

## **Copyright Warning & Restrictions**

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

**Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation**

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

## **ABSTRACT**

### **TWO FACTOR AUTHENTICATION AND AUTHORIZATION IN UBIQUITOUS MOBILE COMPUTING**

**by**  
**Qing Zhu**

Handheld devices, such as mobile phones, PDAs and others, have become an integrated part of our lives. They are perfectly suitable to become the first real-life platforms for mobile computing applications. A lightweight holistic authentication and authorization scheme is presented for this typical wireless scenario. It is shown that either SIM or Kerberos authentication has its limitations. A two-factor authentication method SIM-based Kerberos authentication is developed to implement the Secure Single Sign-on function with LDAP protocol realization for authorization and mobile directory services. Performance analysis and power consumption analyses for this protocol are presented as well. Our results show that our SIM-Based Kerberos authentication is significantly optimized to be suitable for mobile platforms.

**TWO FACTOR AUTHENTICATION AND AUTHORIZATION  
IN UBIQUITOUS MOBILE COMPUTING**

by  
**Qing Zhu**

**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**

Blank Page

**APPROVAL PAGE**

**TWO FACTOR AUTHENTICATION AND AUTHORIZATION  
IN UBIQUITOUS MOBILE COMPUTING**

**Qing Zhu**

---

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

---

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

---

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

## **BIOGRAPHICAL SKETCH**

**Author:** Qing Zhu

**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 Science,  
University of Science and Technology, Beijing, P. R. China, 2002

**Major:** Computer Engineering

To my beloved parents



## ACKNOWLEDGMENT

I would like to express my sincerely appreciation to Dr. Sotirios Ziavras, who served as my research supervisor, providing valuable and countless guidance, insight, and intuition to me. Special thanks are given to Dr. Roberto Rojas-Cessa, Dr. Jie Hu, for actively participating in my committee.

The paper is also in memory of Dr. Constantine Manikopoulos who unfortunately passed away. I appreciate his guidance on choosing this topic and support for solving problems.

Many of my fellow graduate students in the CONEX Research Laboratory and Smartcampus Project are deserving of recognition for their support. I also wish to thank Ke Su for her assistance over the years.

## TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION.....	1
1.1 Motivation.....	1
1.2 Thesis Statement .....	2
2 BACKGROUND .....	3
2.1 SmartCampus Project.....	3
2.2 Single Sign-on Authentication .....	4
2.3 Related Work.....	6
3 KERBEROS.....	9
3.1. Introduction .....	9
3.2 Physical Architecture .....	10
3.2.1 KDC.....	10
3.2.2 Client User .....	15
3.2.3 Server with desired service.....	15
3.3 Kerberos Logical Infrastructure.....	15
3.3.1 Long-Term Symmetric Keys.....	17
3.3.2 Kerberos Ticket's.....	18
3.4 Kerberos Operation .....	22
3.4.1 Kerberos Message Exchange.....	23
3.4.2 Authentication Server (AS) Exchange.....	23

**TABLE OF CONTENTS**  
**(Continued)**

<b>Chapter</b>	<b>Page</b>
3.4.3 The Ticket-Granting Service Exchange.....	27
3.4.4 The Client/Server Exchange.....	28
3.5 Kerberos Services and Benefits.....	30
3.5.1 Kerberos Services.....	30
3.5.2 Kerberos Benefits.....	32
3.6 Limitations .....	34
4 SIM-KERBEROS.....	37
4.1 Two-Factor authentication.....	38
4.2 SIM Authentication.....	39
4.2.1 SIM Card Components.....	39
4.2.2 SIM Authentication Process.....	40
4.3 SIM Card and Java Card technology.....	41
4.4 SIM-Based Kerberos Design and Implementation.....	42
4.4.1 Overall Architecture.....	42
4.4.2 SIM-Kerberos Protocol.....	44
4.4.3 Key Generation.....	49
5 LDAP AND AUTHORIZATION.....	50
5.1 Introduction to LDAP.....	50
5.2 The Structure of LDAP.....	52

**TABLE OF CONTENTS**  
**(Continued)**

<b>Chapter</b>	<b>Page</b>
5.3 Authorization.....	54
6 PERFORMANCE AND POWER CONSUMPTION.....	56
6.1 Evaluation of the phone's specifications.....	56
6.2 Performance measurement using SPB Benchmark on Pocket PC Phone.....	56
6.3 Using JBenchmark to test the Java performance.....	59
6.4 Power Consumption.....	61
7 CONCLUSION AND FUTURE REASEARCH.....	64
7.1 Conclusion.....	64
7.2 Future Research.....	64
APPENDIX A KERBEROS IMPLEMENTATION.....	65
APPENDIX B LDAP IMPLEMENTATION .....	66
REFERENCES .....	68

## LIST OF TABLES

<b>Table</b>	<b>Page</b>
3.1 Kerberos Ticket Components.....	22
6.1 Smart phone Characteristics.....	56
6.2 SPB Benchmark Indices.....	57
6.3 Main test results.....	57
6.4 JBenchmark Result.....	60

## LIST OF FIGURES

<b>Figure</b>	<b>Page</b>
3.1 Kerberos authentication.....	9
3.2 Basic Key Exchange using a KDC.....	11
3.3 Needham-Schroeder Protocol.....	12
3.4 Kerberos authentication service.....	24
3.5 Kerberos ticket-granting service.....	27
3.6 Kerberos application server.....	28
4.1 SIM Authentication.....	40
4.2 Overall Architecture.....	42
4.3 Message flows of full-authentication of SIM-Kerberos protocol .....	45
4.4 Key Generation.....	49
6.1 Battery test result.....	58
6.2 Power Consumption for one authentication in five minutes.....	62
6.3 Power Consumption for five authentications in thirty minutes.....	63
A.1 User Interface .....	65
B.1 Active Directory.....	67

# CHAPTER 1

## INTRODUCTION

### 1.1 Motivation

The computing power of mobile devices increases year-by-year. A natural trend is that more and more resource intensive services are accessed through these devices because of the inherent advantages, and conveniences offered by portable devices. In line with these changes, mobile devices such as mobile phones and PDAs turn into versatile all-in-one personal service managers: They can be used for accessing contents, services, and applications provided via the Internet, storing personal information, important data, etc.

At the New Jersey Institute of Technology, we are creating the SmartCampus Test-Bed, a large scale mobile, wireless campus community system with a number of mobile, locatable, online community system applications.

Many users are reluctant to use applications that send sensitive data over wireless connections because they don't trust wireless security. Therefore, it is very important to ensure strict access control to the transaction data between clients and servers.

Solutions are emerging. The same protocol that makes secure traditional wired networks possible can also help make wireless transactions safe. The only one which is providing Single Sign-On and light weight protocol is Kerberos.

## 1.2 Thesis Statement

The goal of this thesis is to introduce research on lightweight, cross-platform single sign-on authentication architectures to bring mobile networks in line with WLAN networks. We design, implement, and analyze the SIM-Based Kerberos protocol for new cellphones with mobile and WLAN capability.

The rest of the thesis is organized as follows. In Chapter 2, we discuss the background of the SmartCampus project, single sign-on authentication and related works. In Chapter 3, we introduce the Kerberos Protocol and its implementation on the phone. In Chapter 4, we introduce mobile security and SIM-based authentication. In Chapter 5, we present the design and implementation of the SIM-based Kerberos protocol on the phone. In Chapter 7, we introduce the LDAP protocol, directory services and authorization for the SmartCampus middleware. In Chapter 8, we evaluate the performance and power consumption of our authentication architecture.



## CHAPTER 2

### BACKGROUND

#### 2.1 SmartCampus Project

SmartCampus is an ongoing project at the New Jersey Institute of Technology. At the New Jersey Institute of Technology we are creating the SmartCampus Test-Bed, a large scale mobile network which may include hundreds to thousands of users in a wireless campus community system that will serve as a dispersed living laboratory for the study of location-aware community systems using People-To-People-To-Places Services. The completion of the project would help us enhance services as the following in the campus community: e.g., NJIT students would be able to communicate with other students having similar interests, taking similar classes, to build social networks etc. So with the SmartCampus project we could use a range of technologies to locate individuals as they go about their daily activities. The availability of such technologies enables a new class of location aware information systems that link people to people to geographical places. It can strengthen the relationship between social networks and physical places. They can also help individuals collect location information to make new social ties and interact with existing ties. The main tool that is going to help us with this project would be the smart cell phone. It would contain an implementation of the campus mesh program.

A much clearer example for the SmartCampus project would be one where a student wants to play online computer game. Our Campus Mesh program on his smart campus cell phone he sends out a message asking for people to play with him together.

Then someone with a similar interest in the computer game will match the request and get his message. Now both could exchange messages and then play together online.

So the SmartCampus initiative features the development of software that, permits access to a database of individual user interests and daily routines for all participants to facilitate social interactions. Constructing this database of participant profiles involves collecting data from mobile communication devices never attempted before. These profiles will be leveraged in real time to support activities that will make life richer for all. Mobile applications often interact with multiple backend servers, pull information from them as needed, and assemble personalized displays for users. Each information service provider might have its own user authentication and authorization protocols. It is a major inconvenience for mobile users to sign on to each backend server manually.

## **2.2 Single Sign-on Authentication**

For example, our project runs many coexisting Java applications, each requiring authentication in order to access enterprise resources. It is best to implement single sign-on (SSO) security functionality to make authentication less intrusive for your users.

Fundamentally, single sign-on authentication means the sharing of authentication data. For instance, many students might need to access campus resources (database tables, for example) in order to fulfill their requirements, with different students requiring different resources depending on their function. Obviously, we need an authentication mechanism in place that can determine who is trying to access a particular resource. Once the authentication module knows the identity of the student, an authorization module

within the implementation can check whether the authenticated user has the necessary privileges to access the resource. Let's suppose that students use their usernames and passwords for authentication. The authentication module would thus have a database of usernames and passwords. Each incoming request for authentication would be accompanied by a username-password pair, which the authentication module would compare against the pairs in its internal database.

Our SmartCampus project may have several applications running within its scope. Different applications may form different modules of the same project. Each application is complete in itself, which means that it has its own user base, as well as several different tiers, including the back-end database, middleware, and a GUI for its users.

Before, usually people duplicate the authentication process to enable the cross-application authentication. For example, people duplicate the username and password databases on every application. This also means that the user will authenticate separately on different applications -- in other words, she will enter her username and password while accessing each of the applications, and the application will perform all the authentication steps again. In this way, the disadvantage is not only duplicating the username and password database but also duplicating the authentication process overhead. The amount of redundancy should be very obvious especially in the resource restricted mobile application.

One way to combat this problem is through the use of single sign-on service which is sharing authentication data between different applications. If a user is authenticated at one application, her authentication information is transferred to the second. The second

application accepts the authentication information without going through all the authentication steps. There is no redundancy in this solution. The only requirement is that the two applications trust each other so that each application accepts authentication data coming from the other. The advantage to using SSO is that users only need to remember one login name and password. They are less likely to complain about the necessity of it being complex or feel the need to write it down.

