CHAPTER 1 : INTRODUCTION

1.1. Background

Health care is one of the most important thing in our life. Disease or illness can really mean a down turn in our life. The biggest asset we can have in life is the health. Health care is the management of treatment offered by medical, nursing, dental or any other related service to patient for his any health problems. When we talk about the care of health, we are talking of all goods and services that are produced to improve our health. They may be curative, preventative or even palliative solutions. A system of health care is organized to give health services to a large population or a group of people.

Depending on how the system is organized the health care can be for an individual or for a large group of people. In developed countries, the health care system is designed as such that it should cover all the people; whether poor or rich. In society, people are worried about the kinds of systems there are, to deal with issues of health. However, the systems are lacking in regard to flaws. In developing countries, people usually take care of health as an individual thing and, if you do not have enough money, you might not get access to quality care. There are so many disparities and, some systems in certain countries are worse; not able to deal with demand of health. Health is not a cheap affair; you have to have a good system if you want it to work for you. Governments have the responsibility to create or formulate policies that will favor people in this regard. Good systems of health can be set up by the top most leadership of a state.

If technology can be used to simplify and speed up the healthcare, it will lead to improved and higher quality health care facilities for majority.

1.2. Motivation

A healthcare system is an essential requirement for places with mass population. An efficient system with reliable patient record, and secure health flow is required for the care to reach to the right patient at right time. In the developing countries like India the health flow in the public hospitals is based on an OPD (Out Patient Data) Card. The patients have to register for a doctor standing in long queues. Once the doctor sees a patient, prescription is written manually on the card, or may be suggested to move to another department for investigations, test. There is no mapping if the OPD card belongs to a specific patient. There can be security flaws if the OPD card is utilized by an unauthorized patient. Also if the OPD card is lost all patient records are lost. There is a need to store information digitally. Work is still in process for the electronic patient record management. One of the challenges is that record management should be acceptable and be adopted uniformly across different hospitals. At least the patient should be able to retain records electronically as Personal Health Records and use it for secure records. The records could be either be retained electronically on some external source or retained on a mobile ¬device frequently retained by a patient.

In spite of well networked health care system, access to healthcare in rural areas is far from satisfactory. In the current scenario, 75% of the qualified consulting doctors practice in urban, 23% in semi-urban (towns) and only 2% in rural areas where as the vast majority of population live in the rural areas. However increasing the number of doctors alone is not the solution. Accepting that organic solutions will not meet the tyranny of numbers in India, the government needs to adopt technology as a means to deliver healthcare to rural area as technology has often provided a platform for enhancing the provision of quality healthcare and for driving down the health expenditure. [1]
If technology can be used to simplify and speed up the healthcare, it will lead to improved and higher quality health care facilities for majority.

1.3. Thesis Outline

Thesis consists of following chapters:

Chapter 2 explains studies carried out in literature.

Chapter 3 research background is given. This chapter explains the Kerberos Protocol and its concepts.

Chapter 4 explains the Java Authentication and Authorization Service (JAAS) with its architecture, classes and interfaces.

Chapter 5 explains the Java RMI and its various components.

Chapter 6 is for Proposed Architecture and various workflows in architecture are explained.

Chapter 7 gives the implementation details of the project and explains link between different modules of the project.

Chapter 8 shows the gives testing results in the form of snapshots, logfile, tables, etc.

At last we summarize the thesis with conclusion and future work in **Chapter 9**.

CHAPTER 2 : LITERATURE SURVEY

Kerberos Assisted Authentication in Mobile Ad-hoc Networks

Asad Amir Pirzada et.al., in their work Kaman [2], Kerberos Assisted Authentication in Mobile Ad-hoc Networks, proposed the secure authentication scheme for ad-hoc networks. In Kaman there are multiple Kerberos servers used for distributed authentication and load balancing. Also, in Kaman the secret key or password is only known to user while server will have cryptographic hash of the user password. The Kaman servers on periodic basis or on-demand, replicate their databases with each other. Also, a replication sequence number is associated with each replication.

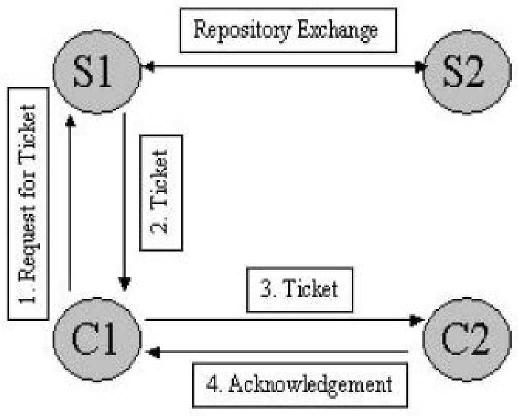


Fig.1 : Operations of Kaman [2]

The communication between two clients is done using the session key which is acquired by client from one of the server. While server generates the key and encapsulates it in ticket to send it requested client. In the end the client use this ticket to establish secure session between two parties. Also the availability check feature minimizes the malicious attacks.

CTES based Secure approach for Authentication and Authorization of Resource and Service in Clouds

Sanjeev Kumar Pippal et.al.[3], in their work proposed modifications to Kerberos and explained the Collaborative Trust Enhanced Security Model with the messages involved in the process of user authentication and authorization for distributed cloud services. The proposed model is more efficient despite of increase number of messages. Also overhead in keep track of active users of network has been reduced. For this , they have introduced the coordinator systems in proposed model. They ascertained that this model overcomes the drawbacks of Kerberos such as password guessing attack, platform dependency, etc.

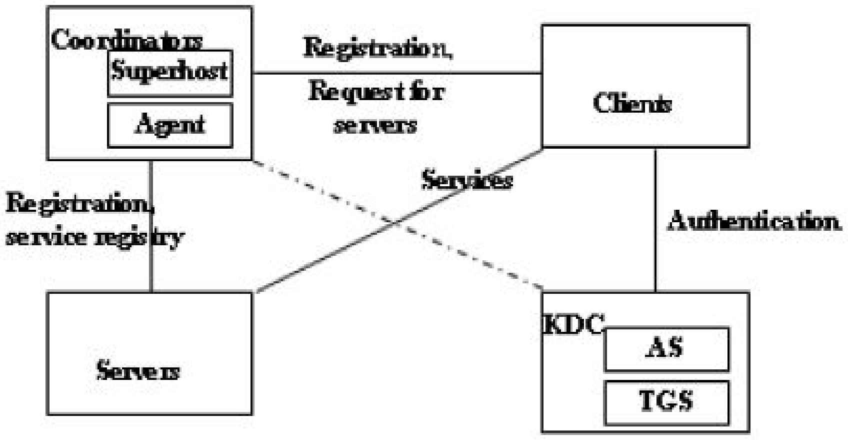


Fig. 2 : Components of CTES Model [3]

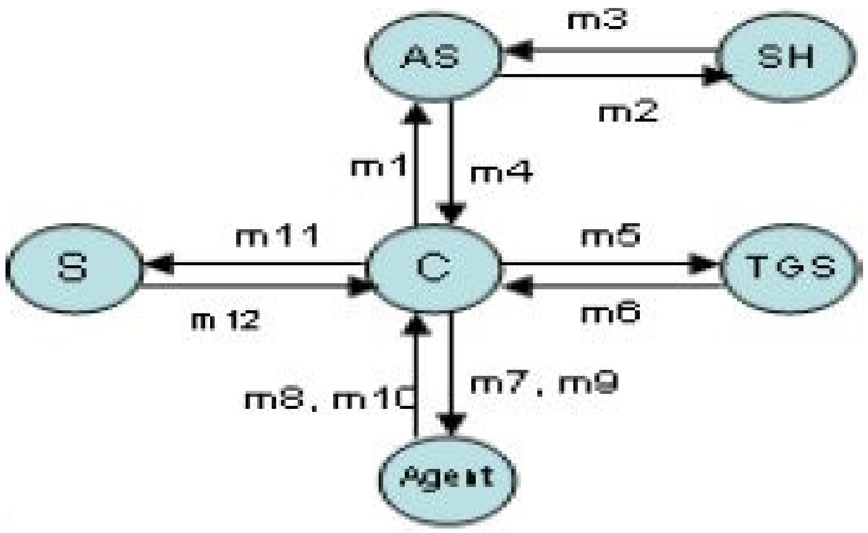


Fig. 3 : Messages used in CTES Model [3]

Socket VS RMI

SeungJun Bang et.al[4], in their work evaluated the performance of two distributed communication mechanism i.e. Socket and RMI. The performance of each mechanism is evaluated on the basis of its processing speed with increase number of computers.

