marti , "Single Sign-on Using Kerberos in Java," Sun Microsystems. Retrieved November 1, 2007 from the World Wide Web: http://java.sun.com/j2se/1.4.2/docs/guide/security/jgss/tutorials/BasicClientServer.html

13. Sun Java Wireless Toolkit for CLDC. Retrieved November 1, 2007 from the World Wide Web: http://java.sun.com/products/sjwtoolkit/

14. ARM Processors. Retrieved November 1, 2007 from the World Wide Web: http://www.arm.com/products/CPUs/families.html

15. S. S. Segars, K. Clarke, and L. Goudge, "Embedded control problems, thumb and the ARM7TDMI," *IEEE MICRO*, pp. 22–30, Oct. 1995.

16. A. Efthymiou, J.D. Garside, and S. Temple, "A Comparative Power Analysis of an Asynchronous Processor," *In Proceedings of the 11th International Workshop - Power and Timing Modeling, Optimization and Simulation (PATMOS'01)*, pp. 1–10, September 2001.

17. L.T. Clark, E.J. Hoffman, and J. Miller et al., "An Embedded 32-b Microprocessor Core for Low-Power and High-Performance Applications," *IEEE Journal of Solid State Circuits*, pages 1599–1608, November 2001.

18. A. Anand, C. Manikopoulos, and et al., "A Quantitative Analysis of Power Consumption for Location-Aware Applications on Smart Phones," *IEEE International Symposium on Industrial Electronics (ISIE)*, pages 1986-1991, June 2007.

19. Internet Graphs, Packets Average RTT. Retrieved November 1, 2007 from the World Wide Web: http://dxmon3.cern.ch/cgi-bin/cricket/grapher.cgi?target=%2Flatency-loss%2Finternet-colt;view=Latency;ranges=d%3Aw%3Am%3Ay

20. Microsoft, Troubleshooting Kerberos Errors. Retrieved October 2, 2007 from the World Wide Web: http://www.microsoft.com/technet/prodtechnol/windowsserver2003/technologies/security/tkerberr.mspx

21. ARM, "Jazelle for Execution Environments" White paper. Retrieved November 1, 2007 from the World Wide Web: http://www.arm.com/pdfs/JazelleRCTWhitePaper_final1-0_.pdf

22. Sun, PicoJava. Retrieved November 1, 2007 from the World Wide Web: http://www.sun.com/software/communitysource/processors/picojava.xml

23. Wikipedia, PicoJava. Retrieved November 1, 2007 from the World Wide Web: http://en.wikipedia.org/wiki/PicoJava

24. ARM, High Performance Java on Embedded Devices. Jazelle DBX White paper. Retrieved November 1, 2007 from the World Wide Web: http://www.arm.com/pdfs/JazelleDBX_WhitePaper_2007v1p1.pdf