Normally, SSO is implemented as a separate authentication server. All applications that need to authenticate users rely on the SSO-based authentication server to check the identity of their users. Single sign-on servers manage user profiles and provide time-stamped access tokens, such as Kerberos tickets, to authenticated users. Service providers interact with single sign-on servers to validate tokens. Based on this authentication information, different applications then enforce their own authorization policies.

### **2.3 Related Work**

In this section, I will discuss the related work in the area of authentication. Authentication is so important that many authentication solutions have been published, but at a later time weaknesses and disadvantage were discovered in them. For examples,

#### **PKI**

A Public Key Infrastructure (PKI) is the key management environment for public key information of a public key cryptographic system. In general, there are three basic PKI architectures based on the number of Certificate Authorities (CAs) in the PKI, where

users of the PKI place their trust (known as a user's trust point), and the trust relationships between CAs within a multi-CA PKI. The most basic PKI architecture is one that contains a single CA that provides the PKI services (certificates, certificate status information, etc.) for all the users of the PKI.

Multiple CA PKIs can be constructed using one of two architectures based on the trust relationship between the CAs. A PKI constructed with superior-subordinate CA relationships is called a hierarchical PKI architecture. Alternatively, a PKI constructed of peer-to-peer CA relationships is called mesh PKI architecture.

### **PKINIT**

PKINIT is an extension of the Kerberos protocol (RFC1510), which enables use of public key cryptography for client authentication. The PKINIT mechanism hasn't been standardized yet and the specification is still under development directed by the IETF Kerberos WG. Implementation of the PKINIT mechanism is intended for the Heimdal open source implementation of Kerberos V5. It is based on the latest draft of the PKINIT specification and was created for Heimdal v.0.5 . Please remember the implementation published here is still under development and not intended for use in production environments.

### **EAP-SIM**

Extensible Authentication Protocol Method for GSM Subscriber Identity, or EAP-SIM, is an Extensible Authentication Protocol (EAP) mechanism for authentication and session key distribution using the Global System for Mobile Communications (GSM) Subscriber Identity Module (SIM). EAP-SIM is described in RFC 4186. Extensible

Authentication Protocol (EAP) mechanism is used for authentication and session key distribution using the Global System for Mobile Communications (GSM) Subscriber Identity Module (SIM). GSM is a second generation mobile network standard. The EAP-SIM mechanism specifies enhancements to GSM authentication and key agreement whereby multiple authentication triplets can be combined to create authentication responses and session keys of greater strength than the individual GSM triplets. The mechanism also includes network authentication, user anonymity support, result indications, and a fast re-authentication procedure.

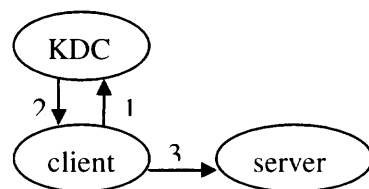
## CHAPTER 3

### KERBEROS

#### 3.1 Introduction

Kerberos authentication provides a mechanism for mutual authentication between a client and a server on an open insecure network. Kerberos the name came from Greek mythology in which it is the three-headed dog that guarded the entrance to Hades (or the world of dead). The name seems very appropriate as this is an Authentication mechanism first of all and by design has 3 main parts which are explained later.

Kerberos uses as its basis the Needham-Schroeder protocol. It makes use of a trusted third party, termed a Key Distribution Center (KDC), which consists of two logically separate parts: an Authentication Server (AS) and a Ticket Granting Server (TGS). 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.



**Figure 3.1** Kerberos authentication

## 3.2 Physical Architecture

The 3 main parts of the Kerberos Physical infrastructure are: Key Distribution Centre (KDC), Client User, and Server with the desired service to access.

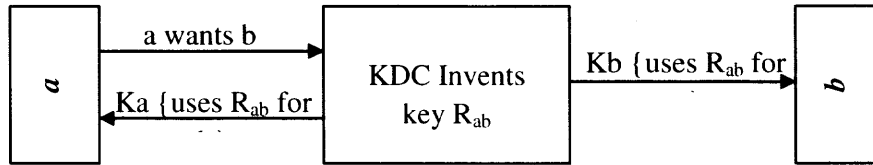
### 3.2.1 KDC

KDC (Key Distribution Server) is the heart of the complete Kerberos infrastructure. KDC is the central piece which implements the concept of Mediated Authentication.

As shown above KDC shares keys with all other components, users or services. All these are called nodes in the KDC realm. Now if  $a$  wants to talk to  $b$ , it will send a request to the KDC which shares keys with both  $a$  and  $b$ . KDC will authenticate  $a$  and choose a random number  $R_{ab}$  which will be used as key between  $a$  and  $b$ . Then it will encrypt  $R_{ab}$  with the key that KDC shares with  $a$  and send this to  $a$  and similarly it will encrypt  $R_{ab}$  with the key that it shares with  $b$  and send it to  $b$ . By this exchange both  $a$  and  $b$  now have the key  $R_{ab}$  that they will use for secure communication later between them.

This is the simplest form of key exchange that KDC performs; along with the Random number there a number of other things that KDC includes along with the encrypted random number and this whole package are called a ticket. Various other things like timestamp, expiry time etc are added to make the communication more secure and robust against various replay attacks. The exchange explained above is shown in the diagram below:





**Figure 3.2** Basic Key Exchange using a KDC

Now this exchange has a lot of issue:

- If a immediately sends a message to b based on the shared key, b may or may not have received the key by then it will not know how to decrypt this message.
- In this case as a is going to talk to b then it is unnecessary for KDC to talk to b as it is not going to communicate to b anyways after this exchange. So a lot of resources are wasted in setting up a secure session between KDC and b just for this one time.

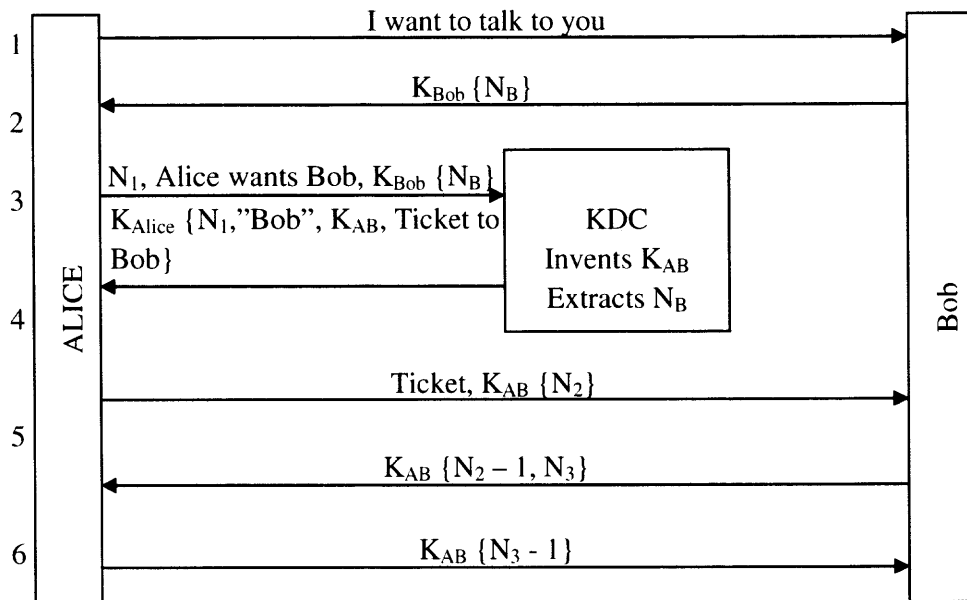
Kerberos uses a different approach in dealing with the issue of exchanging keys.

This is mainly based on the Needham-Schroeder exchange.

### **Needham-Schroeder Protocol**

In case of Needham-Schroeder Protocol the KDC has the Public Keys for all the nodes.

So this becomes the source of public keys and being a trusted third party it prevents the impersonation attacks that may happen in case we get the public key of any service for a non trusted source. The basic working of this protocol is as shown below:



**Figure 3.3** Needham-Schroeder Protocol

**Step1:** Alice sends Bob a message that she wants to talk to him

**Step2:** Bob selects a nonce  $N_B$  and encrypts it using his Key  $K_{Bob}$  that it shares with the KDC

**Step3:** Alice sends a message to KDC the message includes a nonce  $N_1$ , Request to talk to Bob and the message Bob sent to Alice in step 2

**Step4:** KDC checks  $K_{Bob} \{N_B\}$  and if valid sends a message to Alice encrypted with  $K_{Alice}$  that KDC shares with Alice. The message includes a nonce  $N_1$ , "Bob",  $K_{AB}$  and a ticket to Bob which is nothing but using Bob's key encrypted information  $K_{AB}$ , "Alice" and nonce  $N_B$  that was sent by Bob to Alice in the first place. This ticket will be used in the next step to authenticate Alice to Bob.

**Step5:** Alice sends a message to Bob that includes the ticket given by the KDC in the previous step and a nonce  $N_2$  encrypted using the shared key  $K_{AB}$

**Step6:** Bob sends a message to Alice, in which it encrypts using the shared key  $K_{AB}$ ,  $N_2 - 1$  and another nonce  $N_3$

**Step7:** Alice sends Bob the nonce  $N_3 - 1$  encrypted using  $K_{AB}$ . The last 2 exchanges are used for mutual authentication by Bob and Alice.

This protocol is secure against the replay attacks in case any attacker intercepts some of the messages, they cannot be used for creating any new sessions.

NOTE: This protocol can be reduced from 7 to 6 messages in case Alice in step 6 sends another message i.e.  $K_{AB} \{K_{Bob} \{N_B\}\}$ . By this there is no need to send the last message i.e. Step 7 as Bob already authenticates Alice in step 5 only.

There are several other ways in which the mediated authentication via a KDC can be implemented like Otway-Rees, Bellare-Merritt etc.

The main advantage of using a KDC is that it makes the 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 and in case some node is suspected of being compromised the setup again needs to change a just one place. The alternative of KDC will be for nodes to share keys between themselves depending on what service they may need access to.

Although the KDC mediated authentication has its advantages, it has its own 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.

KDC is the single point of failure. If it goes down nobody can access anything. This can be dealt with by having High availability (HA) KDC as is done practically, but that adds to the complexity of the already complex system.

The KDC can become a bottleneck as far as performance is considered because everyone frequently needs to talk to it. Having multiple KDC's can alleviate this problem, but then again the complexity may be an issue.

Moving the concept of KDC a bit further now we will look at the HDC structure as it exists in most practical Kerberos implementations: KDC is divided into 2 parts based on functionality. Each part provides services to the other part. The 2 parts are the Authentication Server (AS) and Ticket Granting Server (TGS).

#### **Authentication Server (AS)**

The AS issues TGTs good for admission to the ticket-granting service in its domain. Before network client's can get tickets for services, each client must get an initial TGT from the authentication service in the user's account domain.