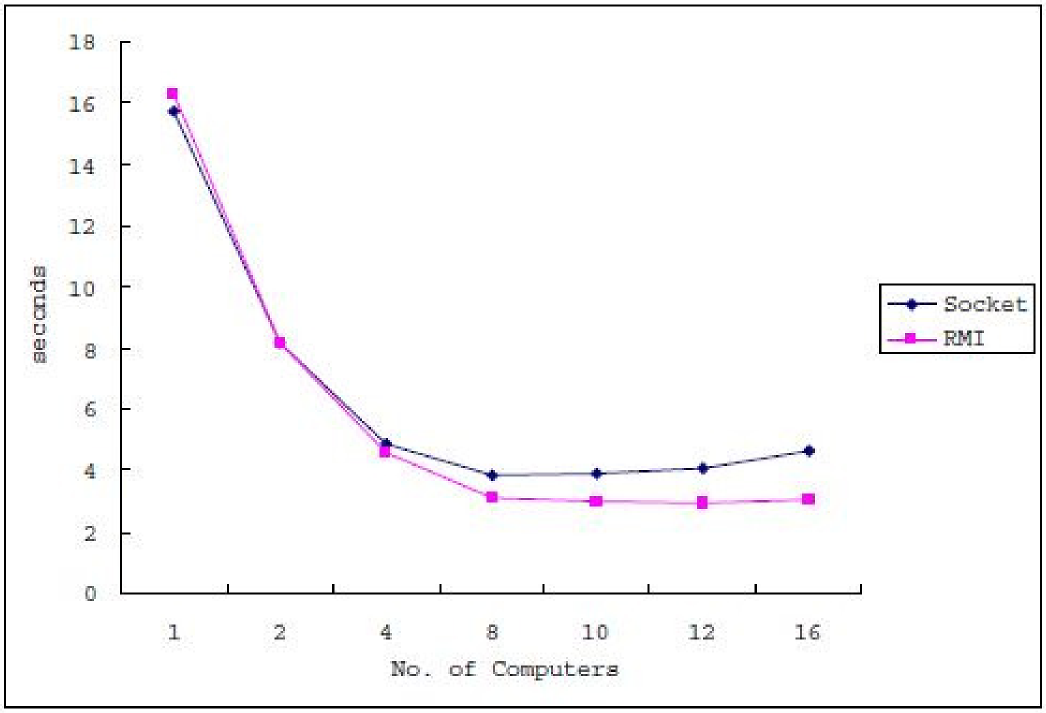


Fig. 4 : Performance evaluation of Calculating Pi application on two different JMPI implementations [4]

No. of Computers	Socket (sec.)	RMI (sec.)
1	35.422	35.250
2	19.125	12.219
4	13.406	13.453
6	31.406	22.469
8	40.594	31.485

Table 1 : The Processing Speed according to No. of Computers used for executing SOR application [4]

The performance results shows that the two mechanisms gets their speedup differently with respect to communication patterns of applications and the number of computers used.

CHAPTER 3 : RESEARCH BACKGROUND

3.1. Kerberos [5]

Kerberos is an authentication service that enables clients and servers to establish authenticated communication. Kerberos provides a secure way of authentication over insecure networks. To prove the identity of both end users and network servers, the Kerberos uses encrypted tickets instead of sending clear plain text passwords.

3.1.1. Requirements

Secure: Kerberos should be strong enough to protect weak link from opponent

Reliable: Kerberos should be highly reliable with one system able to back up another

Transparent: The client/user should not be aware that authentication taking place, expect to enter a password for login

Scalable: The System should be capable of supporting large number of clients and service providers.

3.1.2. Kerberos V4 [7][8]

Kerberos V4 was the first version of Kerberos distributed by MIT. The basics of Kerberos V4 protocol are documented in the Athena Technical Plan. The first three versions of Kerberos is no more in use. While Version 4 and Version 5 are conceptually similar, but slightly different from one another. Version 4 is simple and has better performance, but works only with TCP/IP, while Version 5 has more functionality. Although, Kerberos V4 could not be exported outside the United States because of some export control restrictions on encryption software. Still there are several implementations of Kerberos V4 implementation exist. Now, the original MIT Kerberos 4 implementation is in maintenance mode and officially considered as dead.

3.1.3. Kerberos V5 [7][8]

To add new features and security enhancements which were not present in Version 4, the Kerberos V5 is developed. The list of features added in Version 5 are as follows:

- A better wire protocol, based on ASN.1
- Credential forwarding and delegation
- Replay cache
- More flexible cross-realm authentication
- Extensible encryption types
- Pre-authentication

Apart from the MIT's Kerberos V5, many other implementations of Kerberos V5 have been developed, some are commercial and some open source such as Heimdal.

3.2. Description of how Kerberos works [6]

The client authenticates itself to the Authentication Server (AS) which forwards the username to a Key distribution center (KDC). The KDC issues a Ticket Granting Ticket (TGT), which is time stamped, encrypts it using the user's password and returns the encrypted result to the user's workstation. This is done infrequently, typically at user logon; the TGT expires at some point, though may be transparently renewed by the user's session manager while they are logged in.

When the client needs to communicate with another node ("principal" in Kerberos parlance) the client sends the TGT to the Ticket Granting Service (TGS), which usually shares the same host as the KDC. After verifying the TGT is valid and the user is permitted to access the requested service, the TGS issues a Ticket and session keys, which are returned to the client. The client then sends the Ticket to the service server (SS) along with its service request.

3.3. Kerberos Terminology and Concepts

3.3.1. Realms, Principals and Instances [8]

Kerberos installation contains entities such as individual users, computers and services. Each one has a unique principal associated with it. Also, each principal has associated long term key, which can be a password or passphrase. The principal is divided into the hierarchical structure to accomplish the global unique names.

Each principal is starts with a username or service name called **Primary**. Optionally the username or service name is followed by **Instance**. The instance is used in two situations : for service principals and for administrative use. The username and optional instance, together form a unique identity in respective realm. Every Kerberos installation defines an administrative realm of control that is distinct from other Kerberos installation. Kerberos define this by the realm name. Generally, realm name is given DNS domain name in uppercase.

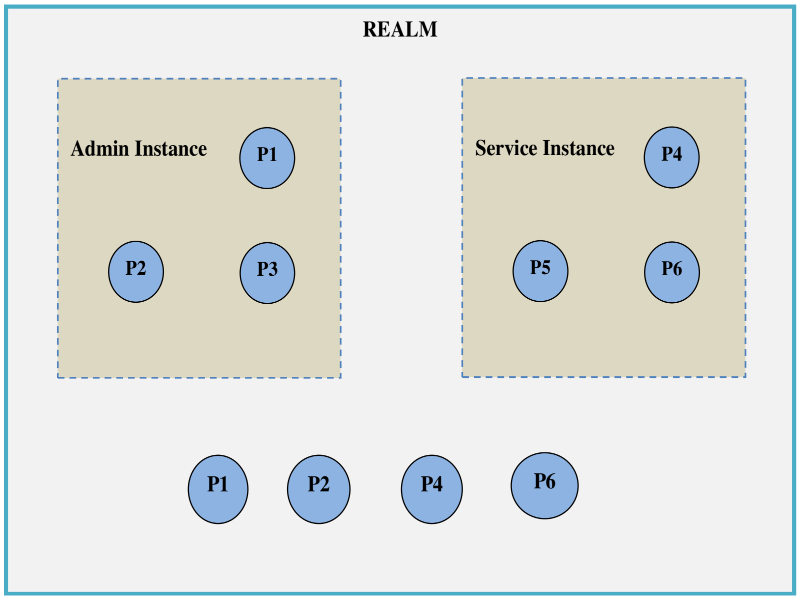


Fig. 5 : Relation between Primary, Instance and REALM

Primary	**/**	**Instance (Optional)**	**@**	**Realm**

Fig. 6 : Typical Form of Kerberos V5 Principal

For example, DTU owns domain name dce.ac.in then for its users it would create a Kerberos realm as DCE.AC.IN. While the user with name Jack will have principal such as jack@DCE.AC.IN. This simple form of principal is valid in both Kerberos 4 and Kerberos 5. Whereas if Jack is also admin in realm DCE.AC.IN then the principal would be jack/admin@DCE.AC.IN.

3.3.2. Service and host principals [8]

Not only the users but also the hosts and servers offering Kerberos services have principals. As, in Kerberos, each endpoint of connection can request mutual authentication, so both endpoints require an identity and a key.

Well, the service principal and user principals are slightly different. In service principal, the username component is the service that the principal represents. While in case of host principal, the username is host. To distinguish service principals for the same service and on different hostnames, the instance component contains the hostname of the machine the service principal is located on. Service that use Kerberos authentication are said as Keberized.

3.3.3. Kerberos 4 principals [8]

Kerberos 4 principals are made up of three elements : the username, an optional instance, and the realm. The username and instance are separated by a period, and the username and instance are separated from the realm by an @ symbol.

For example, jack.admin@DCE.AC.IN

In general, the forms of Kerberos 4 principal are :

- user[.instance]@REALM
- service.hostname@REALM

3.3.4. Kerberos 5 principals [8]

Kerberos 4 principals has same elements like Kerberos 4 principals. While in place of a single instance component of Kerberos 4, the Kerberos 5 may contain several sub instance components. Also, the Kerberos 5 uses a forward slash to separate the username and instance components instead of dot in case of Kerberos 4.

For example, jack/admin@DCE.AC.IN

In general, the Kerberos 5 principals have the following format:

component[/component][/component]....@REALM

Practically there are two types of Kerberos 5 principals such as

- username[/instance]@REALM
- service/fully_qualified_domain_name@REALM

3.3.5. The Key Distribution Center (KDC) [8]

The integral part of the Kerberos System is KDC. It consists three components: a database which consists all principals and its associated encryption keys, the Authentication Server and the Ticket Granting Server.