#### **Ticket-granting service (TGS)**

The TGS issues tickets good for admission to other services in the TGS's domain or to the ticket-granting service of a trusted 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. If the client does not have a TGT valid for admission to that ticket-granting service, it must get one through a referral process that begins at the ticket-granting service in the user account's domain and ends at the ticket-granting service in the service account's domain.

### 3.2.2 Client User

Before explaining what a Kerberos client is we need to know *principal*. All entities within Kerberos, including users, computers, and services, are known as *principals*. Principal names are unique; a hierarchical naming structure ensures their uniqueness. Most KDC implementations store the principals in a database, so you may hear the term "Kerberos database" applied to the KDC.

"Kerberos client" is any entity that gets a service ticket for a Kerberos service. A client is typically a user, but any principal can be a client (unless for some reason the administrator has explicitly forbidden this principal to be a client). In the KDC explanation Alice is a client user.

### 3.2.3 Server with the desired service

All users use the Kerberos tickets to access services that are located usually on a remote server. This server needs to be a trusted one and be a part of the Kerberos realm.

## 3.3 Kerberos Logical Infrastructure

This section introduces the logical infrastructure, the key structure, the ticket types.

The various Physical components of Kerberos contain a number of logical attributes which are of prime importance for Kerberos to work.

The Kerberos protocol relies heavily on an authentication technique involving shared-secret keys. The basic shared-secret concept is quite simple: If a secret is known by only two people, then either person can verify the identity of the other by confirming that the other person knows the secret. For example, as explained in the

Needham-Schroeder protocol in the earlier session, suppose that Alice often sends messages to Bob, and that Bob needs to ensure that a message from Alice really has come from Alice before he acts on its information. They decide to solve their problem by selecting a password and agreeing to share the secret password between the two of them, but not with anyone else. If a message purported to be from Alice can somehow demonstrate that the sender knows the password, Bob can verify that the sender is indeed Alice.

The only question left for Alice and Bob to resolve is how Alice will show that she knows the password. She could include it somewhere in her messages, perhaps in a signature block at the end — Alice, Our secret. This would be simple and efficient and would be effective if Alice and Bob could be sure that no one else is reading their mail. Unfortunately, their messages pass over a network used by people like Carol, who uses a network analyzer to scan traffic in hope that one day she might spot a password. Thus, Alice must not prove that she knows the secret by including it in her message. To keep the password secret, she must show that she knows it without revealing it.

The Kerberos protocol solves this problem with secret key cryptography. Instead of sharing a password, communication partners share a cryptographic key, and they use knowledge of this key to verify one another's identity. In order for the technique to work, the shared key must be symmetric — that is, a single key must be capable of both encryption and decryption. One party proves knowledge of the key by encrypting a piece of information; the other proves knowledge of the key by decrypting the information.

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.

### 3.3.1 Long-Term Symmetric Keys

The long-term symmetric keys are derived from a password. The plaintext password is transformed into a cryptographic key by passing the text of the password through a cryptographic function. The result of the cryptographic function is the key.

**User keys** -When a user is created, the 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.

**System keys** - When a workstation or a server joins a Kerberos domain, it receives a password. In the same manner as a user account, the system account's password is used to create the system key.

**Service keys** - Services use a key based on the account password they use to log on. All KDC's in the same realm use the same service key.

Here a new term has been introduced and we need to define it.

**Realm** - Each principal is a member of a *realm*. By convention, a realm name is the DNS name converted to uppercase, so that the kerbttest.njit.edu domain becomes the KERBTEST.NJIT.EDU realm. Although uppercase realms are not obligatory, using a different case simplifies differentiating between domain names and realms. So essentially

realm is just a domain or an area in which everybody trusts the KDC and has an account with it. As of now this definition will suffice will discuss more when we move to Inter realm communication and transitive trust relationships.

### **3.3.2. Kerberos Ticket's**

The main component of Kerberos authentication is the ticket. The Kerberos messages are used to request and deliver tickets. There are two types of tickets used in Kerberos authentication, TGTs and service tickets.

#### **Kerberos Ticket Requests**

The Kerberos client sends the following ticket requests to the KDC:

TGT – Authentication Service

Service Ticket – Ticket-granting Service

#### **Ticket-Granting Ticket's**

The KDC responds to a client's authentication service request by returning a service ticket for itself. This special service ticket is called a ticket-granting ticket (TGT). A TGT enables the authentication service to safely transport the requester's credentials to the ticket-granting service.

A TGT is:

- A user's initial ticket from the authentication service
- Used to request service tickets
- Meant only for use by the ticket-granting service

TGTs are encrypted with a key shared by the KDC's. The client cannot read tickets. Only KDC server's 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 service (in this case the KDC) will use in communicating with the client. The TGT is encrypted with the KDC's long-term key.

From the client's point of view, a TGT is just another ticket. Before it 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 (described later in this document), and sends both the authenticator and the TGT to the KDC, along with a request for a service ticket for the service. In other words, gaining admission to the KDC is no different from gaining admission to any other service in the domain — it requires a session key, an authenticator, and a ticket.

From the KDC's point of view, TGTs enable it 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.

### **Service tickets**

A service ticket 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 — with the service's copy of the session key safely inside — becomes 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 over 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.

**Benefits of service tickets**

The server does not have to store the session key that it uses in communicating with client's. It is the client's responsibility to hold a ticket for the server in its credentials cache and present the ticket each time it wants access to the server. Whenever the server receives a service ticket from a client, the server can use its secret key to decrypt the ticket and extract the session key. When the server no longer needs the session key, it can discard it. The client does not need to go back to the KDC each time it wants access to this particular server. Service tickets can be reused. 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.

**Information client's have about tickets**

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. The main information within a Kerberos Ticket is summarized in the table below.

In the above diagram we see that client 1's network diagram is four octet long. The field is fixed length. Kerberos was designed to run in the network using TCP/IP protocol suite. This fixed length field means that this version of Kerberos cannot be used in a network with longer addresses. Ticket life time units are 5 minutes. So the maximum

ticket lifetime is a little over 21 hours. Time stamp tells you the time the ticket was created.

**Table 3.1** Kerberos Ticket Components

Ticket
Client 1's name
Client 1's instance
Client1's realm
Client 1's network layer address
Session key
Ticket life time
KDC's timestamp when ticket made
Client 2's name
Client 2's instance
Pad of 0s to make ticket length multiple of eight octets.

### 3.4 Kerberos Operation

The basic Kerberos authentication steps are:

- A client authenticates itself to the KDC by sending the pre authentication data.
- 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 "credentials" for a given server.
- The AS responds with these credentials, encrypted in 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 in 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 communication between the two parties or to exchange a separate sub-session key to be used to encrypt further communication.

### **3.4.1 Kerberos Message Exchange**

The Basic Kerberos exchange is divided into 3 parts:

Authentication Server (AS) exchange

Ticket Granting Server (TGT) exchange

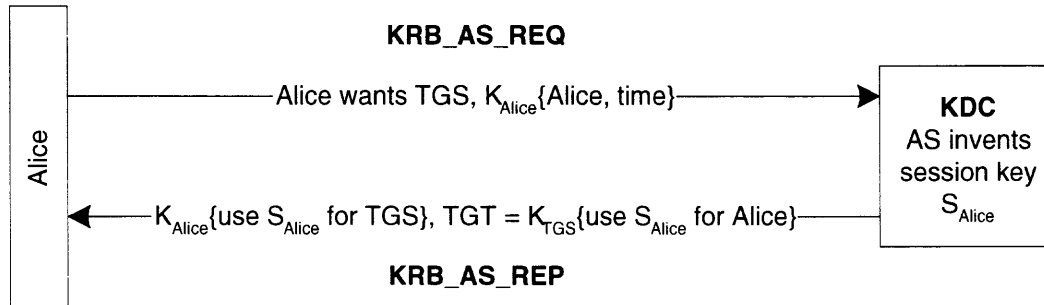
Client/ Server Exchange

### **3.4.2 Authentication Server (AS) Exchange**

The various messages in this exchange are:

#### **Kerberos authentication service request (KRB\_AS\_REQ)**

The client contacts the Key Distribution Center's authentication service for a short-lived ticket (a message containing the client's identity SIDs) called a ticket-granting ticket (TGT). This happens at logon.



**Figure 3.4** Kerberos authentication service

The Kerberos client on the workstation sends the message `KRB_AS_REQ` to the KDC. The message includes:

- 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 gets its copy of the user key from the user's record in its account database. When it receives a request from the Kerberos client on the user's workstation, the KDC searches its database for the user, pulls up the account record, and takes the user key from a field in the record.

This process — computing one copy of the key from a password, 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 TGS session key that it can use in subsequent transactions with the KDC during this logon session.

To verify the user during the complete login session the KDC decrypts the pre-authentication data and evaluates the timestamp inside. If the timestamp passes the test, the KDC can be assured that the pre-authentication data was encrypted with the user key and thus 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 authentication service (AS) constructs the TGT and creates a session key the client can use to encrypt communication with the ticket-granting service (TGS). 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 containing a service ticket for itself. This special service ticket is called a ticket-granting ticket (TGT). Like an ordinary service ticket, a TGT contains a copy of the session key that the service (in this case the 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 in the KDC's long-term key. The user's copy of the session key is encrypted in the user's long-term key.

The message includes:

- 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.

The TGT includes:

- A TGS session key for the KDC to use with the user
- 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 that reason, the TGS session key is often called a logon session key.

From the client's point of view, a TGT is just another ticket. Before the client attempts to connect to any service, the client first checks the user 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, it sends an authenticator and the TGT to the KDC, along with a request for a service ticket for the service. In other words, gaining admission to the KDC is no different from gaining admission to any other service in the domain — it requires a session key, an authenticator, and a ticket (in this case, a TGT).

From the KDC's point of view, TGTs enable the KDC 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 key only once, when it grants an initial TGT. For all other exchanges with this user, the KDC can decrypt the TGT with its own long-term key, extract the logon session key, and use that to validate the user's authenticator.

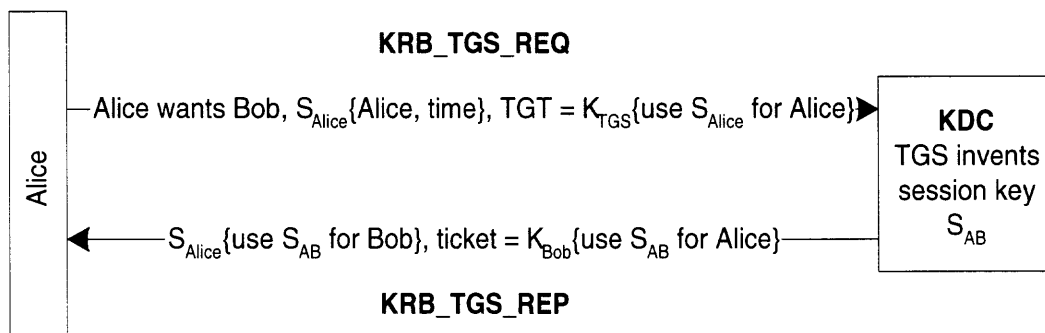


### 3.4.3 The Ticket-Granting Service Exchange

The various messages here are

#### Kerberos ticket-granting service request (KRB\_TGS\_REQ)

The client wants access to local and network resources. To gain access, the client sends a request to the TGS for a ticket for the local computer or some network server or service. This ticket is referred to as the service ticket or service ticket. To get the ticket, the client presents the TGT, an authenticator, and the name of the target server.



**Figure 3.5** Kerberos ticket-granting service

The message includes:

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. If these are acceptable, the TGS creates a service ticket. The client's identity is taken from the TGT and copied to the service ticket. Then the ticket is sent to the client.

The KRB\_TGS\_REP 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.

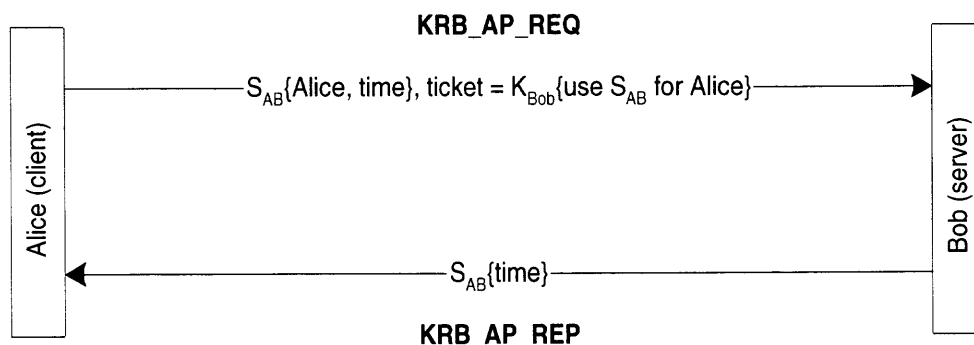
The service ticket includes:

- A session key for the computer to share with the user
- Authorization data copied from the user's TGT

### 3.4.4 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, requesting access. The server will decrypt the ticket and validate the authenticator.



**Figure 3.6** Kerberos application server

This message contains:

- An application option flag indicating whether to use 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 (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.

This is the complete flow for a Kerberos exchange in the case of a single realm. The things become complicated when we move to multiple realms or domains. In this case the trust needs to be between realms as well. Now in case there a number of realms between which we may want to communicate and use services we need trust between all of them and the number of keys increases a lot. We know that the keys are the weakest link of the whole scenario an ideal solution will be to keep them as minimum as possible.

In the next section we will look at a scheme called Transitive trust that comes to our rescue.

### **3.5 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.

#### **3.5.1 Kerberos Services**

##### **Authentication**

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**

By default Kerberos does not provide any authorization services, they are usually implemented as a separate procedure itself. 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 KDC sends to the client to be sent to the server. Once the server decrypts the ticket it has a list of services that this particular client can access and with what privilege level. The server uses these ACL's to authorize the servers future requests for resources.

### **Data Integrity**

Assuming that the client and service have authenticated each other using the Kerberos handshake and now each know the key for the current interaction (or session), we have all the pieces necessary to guarantee either data integrity. Since encryption is a costly operation in terms of time and CPU power, and we are only looking to ensure that the data is authentic; we need not encrypt all the data that is transmitted. Instead, an encrypted one-way hash is computed and transmitted with the plaintext data. Kerberos encrypts the much-shorter one-way hash, and bundles that together with the "plaintext" data, which is the original, unmodified message. The sender can then transmit this package to the receiver, who can look at the package, see what algorithm was used for the one-way hash and quickly compute the hash. Then the receiver can decrypt the received encrypted one-way hash and compare it with the hash that was just computed. If the two hashes match, the receiver knows exactly who sent the message and knows that the message was transmitted without modification.

### **Confidentiality**

There are always occasions when it is insufficient merely to know with whom you're talking and that no one can successfully change the conversation without being detected. 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 ciphertext. The sender transmits the package to the receiver, who decrypts the ciphertext 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 as the cost of computing, and encrypting the one-way hash is minor compared to the cost of encrypting the whole message.

### **3.5.2 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.

#### **Faster Authentication**

Kerberos protocol uses a unique ticketing protocol that provides faster authentication.

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

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.

#### **Mutual Authentication**

Kerberos supports Mutual authentication by which not only client authenticates itself to the server but the server can also authenticate to the client. So there is no assumption made that the servers are always trustworthy.

### **Open Source**

Kerberos is an Open Source protocol so essentially free to use. Because of being open source there is also much faster development done on all the fronts and it is being used and tested by a much larger user base.

### **Support for authentication Delegation**

What delegation really means is 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. You can use delegation for authentication in multi tier applications; an example of such an application is database access using a Web front end. In such a setup the browser, the Web server, and the database server are all running on different machines. In a multi tier application, authentication happens on different tiers. In such application if you want to set authorization on the database using the user's identity, you should be capable of using the user's identity for authentication both on the web server and the database server.

### **Support for Public Key Cryptography and One time pass codes**

Kerberos supports 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 in which each time to authenticate the user uses a different password so a password guessing mechanism or a Trojan horse program will never work as the password changes at each login.

### **3.6 Limitations**

This section introduces the limitations of the Kerberos.

#### **Denial of Service attacks**

Denial of Service attacks can occur in Kerberos, in which case the authentication service provided by the KDC can be denied to the clients and servers. The actual reasons for the DoS are beyond the scope of this paper as they deal with the actual Kerberos code and buffers used in it. Most of the DoS attacks are possible by buffer overflowing attacks on the KDC. Though this does not lead to any compromises of security of the clients and KDC but still can lead to significant damage because of the lost time for which no one is able to use the KDC.

#### **Password Guessing**

Password guessing attacks are not solved by Kerberos. In case the user chooses a poor password, it can be decrypted by an attacker using an offline dictionary attack. In order to mount an offline dictionary or brute force attack, some data that can be used to verify the



user's password is needed. One way to obtain this from Kerberos 5 is to capture a login exchange by sniffing network traffic.

In Kerberos 5 a login request contains preauthentication data that is used by Kerberos to verify the user's credentials when a TGT is issued. The basic preauthentication scheme in most of the Kerberos implementations contains an encrypted timestamp and a cryptographic checksum, both using a key derived from the user's password. The timestamp in the preauthentication data is ASCII-encoded prior to encryption and is of the form YYYYMMDDHHMMSSZ. This provides structured plain text that can be used to verify a password attempt: if the decryption result "looks like" a timestamp, then the password attempt is almost certainly correct. A password attempt that recovers a plausible timestamp can also be verified by computing the cryptographic checksum and comparing it to the one in the preauthentication data.

### **Time synchronization**

Each host on the Kerberos network needs to have its clock "loosely synchronized" to the KDC. This helps in reducing the book keeping overhead on the application servers, to do replay detection. The degree of looseness can be configured on a per server basis and is usually of the order of 5 minutes. This clock synchronization in itself needs to be secured because an attack on this can lead to DoS for the valid clients.

### **Principal Identifier Recycling**

The principal identifiers which identify a particular client or a server on the Kerberos network can be re used once deleted. The issue here is that in case the same principal name had some ACL's (Access Control Lists) associated with it which was not deleted

when the principal was deleted, then the new principal will inherit this ACL and will inadvertently gain access to resources.

These were the main limitations that Kerberos has of on now and but all the limitations and Vulnerabilities have workarounds. So a carefully designed Kerberos system can provide perfect security in a network environment.

## CHAPTER 4

### SIM-BASED KERBEROS

The open nature of university networks and the large amount of personal information they store make them prime targets for identity theft and other information security threats. Unfortunately, there are many ways for hackers to steal and misuse passwords. Our primary SmartCampus authentication uses Kerberos, which is based on Username and Password. Kerberos has the primary strengths and weakness of itself: Kerberos provides a means of verifying the identities of principals on an open network. This is accomplished without relying on authentication by the host operating system, without basing trust in the host addresses, without requiring physical security of all the hosts on the network and under the assumption that packets traveling along the network can be read, modified and inserted at will.

The ability of Kerberos to function in an open hardware and network environment is a unique strength. Kerberos suffers, however, from the same weakness that is characteristic of all traditional authentication paradigms: the reliable identification of the human component of the human/machine system. Kerberos requires only a single factor for user identification: a privately owned password which is used in conjunction with the user's public name. This password is quite susceptible to compromise and is the weakest link in an otherwise strong chain.

Two-factor authentication technology — “something you know” and “something you possess” — can be added to Kerberos Release 5 to provide a level of authentication

for the human component as secure as is available from Kerberos for the machine component. Using optional fields in the initial client-to-KDC (the “AS Request”) exchange, adding a pre-authentication flag and linking with appropriate authentication API are all required.

#### **4.1 Two-Factor authentication**

An authentication factor is a piece of information and process used to authenticate or verify a person's identity for security purposes. Two-factor authentication is a system wherein two different methods are used for authentication. Using two factors as opposed to one delivers a higher level of authentication assurance.

In order to understand two-factor authentication, it is important to understand the three methods by which people authenticate themselves to digital systems:

There are three universally recognized factors for authenticating individuals:

- 'Something you know', such as a password, or PIN.
- 'Something you have', such as a mobile phone, smart card or security token.
- 'Something you are', such as a fingerprint, DNA, or other biometric.

A system is said to leverage Two-factor authentication when it requires at least two of the authentication factors mentioned above. This contrasts with traditional password authentication, which requires only one authentication factor (knowledge of a password) in order to gain access to a application.

Common implementations of two-factor authentication use 'something you know' as one of the two factors, and use either 'something you have' or 'something you are' as the other factor.

For SmartCampus, we use a SIMcard - the card itself is a physical, "something you have" item, and the PIN is the "something you know" password that goes with it.

Using more than one factor is also called strong authentication; using just one factor, for example just a static password, is considered as weak authentication.

## **4.2 SIM Authentication**

A Subscriber Identity Module (SIM) card is part of a removable smart card ICC (Integrated Circuit Card) for mobile cellular telephony devices such as mobile computers and mobile phones. SIM cards securely store the service-subscriber key (IMSI) used to identify a GSM subscriber.

### **4.2.1 SIM Card Components**

SIM cards store specific information used to authenticate and identify subscribers on the Network, the most important of these are International Mobile Subscriber Identity (IMSI), Integrated Circuit Card ID (ICCID), Ciphering Key Generating Algorithm (A8), Authentication Algorithm (A3), Encryption Algorithm (A5), Individual Subscriber Authentication Key (Ki), Encryption Key (Kc).

The secret key Ki is 128 bits long and is used for two things: Generate the secret response (SRES) to a Random challenge and Generate the 64 bit session key Kc, used for over the air encryption.