Every Key Distribution Center contains the database of all the principals contained in the realm with their associated secrets.

3.3.6. The Authentication Server (AS) [8]

The authentication Server issues an encrypted Ticket Granting Ticket to clients who wants to login to the Kerberos realm. Instead of client, the TGT which is encrypted with user's password will prove the identity of client to the KDC. As, user and the KDC know the user's password. The TGT returned by the Authentication Server can be used to request service tickets.

3.3.7. The Ticket Granting Server (TGS) [8]

Ticket Granting Server issues service tickets to clients. The TGS server takes two piece of data from clients, a ticket request which has principal name of service the client wants to contact and a Ticket Granting Ticket which has issued by Authentication Server (AS). After verification of TGT the client get service ticket.

3.3.8. Tickets [8]

Tickets confirm the identity of the two principals. One principal being a user and the other a service requested by the user. Tickets establish an encryption key used for secure communication during the authenticated session.

3.3.9. Keytab Files [8]

These are the files extracted from the KDC principal database and contain the encryption key for a service or host.

3.4. Kerberos Authentication Dialogue [9]

The actions performed by Kerberos are as follows:

Step 1 (KRB_AS_REQ): Client/User logs on to workstation and requests service from resource server. The workstation on the behalf of user sends message to the Authorization Server requesting TGT.

Step 2 (KRB_AS_REP): AS verifies users access right in its database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password and sends message back to user.

Step 3 (KRB_TGS_REQ): Workstation prompts user/client for password and uses password to decrypt incoming message, then sends ticket and authenticator that contain client's name, network address and time to TGS.

Step 4 (KRB_TGS_REP): TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.

Step 5 (KRB_AP_REQ): Workstation sends ticket and authenticator to server.

Step 6 (KRB_AP_REP): Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.

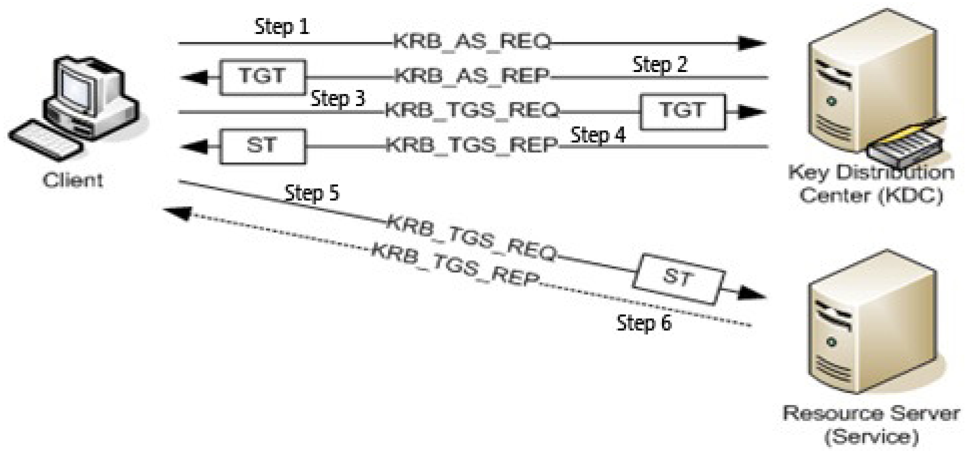


Fig. 7 : Kerberos Authentication Process [10]

3.5. Why Kerberos ?

Today's network applications wants both sides of a connection to be authenticated to prevent phishing and other malicious attacks. Kerberos makes mutual authentication simple. Also, Kerberos is symmetric, any two parties which can authenticate in one way can also authenticate in other direction. Also, Kerberos meets the modern distributed systems requirements. The Kerberos is architecturally sound which makes it easy to integrate into other systems. In today's scenario Kerberos is widely integrated in most popular operating systems and software applications. Now it is an integral part of IT infrastructure.[11]

3.6. Replication [7]

Replication ensures that the accounts database is spread over a number of servers so as to safeguard from node capture and compromise. It also ensures that all user accounts are kept up to date by reflecting any changes to all the servers. This mechanism ensures that all accounts that have been added, modified or revoked since the last replication are updated in the repository in a timely and orderly manner.

3.7. Kerberos Master Slave Replication [12]

Kerberos was designed to allow for a Master/Slave replication cluster. A master is primary server, while there might be one or more slave which acts as the backup for master. The master and slave servers can be thought as Primary and Secondary servers respectively.

The master Kerberos machine maintains the master copy of the authentication database. The information of master Kerberos is stored in application databases which consists of account and data policy data. The slave Kerberos machines have read-only copies of the database elsewhere in the system.

As an extra copy of authentication database is there in slave machine ,so if the master machine is not available, the authentication can still be done by the slave machines. The process to perform authentication on any one of several machines reduces the chances of a bottleneck on the master machine.

The advantages of having multiple copies of the database are those usually cited for replication are:

- higher availability
- better performance.

3.8. Solution to Data Inconsistency [13]

To achieve the data consistency between master and slave machines, the master database is dumped on regular interval of time like hourly or daily. After this the entire data is sent to slave machines which in turn update their own data. The program known as kprop on master machine
send the updated data to a peer program on slave known as kpropd. First master machines sends a checksum of dumped data which encrypted by Kerberos master database key. The master database key is known to both master and slave machine. The propagated data is cached by kpropd on slave machine. The slave machine calculates the checksum of the data received and if it is matching with checksum sent by master, then the new information is used to update slave's database.

CHAPTER 4 : JAAS

4.1. Introduction to JAAS [14]

JAAS is short name for Java Authentication and Authorization Service. JAAS was introduced into J2SDK 1.4.

JAAS is used for two purposes
For Authentication, where users are authenticated to determine who is executing current Java code, irrespective of whether the code is running as application, an applet or a servlet doesn't matter.

For Authorization, where users are authorized to ensure for access control rights for doing particular actions.

Authentication using JAAS is carried out in pluggable fashion. Due to this the applications is independent from underlying authentication technologies. Hence, updated authentication technologies can be plugged into application without requiring alteration to application.

4.2. JAAS High-Level Architecture

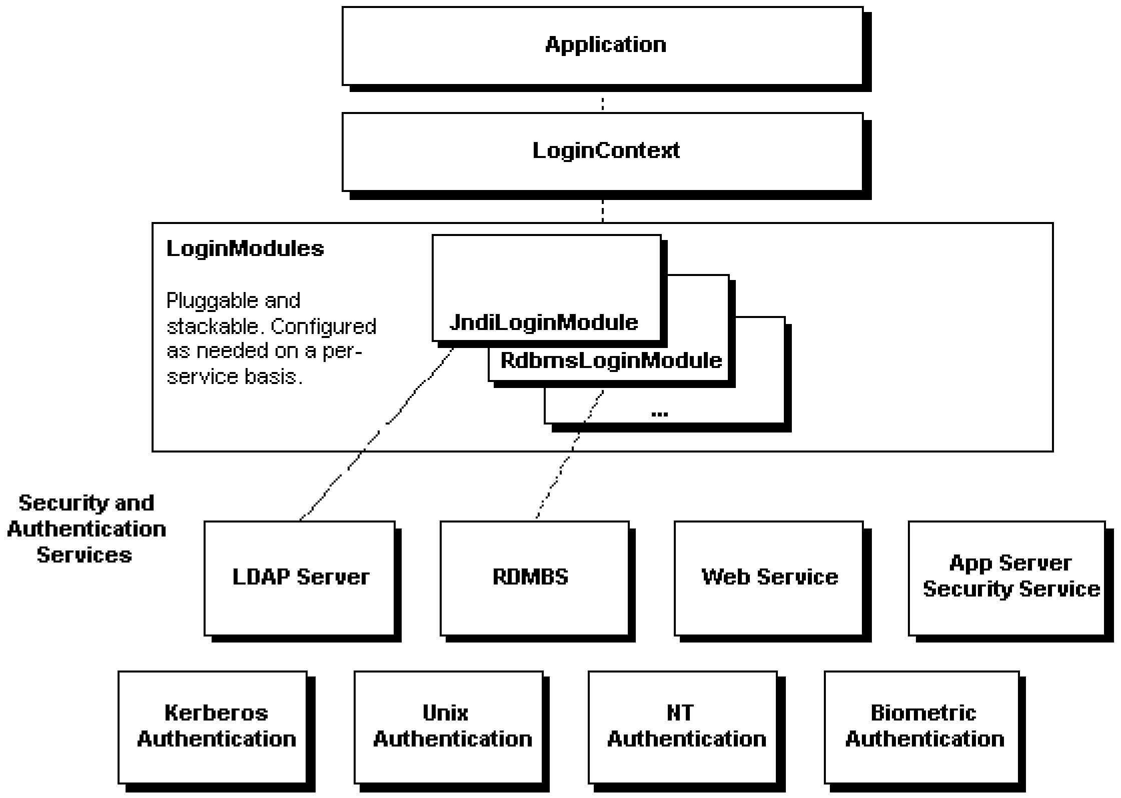


Fig. 8 : JAAS High-Level Architecture [15]

The above Fig.8 shows the high-level overview of how JAAS achieves the pluggability. The application which wants to embed JAAS Authentication firstly deals with LoginContext. Under the bottom of LoginContext there are various dynamically configured LoginModules, which are actually responsible for actual authentication. Also JAAS is stackable, means in single login context, a bunch of security modules can be stack on top of another, while each called in order and each one is dealing with a different security infrastructure.