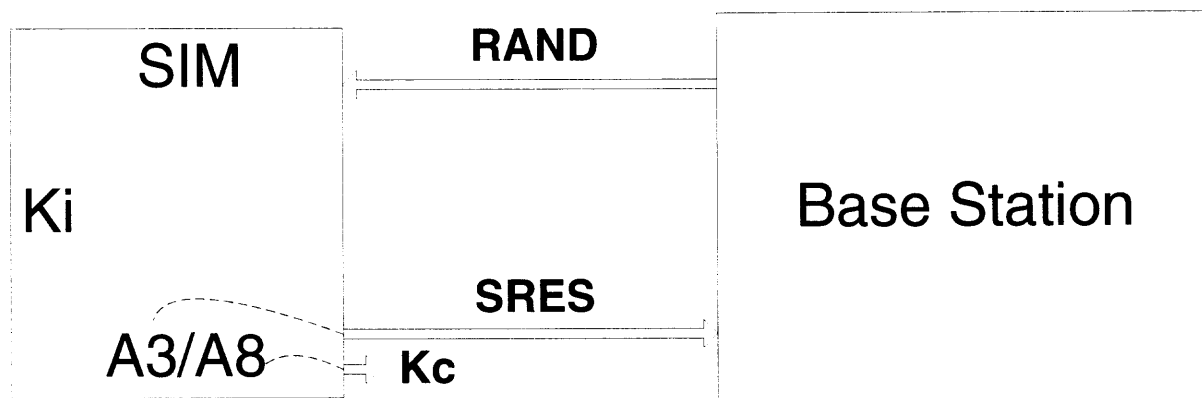
A3: It is the authentication algorithm used in GSM systems. COMP128 is widely used by GSM service providers.

A5: This is the encryption algorithm. There are different versions of this algorithm with A5/1 being the strongest for over the air privacy. A5/x, A5/2 are weaker versions of this algorithm. There is also another version that uses no encryption at all; it is the A5/0 algorithm.

A8: It is the key generation algorithm. Most of the service providers just like the A3 algorithm use COMP128.

#### 4.2.2 SIM Authentication Process:

GSM authentication is based on shared cryptographic primitives  $K_i$  on SIM and Base Station. The authentication process is shown below:



**Figure 4.1** SIM authentication

During authentication, AC generates a random number that it sends to the mobile. Both mobile and AC use the random number, in conjunction with subscriber's secret key and a ciphering algorithm called A3, to generate a number SRES that is sent back to the AC. If number sent by mobile matches number calculated by AC, then subscriber is authenticated.

### 4.3 SIM Card and Java Card technology

SIM uses operating systems which come in two main types: Native and Java Card. Native SIMs are based on proprietary, vendor specific software whereas the Java Card SIMs are based on standards, particularly Java Card, which are a subset of the Java programming language specifically for embedded devices. Java Card allows the SIM to contain programs that are hardware-independent and interoperable. I will use the Java Card's power to build SIM-based security to the Kerberos mobile application.

All Java Cards are essentially smart cards with one extra feature. The Java Card technology allows different people to use the Java language to develop smart card applications that are hosted on an individual card. For example, if the SIM card in your cell phone is a Java Card, it could contain value-added Java Card applications

Also Java Card technology offers an open architecture for smart card application development. A Java Card contains Java Card Virtual Machine, (JCVM) and a set of APIs (collectively known as the Java Card API). Some of the classes and interfaces in the Java Card API are exposed for use by J2ME MIDlets and other client applications.

I will use Java Card technology with J2ME devices, although Java Card technology is not limited to the J2ME platform. The Security and Trust Services API (SATSA) enables the use of Java Card technology in J2ME devices. A Java Card application could work as an authentication module in a J2ME-based SmartCampus middleware. The SmartCampus middleware would allow users to access their accounts using their cell phones. The Java Card application on the Java Card would contain the

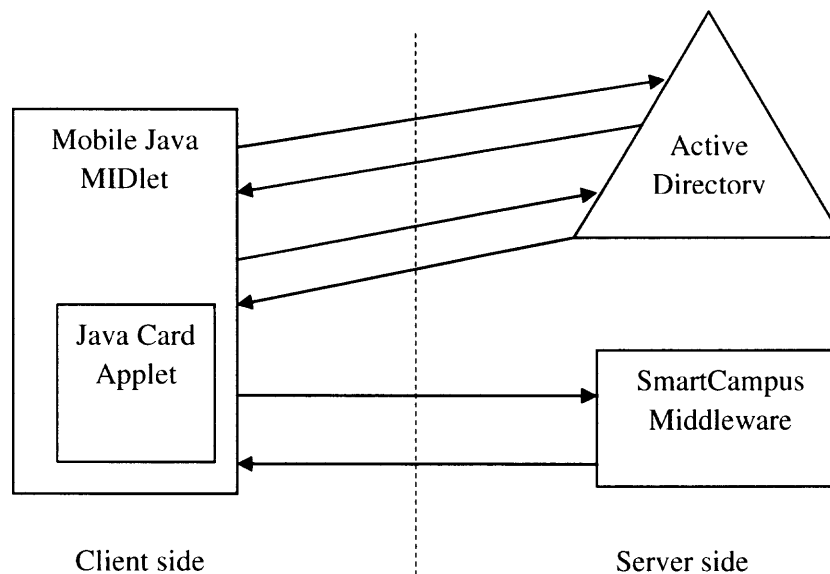
authentication logic that ascertains who is trying to access an account. The J2ME device would contain a MIDlet that would present an easy-to-use GUI for account access.

#### 4.4 SIM-Based Kerberos Design and Implementation

In most cases, as was discussed before, we could see security breach occurs for either of Kerberos or SIM authentications. We present a novel authentication solution that protects user login by providing a double-criterion user authentication through the use of existing mobile phones. The solution is to combine SIM-Based authentication and Kerberos authentication to add another layer of security of our Kerberos-based J2ME mobile application. It overcomes the security inadequacy of the single authentication method and maximizes the security for mobile users.

##### 4.4.1 Overall Architecture

The architecture is developed in following components:



**Figure 4.2** Overall Architecture



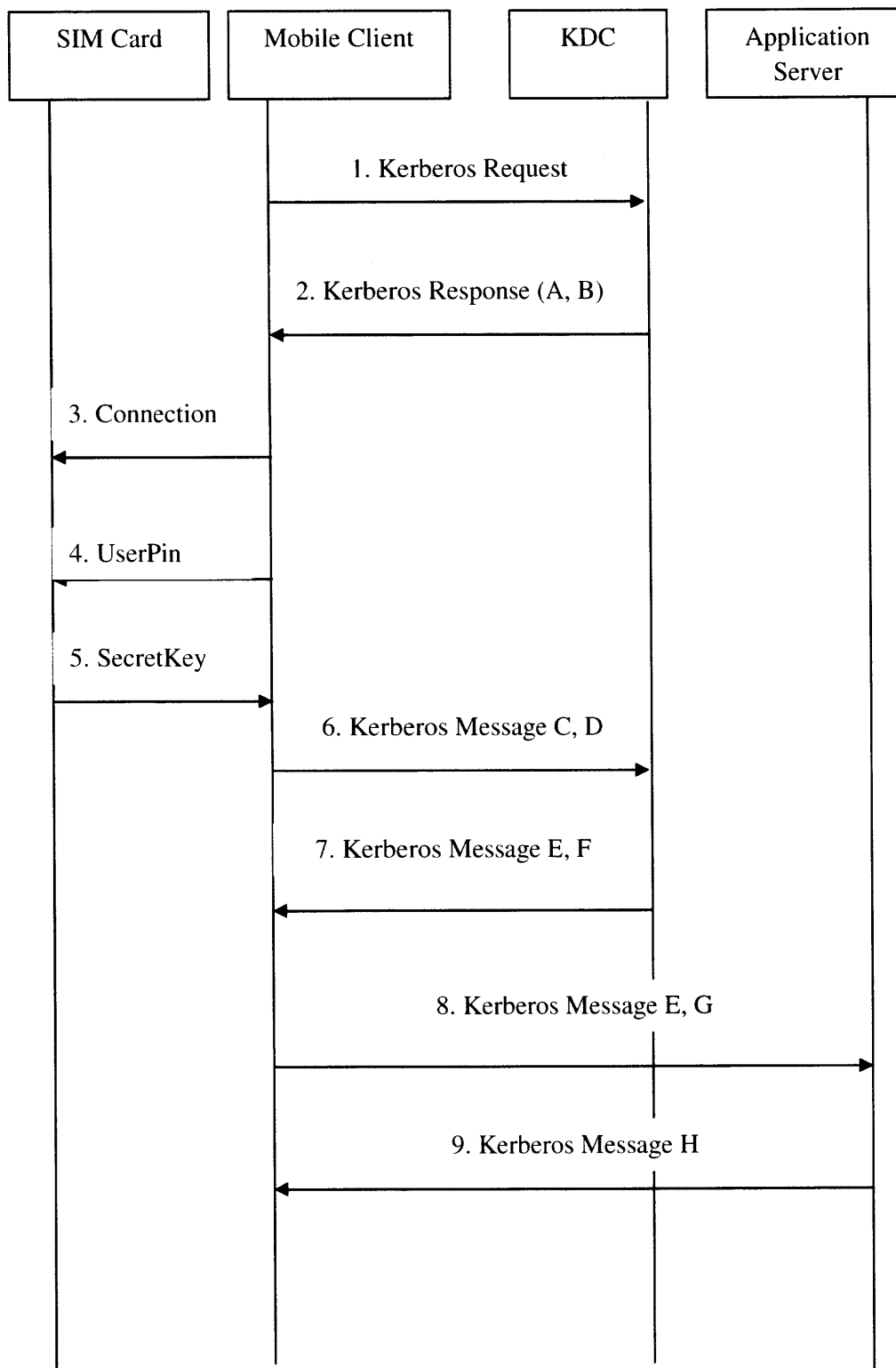
The first part is the SIM Card which we will develop using a Java Card applet called JavaCardKerberosKey. The JavaCardKerberosKey applet works as a secret key manager. It contains the Kerberos secret keys  $K_i$ ,  $K_c$  that are used to secure communication between a J2ME cell phone and the KDC. The secret key generated with  $K_i$  is used to decrypt the encrypted portion of a TGT. The encrypted portion contains the session key, which an application can only extract using the Kerberos secret key. Therefore, knowing the secret key is essential for using a TGT.

The second part is a MIDlet running on the phone which is called KerberosSmartCampus. This MIDlet has three functions. First, it is working as a client to communicate with the Kerberos Server. Second, it is providing a graphical user interface to the campus mesh messenger. Third, KerberosSmartCampus MIDlet is also a client of the JavaCardKerberosKey applet, which means that the KerberosSmartcampus MIDlet will use the JavaCardKerberosKey applet to decrypt the encrypted portion of the TGT and extract the session key. The KerberosSmartCampus MIDlet will request a TGT from the SmartCampus's KDC server. On receipt of the TGT, the KerberosSmartCampus MIDlet will extract the encrypted portion of the TGT and transfer it to the JavaCardKerberosKey applet. The KerberosSmartCampus MIDlet will also provide the user's password to the JavaCardKerberosKey applet. The JavaCardKerberosKey applet will use it with  $K_i$  to decrypt the encrypted portion of the TGT, extract the session key and generate  $K_c$ . The KerberosSmartCampus MIDlet will use the  $K_c$  to secure communication with the SmartCampus server.