To use JAAS authentication in the given application has following basic steps:

1. Create a LoginContext
2. Optionally pass the CallbackHandler to LoginContext, for gathering or processing authentication data
3. Perform authentication by calling LoginContext's login() method
4. Perform privileged actions using retured Subject (if login successds)

For example,

```
LoginContext lc = new LoginContext("MyExample");
try {
   lc.login();
} catch (LoginException) {
   // Authentication failed.
}
// Authentication successful, we can now continue.
// We can use the returned Subject if we like.
Subject sub = lc.getSubject();
Subject.doAs(sub, new MyPrivilegedAction());
```

4.3. JASS Classes and Interfaces [15]

The fundamental JAAS classes and interfaces used in the process are typically divided into three groups:

Group Name	**Details**
Common	Subject, Principal, credential
Authentication	LoginContext, LoginModule, CallbackHandler, Callback
Authorization	Policy, AuthPermission, PrivateCredentialPermission

Table 2 : JAAS's Classes and Interfaces [15]

These classes and interfaces are in the javax.security.auth package's sub-packages, while some prebuilt implementations in the com.sun.security.auth package included in J2SE 1.4.

CHAPTER 5 : JAVA RMI

5.1. Introduction to Java RMI [16]

RMI stands for Remote Method Invocation. Remote Method Invocation is well supported by Java. RMI give capability to client to access the service of remote server as if these services are invoked on local objects. The main idea behind RMI is to give server one or more Remote Objects. Methods are the representation of the operations that take place in the server instead of clients. While the Remote Interface are the methods of Remote Object.

5.2. RMI Concepts [16]

5.2.1. Remote Interface

It is a interface which extends java.rmi.Remote. A remote interface gives the list of available methods on remote object. Also, all methods which are in remote interface are declared with throws java.rmi.RemoteException.

5.2.2. Remote Object

Remote object provides its public methods to remote clients across network. A remote object extends java.rmi.server.UnicastRemoteObject and implements a remote interface.

5.2.3. Remote Methods

These are the methods which are listed in remote interface and implemented by remote object. All parameters and result types in remote methods must be serializable.

5.2.4. Serializable Object

An object which is serializable must implement the interface java.io.Serializable. Objects representing resources or things in the operating systems cannot be serialized. While, Objects representing data can be serialized.

5.2.5. Remote Exception

It is an exception which shows the communication problem between client and the server. The typical remote exceptions are related to broken network connection or crashed server.

5.2.6. Remote Stub

It is an object which acts on the behalf of the remote object in client JVM. It implements the same remote interface as the remote object which it represents. The remote stub receive method calls from the client and pass the calls on the remote object.

5.2.7. RMI Registry

It is the naming service which maintains the track of remote objects in server and its service endpoints. For the purpose of client lookup operations the rmiregistry associates each remote object with a name.

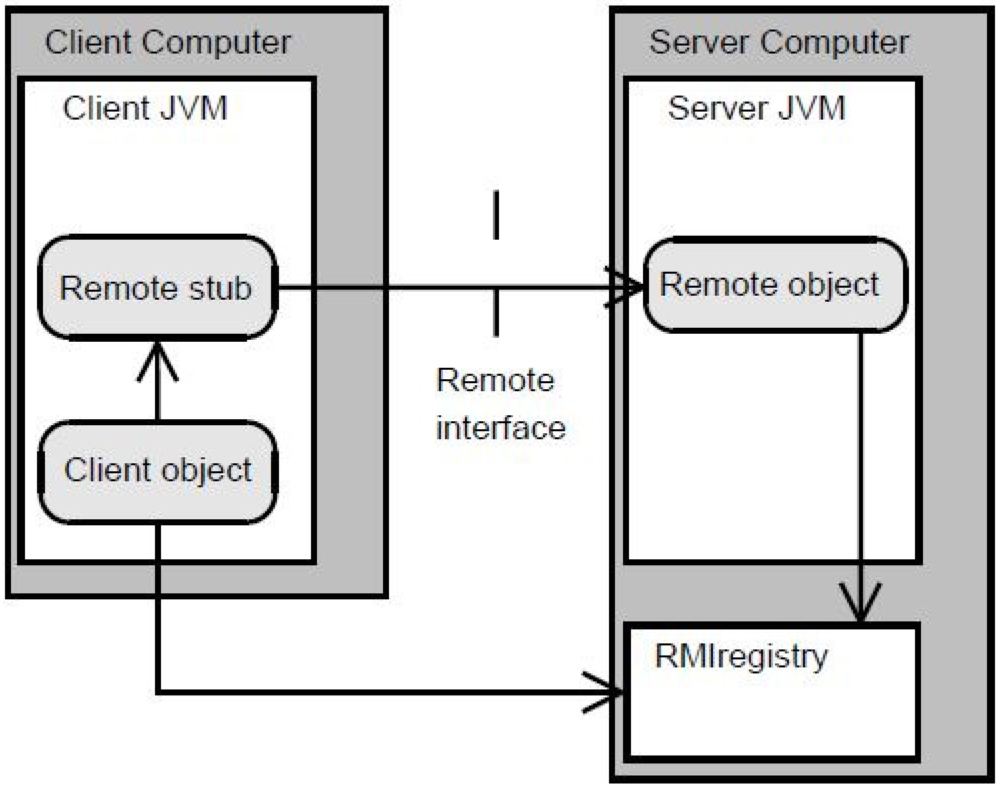


Fig. 9 : Basics of RMI [16]

CHAPTER 6 : DESIGN ARCHITECTURE

In this chapter, the proposed design architecture and the basic flow associated with each architectural component is explained in depth.

6.1. Proposed Architecture

The proposed architecture is based on four main entities :

[1] Kerberos Servers
   a. Kerberos Master
   b. Kerberos Slave

[2] Intermediate Server

[3] Service Server

[4] Clients

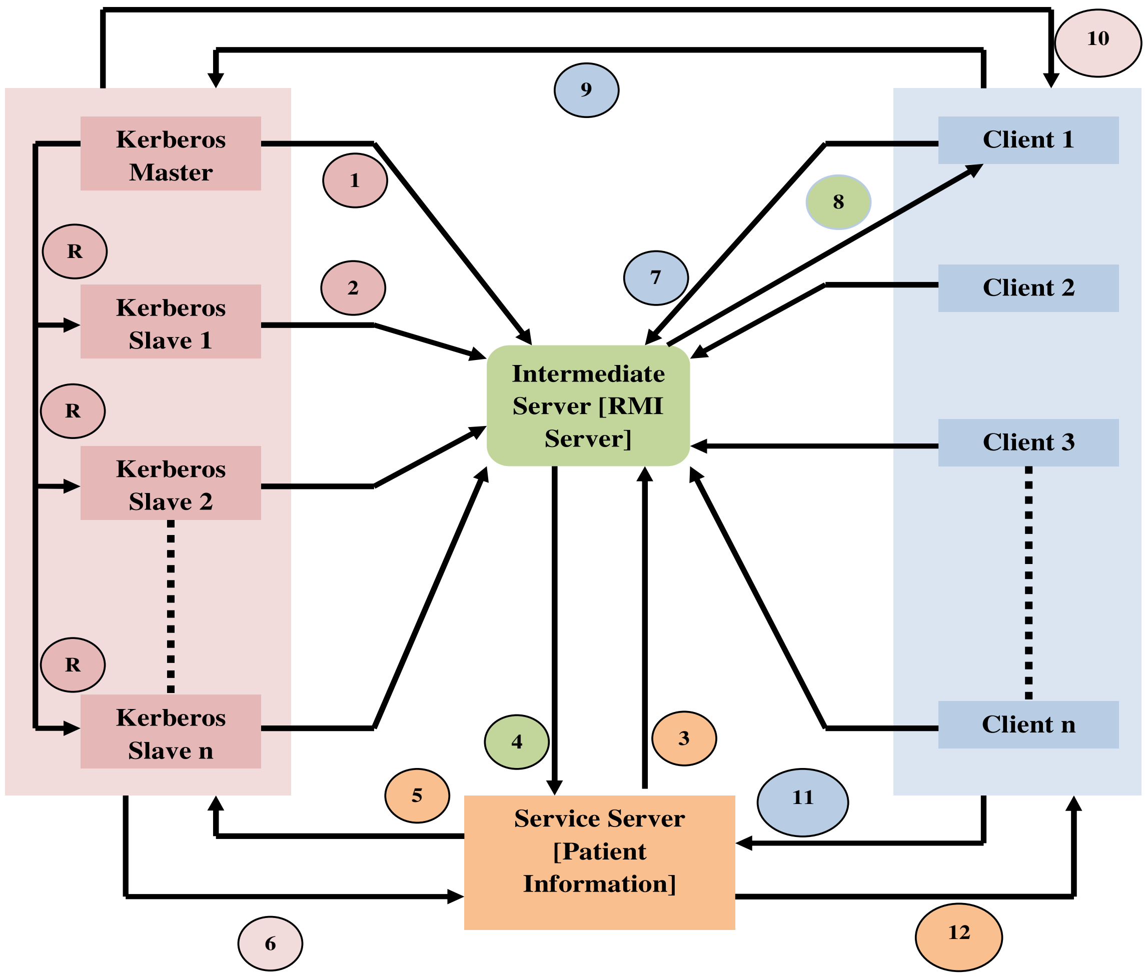


Fig. 10 : Proposed Architecture

The Sequential steps of working of the proposed architecture are:

1. Register the Kerberos Master server to Intermediate Server.
2. Register the Kerberos Slave server to Intermediate Server.
3. Service Server request Scheduled Kerberos server to get authenticated.
4. Service Server get Scheduled Kerberos server domain name/IP from Intermediate Server
5. Service Server sent Kerberos authentication request on scheduled Kerberos Server
6. Service Server get authenticated by scheduled Kerberos Server
7. Client request Scheduled Kerberos server to get authenticated.
8. Client get Scheduled Kerberos server domain name/IP from Intermediate Server
9. Client sent Kerberos authentication request on scheduled Kerberos Server
10. Client get authenticated by scheduled Kerberos Server
11. Client sent request to Service Server to get its service
12. Service accept the client request and give access to its services, if its authenticated client

R. Replication of Master KDC's Database to Slave KDC.

6.2. Kerberized Client and Application Server

MIT Kerberos V5 will be used on backend to provide security to our network. Kerberized Client and Kerberized Application(Service) Server will use the third party authentication service provided by Kerberos Server. The basic flow using Kerberos's KDC, where Doctor will be the Kerberized Client and Service Server will the Kerberized Application Server.

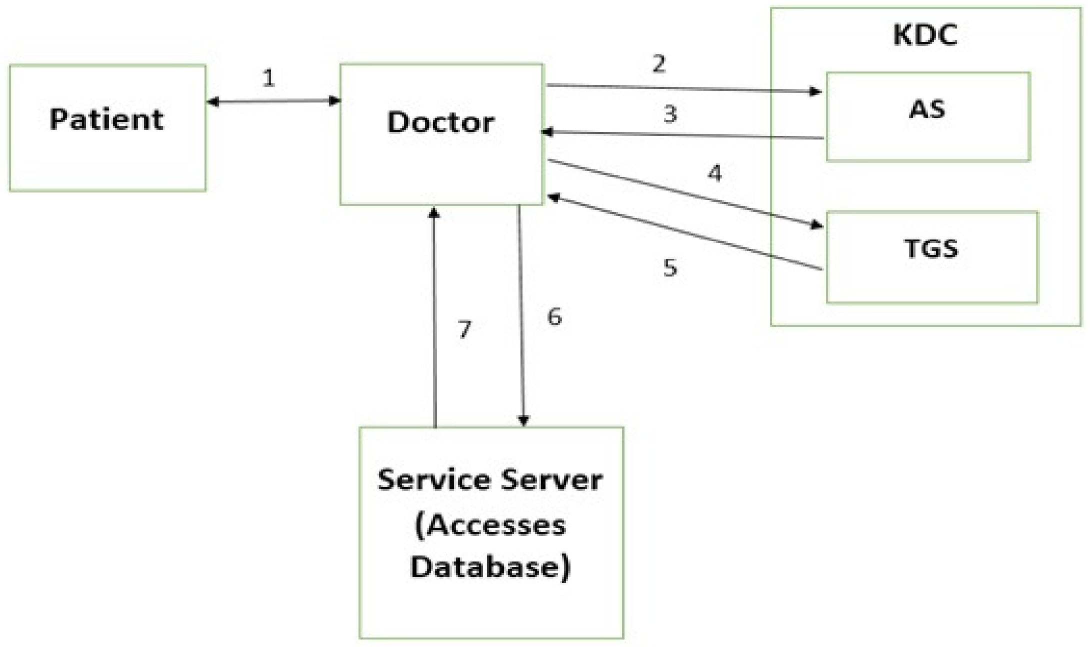


Fig. 11 : Basic Flow between Kerberos KDC, Doctor and Service Server

1. Patient taps his/her card on the device for patient doctor interaction.
2. Doctor asks for TGT (Ticket Granting Ticket) from Authentication Server (AS).
3. AS provides the TGT to the Doctor, after authentication.
4. Doctor sends the TGT acquired from AS to the Ticket Granting Server (TGS) and asks for the ticket to access the service server.
5. TGS sends the Ticket Granting Service (TGS) in response only after validating TGT.
6. The TGS acquired is sent to the Service Server (already registered with KDC) to ask for its' services.
7. Service server validates the TGS and provides the access to its services, if validated.

6.3. Kerberized Client and Intermediate Server

Now an Intermediary server was introduced basically for load balancing and to check if any KDC goes down. The basic flow of the intermediary server is as follows:

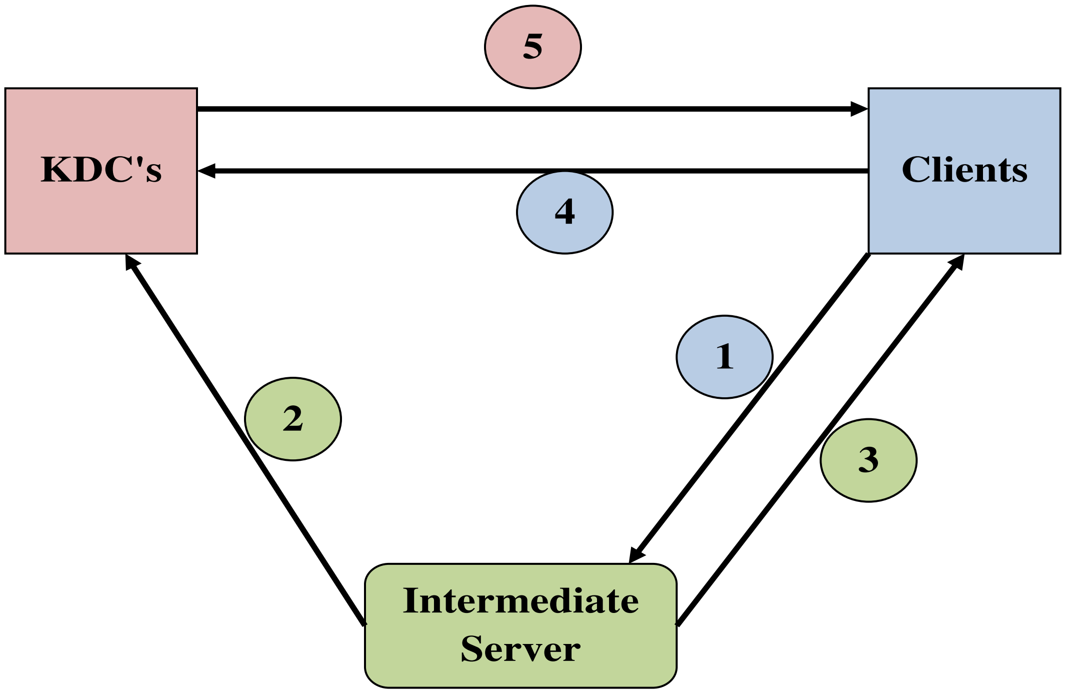


Fig. 12 : Basic Flow of Intermediate Server

1. Clients make a request for IP address of KDC.
2. Intermediary Server checks for the KDC if it is up.
3. Intermediary Server, taking care of load balancing provides the IP address of the KDC to the client.
4. The client then makes request using the given IP address to the KDC for tickets.
5. The KDC responds back and the basic flow given above for KDC is carried forward.

The main task of Intermediary Server will be Load balancing (make sure that all the requests are not allocated to a single KDC ,i.e. Resource distribution) and to keep a check on whether there exists at least one Master machine in the whole Kerberos System which can add users to the database. Also it was to keep a check on Master KDC, if it goes down another KDC will be made the master by Intermediate Server.

6.4. Kerberos Master-Slave Replication

For Kerberos Database to be consistent on each KDC, replication of Master KDC's database to Slave KDC is important. This replication of Kerberos Database is done on hourly basis. The Intermediate Server will maintain the list of KDC's in the Authentication System. Whereas Client will connect to Intermediate Server and get the Scheduled KDC Name/IP for authentication.

The below Fig. 11 shows the Kerberos Master-Slave Replication and interaction between Client, Intermediate Servers and KDC's.