The third part of the application is the Key Distribution Center which will serve as the authentication server for the mobile client. The implementation of KDC runs on a physically secure node somewhere in the network and a library of subroutines that are used by distributed applications which work to authenticate their users. We built it using Active Directory.

The last one is the Application Server which integrates with SmartCampus middleware and runs a sample application to demonstrate the user login with the CampusMesh messenger. We build the middleware using Apache Tomcat with OSGI.

#### **4.4.2 SIM-Kerberos Protocol**



**Figure 4.3** Message flows of full-authentication of SIM-Kerberos protocol

1. The J2ME mobile phone user invokes the KerberosSmartCampus MIDlet in a J2ME cell phone and provides the username and password to the KerberosSmartCampus MIDlet. The MIDlet sends a clear-text message along with its identity to the KDC requesting services on behalf of the user. For Example, "User XYZ would like to request services". Note: Neither the secret key nor the PIN is sent to the KDC.

2. The KDC checks to see if the user is in its database. If so, the KDC sends back the TGT to the KerberosSmartCampus MIDlet. It contains following two messages:

Message A: Ticket Granting Ticket: [client, address, validity, Ksess]Ktgs.

Message B: [Ksess]Km

3. When the KerberosSmartCampus MIDlet gets the Kerberos ticket granting ticket (TGT) from a Kerberos distribution center (KDC), the MIDlet will extract this encrypted portion from the TGT. The KerberosSmartCampus MIDlet is going to extract the session key from the encrypted portion. The session key will allow the KerberosSmartCampus MIDlet to further communicate with the SmartCampus middleware. The MIDlet will communicate with the SIM card which is implemented by a JavaCardKerberosKey applet. The KerberosSmartCampus MIDlet will ask SATSA to select the JavaCardKerberosKey applet. The MIDlet will provide the AID of the JavaCardKerberosKey applet to SATSA and ask it to create a synchronous connection with the applet.

4. The MIDlet will pass the UserPIN to the JavaCardKerberosKey applet (the SIM card) after the connection is established. The SIM in the mobile will use the UserPin and

existing MobilePin(Ki) to generate the key. The applet performs a one-way function on the entered key, and this becomes the secret key of the mobile client

5. The JavaCardKerberosKey applet sends the secret key to the MIDlet. Now that the client has messages A and B and the secret key, it decrypts message A to obtain a key Kt (Note: The client cannot decrypt the Message B, as it is encrypted using KDC's secret key.) The key used to encrypt the data on the link, client/KDC session key Kc, is generated from the Kt and Ki by running the encryption algorithm. This session key Kc is used for further communications with KDC. At this point, the client has enough information to authenticate itself to the KDC.

6. When requesting services on the Middleware server, the MIDlet sends the following two messages to the KDC:

Message C: Ticket Granting Ticket: service, [client, client address, validity, Ksess]Ktgs.

Message D: Authenticator: [client, timestamp]Kc.

7. Upon receiving messages C and D, the KDC retrieves message B from message C. It decrypts message B using the KDC secret key, then computes the client/KDC session key Kc. Using this key, the KDC decrypts message D (Authenticator) and sends the following two messages to the client:

Message E: Ticket (client, service): service, [client, client address, validity, Kcs]Ks.

Message F: [Kcs]Kc.

8. Upon receiving messages E and F from KDC, the client has enough information to authenticate itself to the Middleware Server. The client connects to the Middleware Server and sends the following two messages:

Message E from the previous step: Ticket (client, service): service, [client, client address, validity,  $K_{cs}$ ]  $K_s$ .

Message G: a new Authenticator: Authenticator : [client, timestamp]  $K_{cs}$ .

9. The Middleware on the application server decrypts the ticket using its own secret key and sends the following message to the MIDlet to confirm its true identity and willingness to serve the MIDlet:

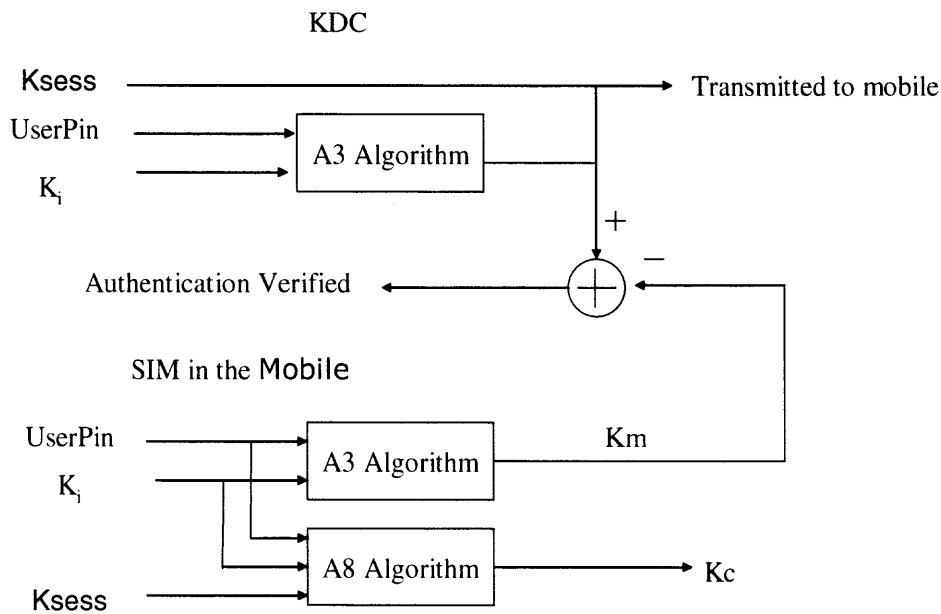
Message H: the timestamp found in client's recent Authenticator plus 1, encrypted using  $K_{cs}$ .

Finally, the client decrypts the confirmation using the client/server session key and checks whether the timestamp is correctly updated. If so, then the client can trust the server and can start issuing service requests to the middleware server. The middleware will provide the requested services to the client.

To summarize the above steps, the first five authenticate the mobile client to the KDC. In steps three, four and five, the MIDlet communicates with the SIM card to generate the client/KDC session key  $K_c$ . For steps six and seven, the mobile client receives a client/server key  $K_s$  from KDC to use with the server application. In steps eight and nine, the mobile client exchanges the key  $K_c$  with the server application. After that, all communications are encrypted with this key.

### 4.4.3 Key Generation

The detailed key generation process is shown below:



**Figure 4.4** Key Generation

1. The mobile obtains the IMSI from the SIM card, and passes this with username to the KDC requesting authentication.
2. The KDC searches its database for the incoming username, IMSI and its associated password and  $K_i$ .
3. The KDC then generates a session key  $K_{sess}$  and encrypts it along with TGT using the  $K_i$  associated with the password.
4. The KDC then sends the  $K_{sess}$  along with the TGT to the Mobile phone, which passes it to the SIM card. The SIM card decrypts the  $K_{sess}$  with its  $K_i$ , password, producing the encryption key  $K_c$ .
5.  $K_c$  is used to encrypt all further communications between the Mobile client and the KDC

## **CHAPTER 5**

### **LDAP AND AUTHORIZATION**

It is important to have the authorization policy after the user has been authenticated by the SmartCampus middleware. We decided to use LDAP for our authorization scheme not only for its support for group authorization but also the capacity for directory service. Smartcampus LDAP directory service is really a database that stores information about all objects in the network. The mobile directory service is like a phone book for the mobile network. For example, to find a resource on the network, it will not require remembering where it is located, but just do a search in directory to find that resource. These resources include users, groups, activities, and interests, to name a few.

#### **5.1 Introduction to LDAP**

LDAP, or "Lightweight Directory Access Protocol", is similar to X.500 in DAP: both have an information model and a protocol for querying and manipulating it. The major difference is that the LDAP protocol itself is designed to run directly over the TCP/IP stack, and it lacks some of the more esoteric DAP protocol functions. The IETF designed and specified LDAP as a better way to make use of X.500 directories - having found the original Directory Access Protocol (DAP) too complex for simple internet clients to use. The common term "LDAP directory" can mislead. No specific type of directory is an "LDAP directory". One could reasonably use the term to describe any directory accessible using the LDAP protocol and which can identify objects in the directory with X.500



identifiers. Directories such as OpenLDAP, though primarily designed as native repositories optimized for access by LDAP rather than as a gateway to X.500 protocols as was provided in ISODE, are nevertheless no more "LDAP directories" than any other directory accessible by the LDAP protocol.

An LDAP directory entry consists of a collection of attributes with a name, called a distinguished name (DN), which refers to the entry unambiguously. Each of the entry's attributes has a type and one or more values. The types are typically mnemonic strings, like "cn" for common name, or "mail" for email address. The values depend on the type, and most non-binary values in LDAPv3 use UTF-8 string syntax. For example, a mail attribute might contain the value "user@example.com".

LDAP directory entries feature a hierarchical structure that reflects political, geographic, and/or organizational boundaries. In the original X.500 model, entries representing countries appear at the top of the tree; below them come entries representing states or national organizations. Typical LDAP deployments use DNS names for structuring the top levels of the hierarchy. Further below might appear entries representing people, organizational units, printers, documents, or just about anything else.

Just as a Database Management System is used to process queries and updates for a relational database, an LDAP server is used to process queries and updates to an LDAP information directory. In other words, an LDAP information directory is a type of database, but it is not a relational database. Unlike databases that are designed for processing hundreds or thousands of changes per minute, LDAP systems are heavily optimized for read performance.

So LDAP is particularly useful for storing information that needs to be read from many locations, but updated infrequently. For example:

- The phone book, group chart and related staff
- Infrastructure services information, including NIS maps, email aliases, and so on
- Personal Information such as birthday, address, school, hobby and so on
- Public certificates and security keys

## 5.2 The Structure of LDAP

The protocol accesses LDAP directories follow the X.500 model:

- A directory is a tree of directory entries.
- An entry consists of a set of attributes.
- An attribute has a name (an attribute type or attribute description) and one or more values. The attributes are defined in a schema.
- Each entry has a unique identifier: its Distinguished Name (DN). This consists of its Relative Distinguished Name (RDN) constructed from some attribute(s) in the entry, followed by the parent entry's DN. Think of the DN as a full filename and the RDN as a relative filename in a folder.