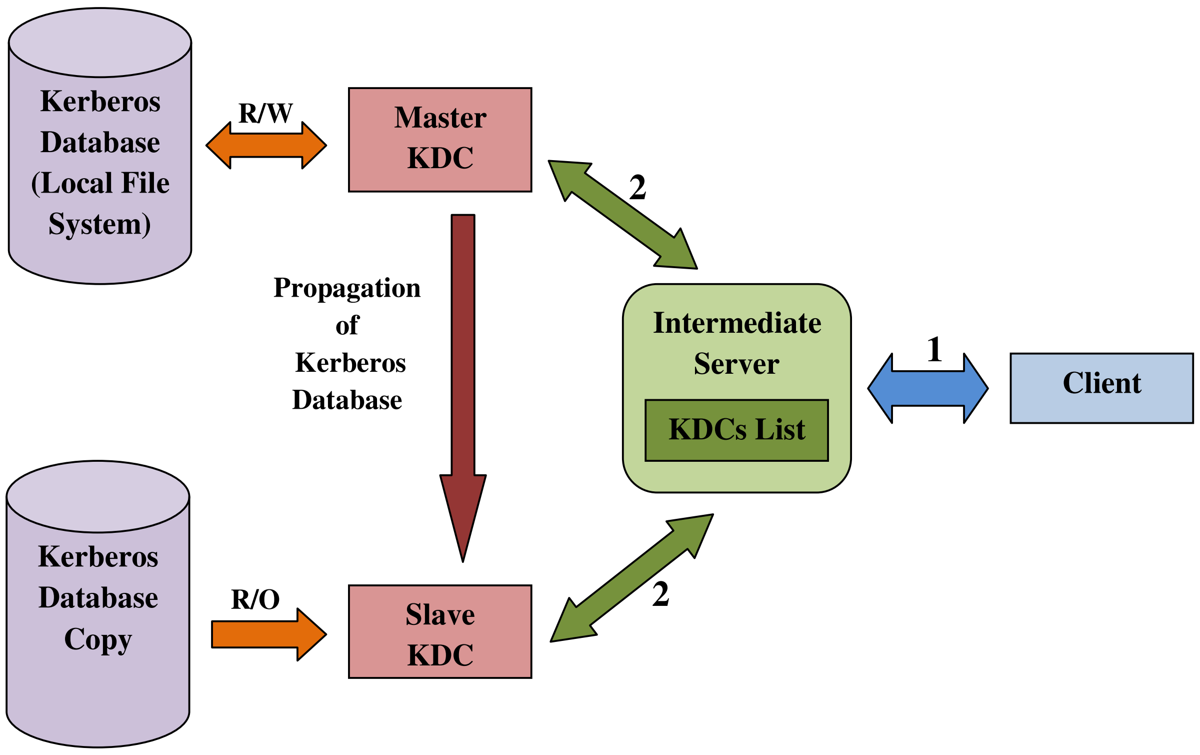


Fig. 13 : Kerberos Master-Slave Replication

1.Client and Intermediate Server Interface
2. Intermediate Server and KDC's Interface
R/W = Read and Write
R/O = Read Only

CHAPTER 7 : IMPLEMENTATION

This section explains the implementation of Kerberos Server Setup with Master-Slave replication, Intermediate Server based on Java RMI Server, Service Server and Clients based on RMI clients with JAAS and GSS-API.

7.1. Kerberos Setup

MIT Kerberos is implemented on Ubuntu Servers 10.04 and 100 principals were created and tested for master-slaves.

Initially, a KDC, i.e. A Key Distribution Server of Kerberos was installed on Ubuntu, using the Ubuntu Server Guide. Network Configurations were done so as to make the KDC working. The ticket generation was tested by calling and viewing a generated ticket for a specific principal. The same was done and tested for 100 principals.

7.2. Master Slave Replication[17]

7.2.1. Methodology used to Install Master Kerberos:

1. The first step in installing a Kerberos Realm is to install the krb5-kdc and krb5-admin-server

packages. From a terminal enter:

```
sudo apt-get install krb5-kdc krb5-admin-server
```

2. Next, create the new realm with the kdb5_newrealm utility

```
sudo krb5_newrealm
```

The user is asked to create a new realm which in our case is **LANSLAB.EDU**

3. Now that the KDC running an admin user is needed. It is recommended to use a different

username from your everyday username. Using the kadmin.local utility in a terminal prompt

enter:

```
sudo kadmin.local
Authenticating as principal root/admin@LANSLAB.EDU with password.
kadmin.local: addprinc princ1/admin
```

```
WARNING: no policy specified for harsh/admin@LANSLAB.EDU; defaulting to no policy
Enter password for principal "princ1/admin@LANSLAB.EDU":
Re-enter password for principal "princ1/admin@LANSLAB.EDU":
Principal "princ1/admin@LANSLAB.EDU" created.
kadmin.local: quit
```

4. Next, the new admin user needs to have the appropriate Access Control List (ACL) permissions.

The permissions are configured in the /etc/krb5kdc/kadm5.acl file:

```
princ1/admin@LANSLAB.EDU                        *
```

This entry grants princ1/admin the ability to perform any operation on all principals in the realm.

5. Now restart the krb5-admin-server for the new ACL to take affect:

```
sudo /etc/init.d/krb5-admin-server restart
```

6. The new user principal can be tested using the kinit utility:

```
kinit -p princ1/admin@LANSLAB.EDU
harsh/admin@LANSLAB.EDU's Password:
```

7. After entering the password, use the klist utility to view information about the Ticket Granting Ticket (TGT) :

```
klist
Credentials cache: FILE:/tmp/krb5cc_1000
Principal: harsh/admin@LANSLAB.EDU
Issued Expires Principal
Jan 13 14:53:34 Jan 14 14:53:34 krbtgt/LANSLAB.EDU@LANSLAB.EDU
```

8. An entry is added into the /etc/hosts section:

```
172.16.6.28          kdc01.lanslab.edu          kdc01
```

After the successful setup of Kerberos the next task was Master Slave Replication.

9. Configuring the master KDC

In our setup the following names were used:-

kmaster.lanslab.edu - master KDC
kslave.lanslab.edu ,kslave2.lanslab.edu ,kslave3.lanslab.edu - slave KDC's
.k5.LANSLAB.EDU - stash file
admin/admin - admin principal

10. Edit KDC configuration files

the configuration files, *krb5.conf* and *kdc.conf* were modified to reflect the correct information (such as domain-realm mappings and Kerberos servers names) for our realm
An example krb5.conf file:

```
[libdefaults]
  default_realm =LANSLAB.EDU

[realms]
  LANSLAB.EDU = {
     kdc = kmaster.lanslab.edu
     kdc = kslave.lanslab.edu
     kdc = kslave2.lanslab.edu
     kdc = kslave3.lanslab.edu
     admin_server = kmaster.lanslab.edu
}
```

11. Create the KDC database

```
kdb5_util create -r LANSLAB.EDU -s

Initializing database '/usr/local/var/krb5kdc/principal' for realm 'LANSLAB.EDU',
master key name 'K/M@LANSLAB.EDU'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key:  <= Type the master password.
Re-enter KDC database master key to verify:  <= Type it again.
```

12. Add administrators to the ACL file

Next we created an Access Control List (ACL) file and put the Kerberos principal of at least one of the administrators into it. This file is used by the *kadmind* daemon to control which principals may view and make privileged modifications to the Kerberos database files
An example kadm5.acl file

```
*/admin@LANSLAB.EDU       *
```

13. Start the Kerberos daemons on the Master KDC

```
service start krb5-kdc
```

after the setup of primary kdc the next task was to install slave KDC's

7.2.2. Install The Slave KDCs [18]

1. Create host keytabs for slave KDCs

Each KDC needs a host key in the Kerberos database. These keys are used for mutual authentication when propagating the database dump file from the master KDC to the secondary KDC servers.

On the master KDC, connect to administrative interface and create the host principal for each of the KDCs' host services. The commands are:-

```
kadmin
kadmin: addprinc -randkey host/kmaster.lanslab.edu
NOTICE: no policy specified for "host/kmaster.lanslab.edu@LANSLAB.EDU"; assigning
"default"
Principal "host/kmaster.lanslab.edu@LANSLAB.EDU" created.

kadmin: addprinc -randkey host/kslave.lanslab.edu
NOTICE: no policy specified for "host/kslave.lanslab.edu@ATHENA.MIT.EDU"; assigning
"default"
Principal "host/kslave.lanslab.edu@LANSLAB.EDU" created.
```

Next, extract host random keys for all participating KDCs and store them in each host's default keytab file. Ideally, you should extract each keytab locally on its own KDC. If this is not feasible, you should use an encrypted session to send them across the network. To extract a keytab on a slave KDC called kslave.lanslab.edu, you would execute the following command:

```
kadmin: ktadd host/kslave.lanslab.edu
Entry for principal host/kslave.lanslab.edu, with kvno 2, encryption
  type aes256-cts-hmac-sha1-96 added to keytab FILE:/etc/krb5.keytab.
Entry for principal host/kslave.lanslab.edu, with kvno 2, encryption
  type aes128-cts-hmac-sha1-96 added to keytab FILE:/etc/krb5.keytab.
Entry for principal host/kslave.lanslab.edu, with kvno 2, encryption
  type des3-cbc-sha1 added to keytab FILE:/etc/krb5.keytab.
Entry for principal host/kslave.lanslab.edu, with kvno 2, encryption
  type arcfour-hmac added to keytab FILE:/etc/krb5.keytab.
```

2. Configure slave KDCs

Database propagation copies the contents of the master's database, but does not propagate configuration files, stash files, or the kadm5 ACL file. The following files were copied to each slave

- krb5.conf
- kdc.conf
- kadm5.acl
- master key stash file

3. Propagation ACL

Create a file named kpropd.acl in the KDC state directory containing the host principals for each of the KDCs:

```
host/kmaster.lanslab.edu@LANSLAB.EDU
host/kslave.lanslab.edu@LANSLAB.EDU
host/kslave2.lanslab.edu@LANSLAB.EDU
host/kslave3.lanslab.edu@LANSLAB.EDU
```

Then, add the following line to /etc/inetd.conf on each KDCkrb5_prop stream tcp nowait root /usr/local/sbin/kpropd kpropd

You also need to add the following line to /etc/services on each KDC, if it is not already present (assuming that the default port is used):

```
krb5_prop       754/tcp             # Kerberos slave propagation
```

Restart inetd daemon.

4. Database propagation

The next step is to propagate the database from the master kdc to the slave kdc's

From a terminal on the *kmaster*, create a dump file of the principal database:

```
sudo kdb5_util dump /var/lib/krb5kdc/dump
```

Using the **kprop** utility push the database to the Secondary KDC:

```
sudo kprop -r LANSLAB.EDU -f /var/lib/krb5kdc/dump kslave.lanslab.edu
```

If the propagation is Successful, then Succeeded message will seen on terminal.

Propagation script

The master KDC, kmaster.lanslab.edu, must regularly push its database out to the slaves to maintain synchronization. One way to do this is to create a script on kmaster, called /etc/cron.hourly/krb5-prop, to regularly perform the database dump and propagation tasks.
For pseudo code of Propagation script refer Appendix 1.

This completes the setup of Master and Slave replication process.

The next task was to avoid the situation where the master goes down, i.e. the application database goes down. In such a situation we want one of the slaves to work as a master to provide the application privileges.

Role Reversal

In the following instructions,kdc2.example.com will become the master and kdc1 its slave.
First, on kmaster, stop the Kerberos administration server process (kadmin):

```
/etc/init.d/krb5-admin-server stop
```

Then, still on kmaster, disable the propagation script and run it once more manually:

```
mv /etc/cron.hourly/krb5-prop/krb5-prop
```

Now, over on kslave, install the Kerberos administration server:

```
 apt-get install krb5-admin-server
```

Also, create a propagation script, /etc/cron.hourly/krb5-prop, just like the one above in step 8, except that the slave mentioned in it should be kmaster instead of kslave.
Next, still on kslave, create a new file on kslave, called /etc/krb5kdc/kadm5.acl, with the following contents:

```
*/admin *
admin *
```

This same file should be deleted from the previous master server, kmaster
Finally edit the admin-server name in the krb5.conf file and change it to refer to kslave instead of kmaster
A new cron job script is created to propagate the database from newly created master to the rest of the slaves.

7.3. Kerberized Client and Application Server [14][19]

We developed two programs (kerberized client and application server) in Java using JAAS and GSS-API:

Using JAAS authentication from the application typically involves the following steps:

1. Create a LoginContext
2. Optionally pass a CallbackHandler to the LoginContext, for gathering or processing authentication data
3. Perform authentication by calling the LoginContext's login() method
4. Perform privileged actions using the returned Subject (assuming login succeeds)

Underneath the covers, a few more things occur:

1. During initialization, the LoginContext finds the configuration entry "MyExample"in a JAAS configuration file (which we configured) to determine which LoginModules to load.
2. During login, the LoginContext calls each LoginModule's login() method
3. Each login() method performs the authentication or enlists a CallbackHandler
4. The CallbackHandler uses one or more Callbacks to interact with the user and gather input
5. A new Subject instance is populated with authentication details such as Principals and credentials.

This all is done just to login to the KDC i.e. to get TGT in the subject.

Now that we have got TGT, the TGT authenticator is created and used to ask for TGS then. The passing of TGS and the communication between client and application server takes place with the help of GSS-API.[23][24]

Broadly speaking, the GSS-API does two main things:

1. It creates a security context in which data can be passed between applications. A context can be thought of as a sort of “state of trust” between two applications. Applications that share a context know who each other are and thus can permit data transfers between them as long as the context lasts.
2. It applies one or more types of protection, known as security services, to the data to be transmitted. Security services are explained in Security Services.

The GSS-API describes many procedure calls. Significant ones which we used in our code are:

- GSS_Init_sec_context - generates a client token to send to the server, usually a challenge
- GSS_Accept_sec_context - processes a token from GSS_Init_sec_context and can generate a response token to return
- GSS_Wrap - converts application data into a secure message token (typically encrypted)
- GSS_Unwrap - converts a secure message token back into application data

These are the basic steps in using the GSS-API:

1. Each application, client and server, acquires credentials explicitly.
2. The client initiates a security context and the server accepts it.
3. The client applies security protection to the message (data) it wants to transmit. This means that it either encrypts the message or stamps it with an identification tag. The sender transmits the protected message.
4. The server decrypts the message (if needed) and verifies it (if appropriate).
5. The recipient returns an identification tag to the sender for confirmation for mutual authentication.
6. Both applications then communicate in a secure manner.

A general schema of this process is presented in Fig. 14, which shows one way that the GSS-API can be used.

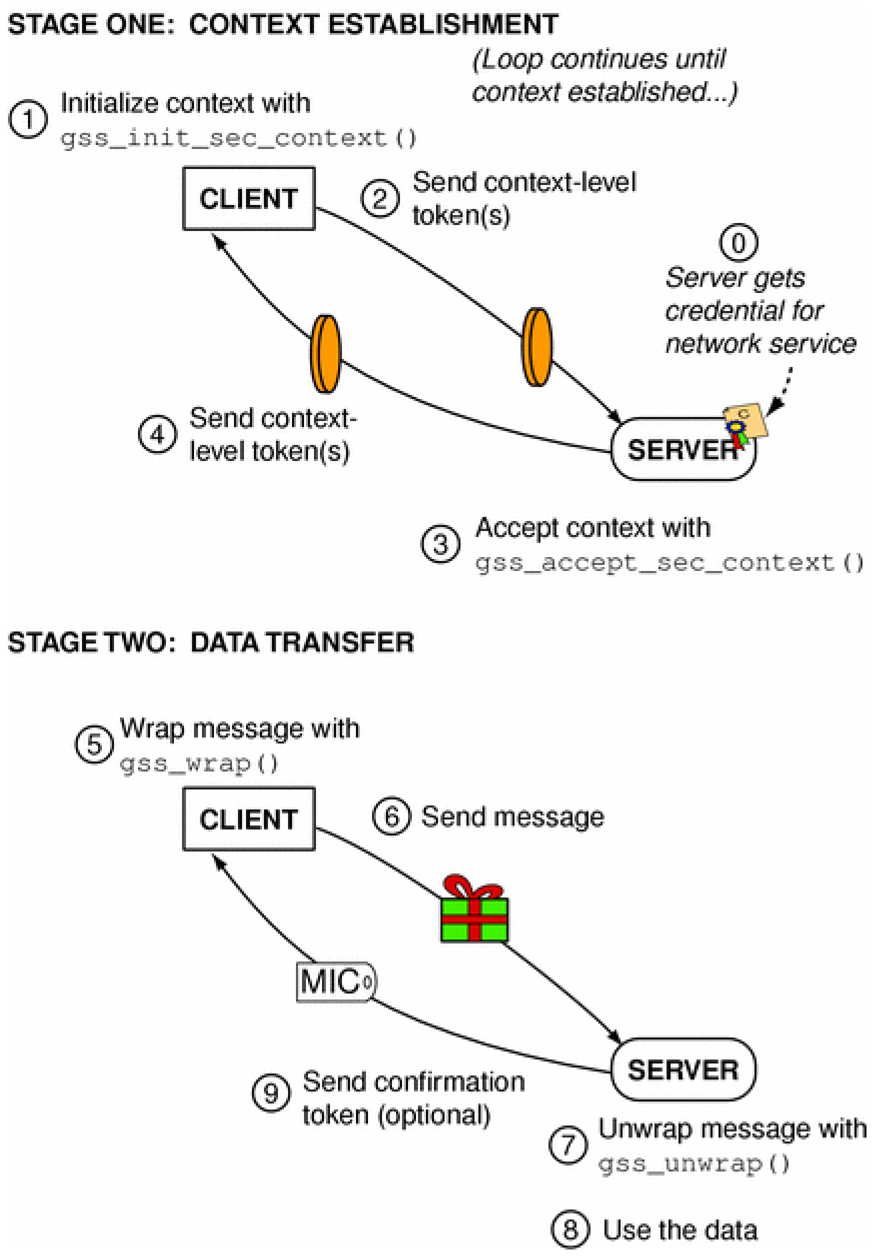

STAGE ONE: CONTEXT ESTABLISHMENT
(Loop continues until context established...)
① Initialize context with gss_init_sec_context()
CLIENT
② Send context-level token(s)
⓪ Server gets credential for network service
④ Send context-level token(s)
SERVER
③ Accept context with gss_accept_sec_context()
STAGE TWO: DATA TRANSFER
⑤ Wrap message with gss_wrap()
CLIENT
⑥ Send message
MIC
SERVER
⑨ Send confirmation token (optional)
⑦ Unwrap message with gss_unwrap()
⑧ Use the data

Fig. 14 : GSS-API Overview [19]

This was all successfully implemented and tested with more than one clients.

7.4. Implementation of Scalable Master Slave Replication

The manner in which the priority is given to the machines depends on the configuration files krb5.conf which consists of all the KDC's

String type IP Arraylist is created which consist of IP addresses of all the KDC's. Intermediary Server keeps on checking whether the Master is still active in every 15 seconds. If the master is active it does not do anything and whenever the Master is down ,it chooses a new master and activate its services (special services are available to only the master machine).