LDAP uses schemas to define what attributes an object can and must have. An entry in our LDAP implementation looks like this when represented in LDAP Data Interchange Format (LDIF):

```
dn: cn= Qing Zhu,dc=smartcampus, dc=njit, dc=edu  
cn: Qing Zhu
```

```
givenName: Qing
sn: Zhu
telephoneNumber: 973 596 5366
telephoneNumber: 973 596 5366
mail: qz22@njit.edu
advisor: cn=Sotirios Ziavras,dc=smartcampus, dc=njit, dc=edu
objectClass: inetOrgPerson
objectClass: groupPerson
objectClass: person
objectClass: top
```

dn is the name of the entry; it's not an attribute nor part of the entry. "cn=Qing Zhu" is the entry's RDN, and "dc=smartcampus, dc=njit, dc=edu" is the DN of the parent entry, where dc denotes Domain Component. The other lines show the attributes in the entry. Attribute names are typically mnemonic strings, like "cn" for common name, "dc" for domain component, "mail" for e-mail address and "sn" for surname.

A server holds a subtree starting from a specific entry, e.g. "dc=smartcampus, dc=njit, dc=edu" and its children. Servers may also hold references to other servers, so an attempt to access "ou=department,dc=njit,dc=edu" could return a referral or continuation reference to a server which holds that part of the directory tree. The client can then contact the other server. Some servers also support chaining, which means the server contacts the other server and returns the results to the client.

LDAP rarely defines any ordering: The server may return the values in an attribute, the attributes in an entry, and the entries found by a search operation in any order. This follows from the formal definitions - an entry is defined as a set of attributes, and an attribute is a set of values, and sets need not be ordered.

### 5.3 Authorization

LDAP can securely delegate read and modification authority based on your specific needs using ACL (Access Control List). ACLs can control access depending on who is asking for the data, what data is being asked for, where the data is stored, and other aspects of the record being modified. Using LDAP ACLs, we have following functions:

1. Granting users the ability to change their phone number, address and interests, while restricting them to read-only access for other data types.
2. Granting anyone in the group "Admin" the ability to modify any user's information for the following fields: title, name, ID number, group name, and organization name. There would be no write permission to other fields.
3. Denying read access to anyone attempting to query LDAP for a user's password, while still allowing a user to change his or her own password.
4. Granting Admin read-only permission for phone numbers, while denying this privilege to anyone else.
5. Granting anyone in the group "Admin" to create, delete, and edit all aspects of host information stored in LDAP.

6. Allow people to selectively grant or deny themselves read access to subsets of the friends contact database. This would, in turn, allow these individuals to download the friend contact information to their local laptops or to a phone.

7. Allow any group owner to add or remove any entries from groups they own. For example, this would allow Admin to grant or remove access for people to modify Web pages. Restrictions can also be based on phone ID or domain name.

## CHAPTER 6

### PERFORMANCE AND POWER CONSUMPTION

#### 6.1 Evaluation of the phone's specifications.

In order to analyze performance and power consumption, we ran micro benchmarks to measure the SmartCampus authentication on the phone. The experiments were performed on Sprint 6700 Pocket PC phone.

**Table 6.1** Smart phone Characteristics

CPU Class	ARM9 CPU
MHz	495 MHz
Battery Type	1350 mAh Lithium Ion Polymer rechargeable
Memory	64 MB built-in RAM, 128 MB Flash ROM
Operating System	Windows Mobile 5

#### 6.2 Performance measurement using SPB Benchmark on Pocket PC Phone.

Developed by SPB Software House, SPB Benchmark is now the industry standard for benchmarking PPC Phones. It combines tests of real-world functions such as Internet access, playing videos, and editing Pocket Word files with tests of the synthetic processor, memory and graphics, in order to establish a detailed picture of a PPC Phone's performance.

Our experiments included tests of four of the SPB Benchmark's scores: (1) The CPU index, which tests processor-heavy functions such as file compression and the transfer of data to and from memory; (2) The file system index, which reports the speed of copying and moving 10K and one-megabyte files in RAM; (3) The graphics index, which

reports the speed that 2D images are displayed on the screen and which has direct relevance for assessing multimedia performance; and (4) the platform index, which measures Pocket Word, Pocket Internet Explorer and File Explorer performance. Finally, the overall index captures the results of all these tests as well as the results of a relatively simple arcade game test. All the indices are normalized to the speed of a Compaq iPAQ 3650, which gets a 1000 on all scores. In addition to these tests of the phone's processor performance, we also used the SPB Benchmark to test our PPC Phone's battery life. Using SPB's typical use test, we set the backlight to 100 percent before taking the measurements in order to make sure that the results would be consistent with its performance in a user's daily life.

SPB Benchmark results are in the table below:

**Table 6.2** SPB Benchmark Indices

Spb Benchmark index	328.94	(iPAQ 3650 scored 1000)
CPU index	1600.38	(iPAQ 3650 scored 1000)
File system index	132.25	(iPAQ 3650 scored 1000)
Graphics index	2154.11	(iPAQ 3650 scored 1000)
ActiveSync index	-	(iPAQ 3650 scored 1000)
Platform index	374.77	(iPAQ 3650 scored 1000)

**Table 6.3** Main test results

Test	Time	Speed	% of iPAQ 3650 speed
Write 1 MB file	1620 ms	632 KB/sec	80%
Read 1 MB file	266 ms	3.76 MB/sec	21%
Copy 1 MB file	1851 ms	553 KB/sec	70%
Write 10 KB x 100 files	4644 ms	220 KB/sec	39%
Read 10 KB x 100 files	515 ms	1.94 MB/sec	31%
Copy 10 KB x 100 files	4562 ms	224 KB/sec	47%
Directory list of 2000 files	1643 ms	1.22 thousands of files/sec	1%
Internal database read	645 ms	1552 records/sec	368%
Graphics test: DDB BitBlt	4.97 ms	201 frames/sec	748%

Graphics test: DIB BitBlt	30.5 ms	32.8 frames/sec	243%
Graphics test: GAPI BitBlt	3.81 ms	262 frames/sec	122%
PocketWord document open	33579 ms	7.76 KB/sec	25%
Pocket Internet Explorer HTML load	3292 ms	7.52 KB/sec	57%
Pocket Internet Explorer JPEG load	1889 ms	134 KB/sec	254%
File Explorer large folder list	3489 ms	573 files/sec	111%
Compress 1 MB file using ZIP	4989 ms	203 KB/sec	191%
Decompress 1024x768 JPEG file	582 ms	483 KB/sec	151%
Arkaball frames per second	6.4 ms	156 frames/sec	144%
CPU test: Whetstones MFLOPS	5196 ms	0.072 Mop/sec	155%
CPU test: Whetstones MOPS	1600 ms	39.4 Mop/sec	115%
CPU test: Whetstones MWIPS	10414 ms	4.8 Mop/sec	161%
Memory test: copy 1 MB using memcpy	10 ms	99.6 MB/sec	142%

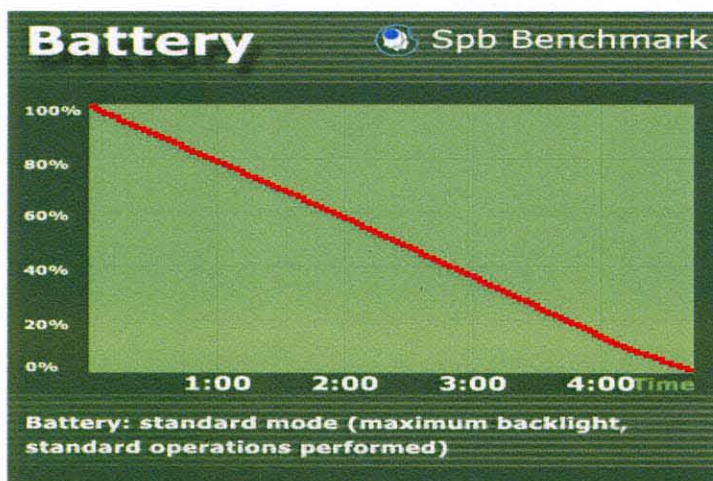


Figure 6.1 Battery test result



### 6.3 Using JBenchmark to test the Java performance

We employed JBenchmark, which is 100% Java-based and is used to evaluate Java ME-enabled mobile devices, to test the IBM J9 JVM system on the phone. JBenchmark can be incorporated into the development process and can also be used independently for testing application performance. JBenchmark has the following features:

- Simple and user-friendly
- Quickly creates benchmark scripts
- Common Testing Constructs, HttpGet, Random or Weighted choice of tests, ThreadGroups, and Loops
- Dynamically generates URLs; randomly sets variables based on a given list of values or ones from JDBC Source
- Uses Jelly scripting
- Various assertions for response status, and response size, ensuring that positive performance indicators are not the result of page errors
- Extensible Result Publishers

The table below shows the estimate of Java VM acceleration on the mobile device.

**Table 6.4** JBenchmark Result

<b>Test results</b>	
JAR reading	148
Image loading(PNG)	71
<b>HTTP specific details</b>	
WAP Profile	
HTTP User Agent	MIDP-2.0 Configuration /CLDC-1.1
<b>JVM general</b>	
Total Memory	1572864
<b>JVM display</b>	
Color Display	True
Double Buffered Screen	True
Number of Colors	65536
Alpha Levels	2
Form Size	240x268
Canvas Size	240x268
Canvas Full Size	240x268
GameCanvas Size	320x320
GameCanvas Full Size	320x320
<b>JVM configuration</b>	
ME Configuration	CLDC-1.1
ME Profiles	MIDP-2.0
CLDC Version	1.1
Platform	Windows CE 5.1, Pocket PC, PA10A1
ME Encoding	CP1252
<b>JVM platform</b>	
MIDP Version	2.0
MSA version	No
ME Locale	en-US
<b>JVM CDC</b>	
Java Installation Directory	J9MIDP20
Java Class Path	/My Documents/temp/N1889033690346V.jar
Operating System Name	Windows CE
Operating System Architecture	arm
Operating System Version	5.1 build 195
List of Paths to Search when Loading Libraries	J9MIDP20bin;jtwi_version=No
Default Temp File Path	TEMP

## 6.4 Power Consumption

Battery life is extremely important for cell phone performance, especially for those users who do not have frequent opportunities to re-charge them or to swap in fresh batteries. For some cell phone users, every minute that can be squeezed from the battery counts. This section presents the experimental results from tests of the phone's power consumption, demonstrating that our authentication functionality is highly efficient and will not affect the daily battery life. Outlined below are the three major steps of the authentication process:

*Step One:* The authentication uses a key generation function to calculate the user key and SmartCampus key.

*Step Two:* Next, the authentication uses key and encryption algorithms to encrypt the message, and uses a hash function to make a MAC of the message and append it to the packet.

*Step Three:* the authentication exchanges messages with the server via a WiFi network, decrypting each message in order to securely retrieve the content.

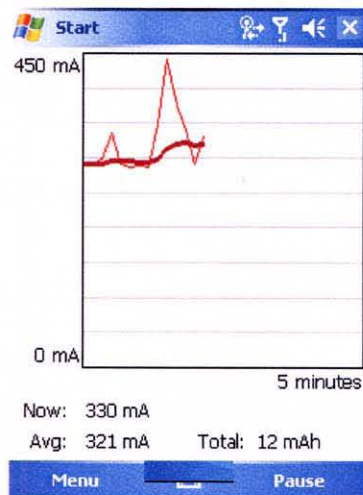
In our experimental methodology, once the encryption is complete, the message was sent over the network to a server as a Kerberos message. This procedure was repeated at regular intervals, and the results indicated that the CPU and WiFi are the largest drains on the power supply. We wanted to know how much current a PDA draws from the battery, and to have that data in real time.

In order to find out how much power a PDA draws from the battery, we used acbPowerMeter and acbTaskMan software to display the power consumption values.

acbPowerMeter is a freeware application that shows the current consumption numerically and graphically, both in real time and over the course of a given amount of time. acbPowerMeter measures and displays the total and average current consumed since the application started or was last reset.

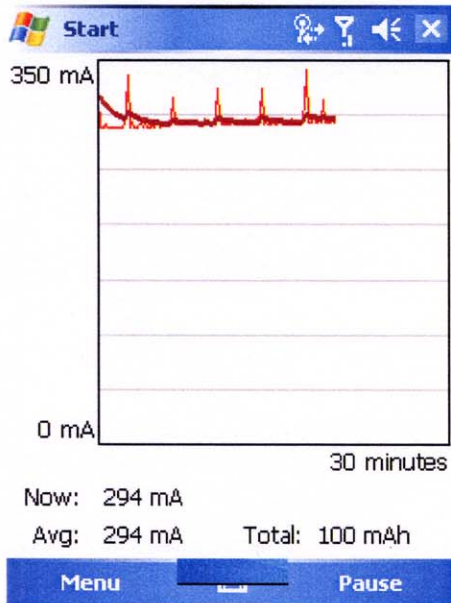
For our testing protocols, I downloaded acbPowerMeter, installed and started it, in order to see what would happen when I started a program in the foreground. In particular, I need to make sure that the wireless transmission was enabled on the phone, and that it was not connected to any power source (including USB-based recharging.) acbPowerMeter can run in the background without any risk of it interfering with our application, and causes only minimal overhead. After running the application in question for 30 to 40 seconds, I terminated the program and returned to acbPowerMeter in order to evaluate the results. The output shows the Amperage used when the tested application was active.

The chart below shows the power consumption when one authentication was done in five minutes.



**Figure 6.2** Power Consumption for one authentication in five minutes

This chart below shows the power consumption for five authentications in thirty minutes



**Figure 6.3** Power Consumption for five authentications in thirty minutes

### Analysis

In the two distinct charts above, the horizontal axis shows the time and the vertical axis displays the Amperage. The thin line indicates the battery load in milliamps at any given time, while the thick line shows the average power consumption. In the first chart, approximately 300 mA was used in three minutes, whereas in the second chart, 300 mA was used over the course of twenty minutes. Although we can see a pulse Ampere when we send out a message, we can conclude that the authentication would not consume much power because it did not have much impact on the line indicating the average power.

## **CHAPTER 7**

### **CONCLUSION AND FUTURE RESEARCH**

#### **7.1 Conclusion**

Most of today's technology conscious students use mobile computing devices such as smart phones, laptops and PDAs to carry out their every day activities. Such technologies and associated applications raise novel and significant challenges in terms of security, privacy and trust which are of considerable importance to our student population. This research plan designed and implemented a novel two-factor SIM-Kerberos authentication and authorization scheme. Such systems provide a fast, lightweight, double secure, cross platform application for solving trustworthy computing issues including software security, reliability, mobility, and user privacy. In addition, the analysis showed this fast, less memory and power consumed security application is ideal for daily use on the communication restricted and computation restricted mobile equipments.

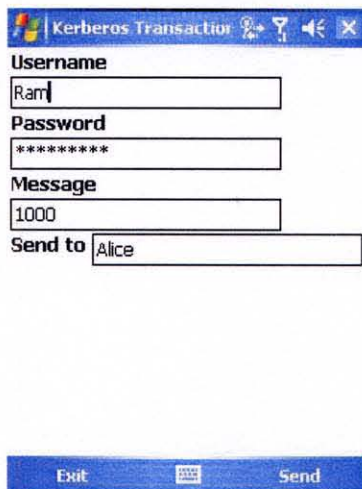
#### **7.2 Future Research**

Future work can be done in authentication and authorization two aspects. First, the Extensible Authentication Protocol (EAP) protocol can be included in the authentication scheme to provide an end to end secured wireless communication channel including client, AP, and server. In other aspect, the Role Based Access Control (RBAC) model can be involved into the authorization module which will provide a delegate control based on user's role and even context aware control.

## APPENDIX A

### KERBEROS IMPLEMENTATION

The entire core Kerberos functions are written in JAVA Micro Edition (J2ME). They are implemented under Connected Limited Device Configuration (CLDC) 1.1, Mobile Information Device Profile (MIDP) 2.0 specification and run on the IBM J9 JVM on our mobile phone. J2ME provide a good platform to implement mobile application as well as Java Card applicaton. J2ME includes some security primitives for code signing and to Support application security, but it is not enough to support our complicated secure key exchange. In addition to J2ME, we add Bouncy Castle Library support to implement our cryptographic APIs. Bouncy Castle provides a smaller footprint for use where resources are limited and less functionality is required as well as support to a range of well-known cryptographic algorithms. For the server side, we use Windows Server 2003 with Active Directory to host the KDC. All users' names, passwords and phone number are stored in KDC - Active Directory. The user interface is shown below:



Kerberos Transaction

**Username**  
Ram

**Password**  
\*\*\*\*\*

**Message**  
1000

**Send to** Alice

Exit Send

Figure A.1 User Interface

## **APPENDIX B**

### **LDAP IMPLEMENTATION**

We created the LDAP using Active Directory in the computer with windows 2003 Server, it worked perfectly. We were able to run it without any problems and get the expected results. The main goal of active directory is to provide directory service and access control to the clients. For example say in our college environment, and there are students and teachers. There are certain groups only the teachers can access. So it's the duty of the active directory to sort out who has the rights and who doesn't. So the active directory when built using the windows is a perfect match of a purpose.

Active Directory objects store access control permissions in security descriptors. A security descriptor contains two ACLs used to assign and track security information for each object: the discretionary access control list (DACL) and the system access control list (SACL).

DACLs identify the users and groups that are assigned or denied access permissions on an object. If a DACL does not explicitly identify a user, or any groups that a user is a member of, the user will be denied access to that object. By default, a DACL is controlled by the owner of an object or the person who created the object, and it contains access control entries (ACEs) that determine user access to the object.

SACLs identify the users and groups that you want to audit when they successfully access or fail to access an object. Auditing is used to monitor events related to system or network security, to identify security breaches, and to determine the extent



and location of any damage. By default, a SACL is controlled by the owner of an object or the person who created the object. A SACL contains access control entries (ACEs) that determine whether to record a successful or failed attempt by a user to access a object using a given permission, for example, Full Control and Read.

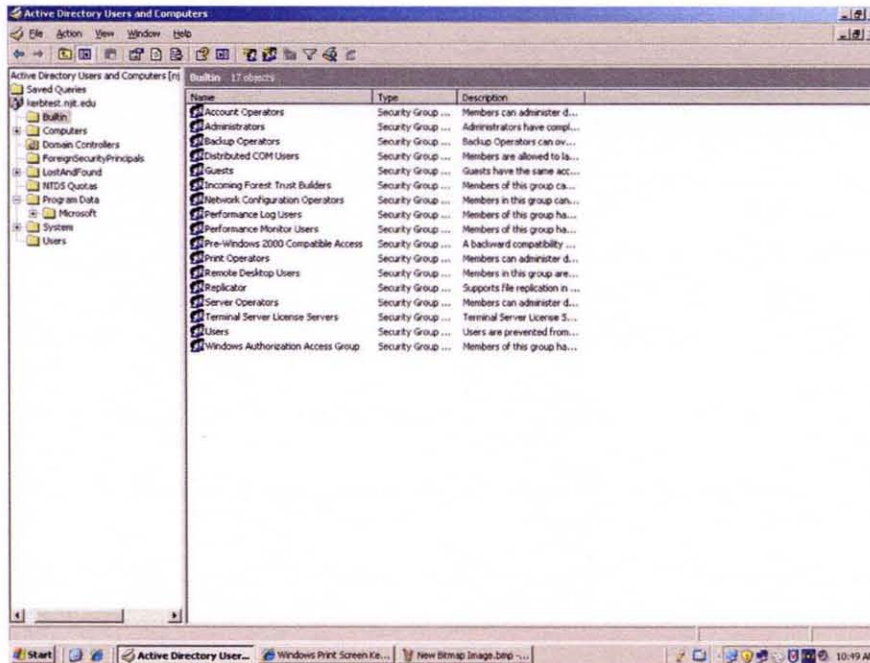


Figure B.1 Active Directory

## REFERENCES

1. C. Neuman, T. Yu, S. Hartman, K. Raeburn, "The Kerberos Network Authentication Service (V5)", IETF, RFC 4120, July 2005.
2. T. van Do, et al, "Offering SIM Strong Authentication to Internet Services", White Paper, 3GSM World Congress, Barcelona, February 2006.
3. BC Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33-38, 1994.
4. Sun Developer Network. Java platform, micro edition (java me). Retrieved November 6 from World Wide Web: <http://java.sun.com/javame/>.
5. B. Tung, C. Neuman, and J. Wray. Public key cryptography for initial authentication in Kerberos. Internet Draft, April 2000.
6. Alan Harbitter Daniel A. Menascé. *The Performance of Public Key-Enabled Kerberos Authentication in Mobile Computing Applications*, 2005
7. Arjun Anand, Constantine Manikopoulos, Quentin Jones, and Cristian Borcea. 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.
8. Faheem Khan, Securing Java Card applications Retrieved November 10, 2007 from the World Wide Web: <http://www.ibm.com/developerworks/java/library/wi-satsa/>
9. H. Haverinen, J. Salowey, "EAP-SIM Authentication", RFC 4186, IETF, January 2006.
10. MIT, Kerberos. Retrieved August 2, 2007 from the World Wide Web: <http://web.mit.edu/Kerberos/>.
11. J. Garman, "Kerberos: The Definitive Guide", 2nd ed. California: O'Reilly, 2003. [E-book] Available: Safari e-book.
12. C. Kaufman, R. Perlman, M. Speciner, *Network Security: Private Communication in a Public World*, Second Edition, New Jersey: Prentice Hall, 2002, pp. 307-371.
13. 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.msp?mfr=true>.

14. Jan De Clerc, "Windows Server 2003 security infrastructures: Core Security Features (HP Technologies)," 1st ed. Digital Press, 2004. [E-book] Available: Safari e-book.
15. IBM, WebSphere Everyplace Micro Environment. Retrieved November 1, 2007 from the World Wide Web: <http://www-306.ibm.com/software/wireless/weme/>
16. 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/>
17. Bouncy Castle Cryptography. Retrieved November 1, 2007 from the World Wide Web: <http://www.bouncycastle.org/>