While the new master is active, if the previous master goes up again it sends an on boot message to the intermediary server and intermediary server in return deactivates its services and now forces it to act as a slave.

The client initially retrieves the IP address of the KDC, it shall contact for the tickets from the Intermediary server. The Intermediate server checks for the KDC which is up and does not bear much load, and then gives its IP address to the client. After client gets the IP address of the KDC, it contacts the KDC for the tickets and then sends the IP address of KDC machine to the Intermediate server. The Intermediate server in response releases the resources that were allocated to that KDC machine.

- ❖ The following tasks needs to be performed to start the services on the master KDC
  - service krb5-admin-service start
  - create a crontab file for the database propagation from the master machine to the slave machine.
  - Create kadm5.acl with an admin principal name
  - Edit the krb5.conf file .Change the name of the admin-server to the name of the kdc machine
- ❖ The following tasks needs to be performed to stop the services on the master KDC
  - service krb5-admin-service stop
  - delete the crontab
  - delete the kadm5.acl .
  - Edit the krb5.conf file .Change the name of the admin-server to the name of the kdc machine which is the current master.

7.5. Intermediate Server and Client Implementation

Programmatically, everything is handled using Java RMI Client - Server Processes. The Intermediate Server acts as RMI Server which has 5 main tasks :

a. Maintain KDC List
b. Give Scheduled KDC Name/IP to Clients
c. Implement Some Scheduling Algorithm
d. Maintain Client and Service Server List
e. Check if Master is still up

Every KDC as soon as it is up, is supposed to send a message of request to add in KDC List of the Intermediate server and maintain a connection with it so that clients can be allocated to the respective KDC. While, the Clients are RMI client which first contact the Intermediate Server for scheduled KDC Name/IP, after getting the scheduled KDC Name/IP it will send authentication request on KDC. If the client is authenticated user then it will get authenticated by KDC and gets TGT. Now, client has TGT which is used for requesting service from Application Server (Service Server).

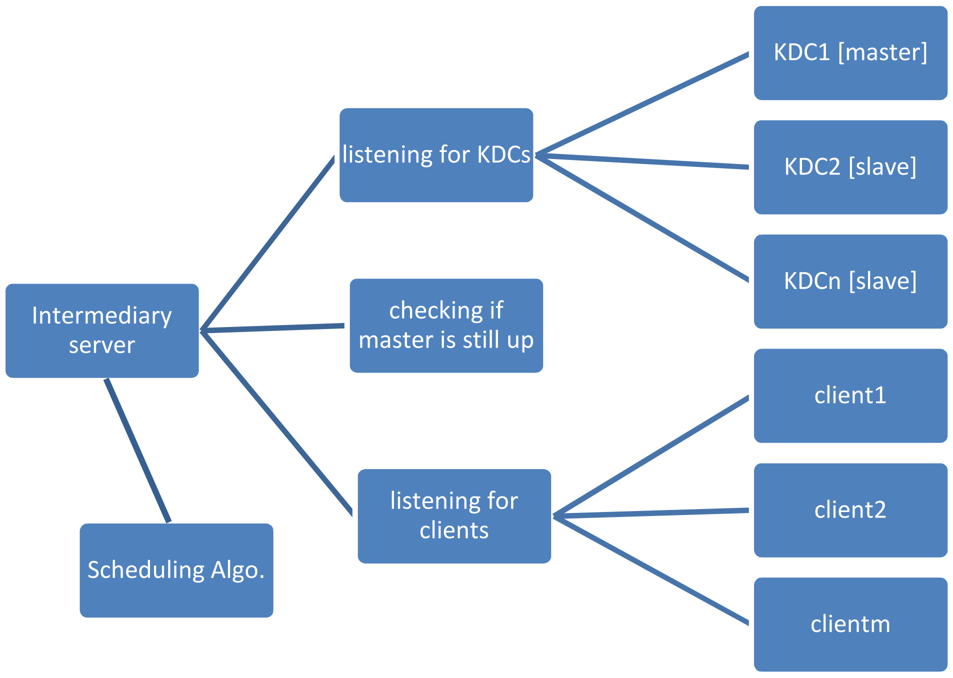


Fig. 15 : Main Tasks of Intermediate Server

Now when resource allocation comes into picture, the problem of critical section follows it. So to handle it we used a flag bit, which will be set if a thread is allocating resource to a client. At that very point no other client will be allocated any resource. As soon as the former client is allocated with the KDC, the flag bit is set to false and now any other client can enter the critical section and again set flag true symbolizing locked critical section. Also the problem of infinite waiting was solved using FCFS approach by maintaining a waiting queue and adding the clients into it as soon as they come and removing the client sequentially.

CHAPTER 8 : TESTING AND RESULTS

The results which come after setting up the Kerberos Master-Slave Servers and implementation of RMI Intermediate Server, Service Server and Clients requesting the services are shown with the help of screenshots, tables and graphs.

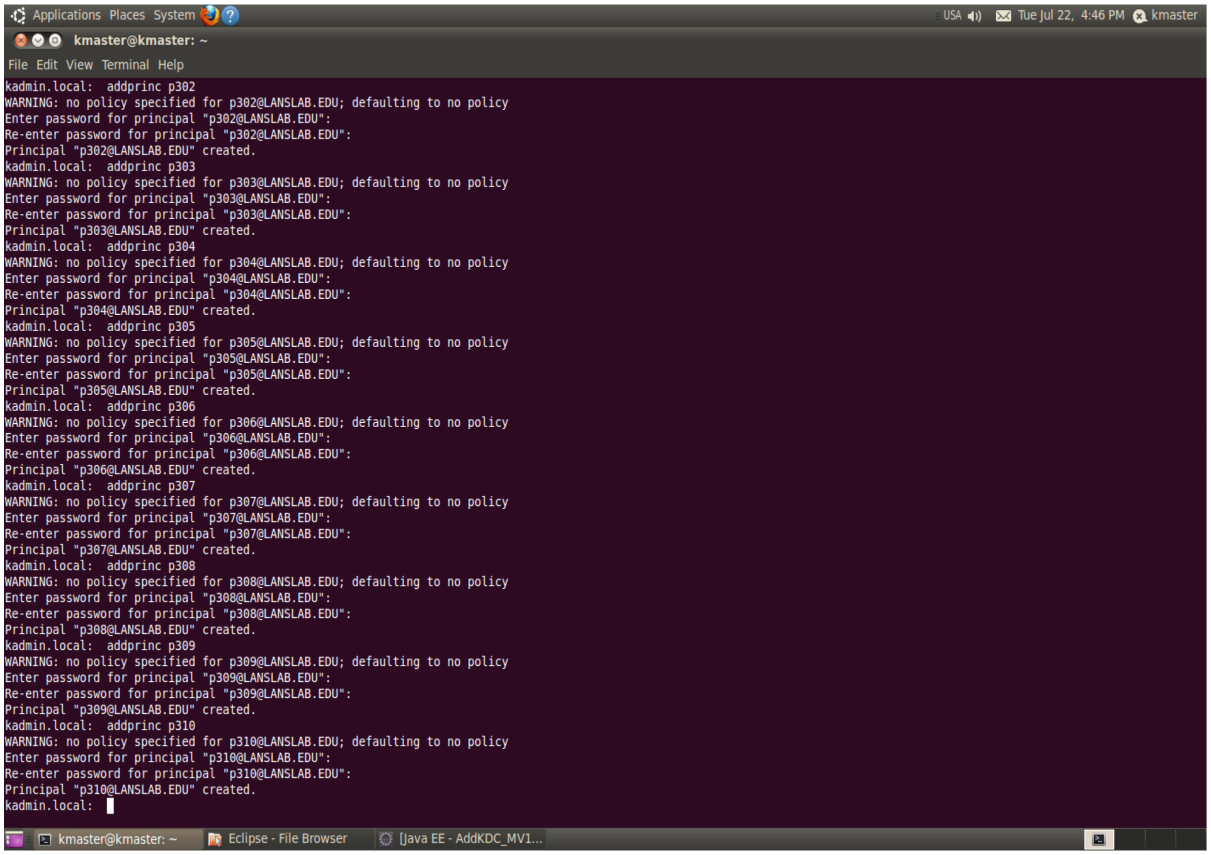
8.1. Test Environment

For testing we have used 3 KDCs i.e. 1 Master KDC and 2 Slave KDCs. The configuration details are listed in below Table 3.

	Kerberos Master	**Kerberos Slave 1**	**Kerberos Slave 2**
OS	Ubuntu Server 10.04	Ubuntu Server 10.04	Ubuntu Server 10.04
Processor	Intel Core 2 Duo	Intel Core 2 Duo	Intel Core 2 Duo
RAM	5.7 GB	1.9 GB	3.7 GB
Hard Disk	250 GB	250 GB	250 GB
Host Name	kmaster.lanslab.edu	kslave.lanslab.edu	kslave2.lanslab.edu
IP Address	172.16.6.24	172.16.6.18	172.16.6.22
Functionality	Intermediate RMI Server, AppServer, Master KDC, Clients	SlaveKDC	Slave KDC
Gateway IP	172.16.1.1	172.16.1.1	172.16.1.1

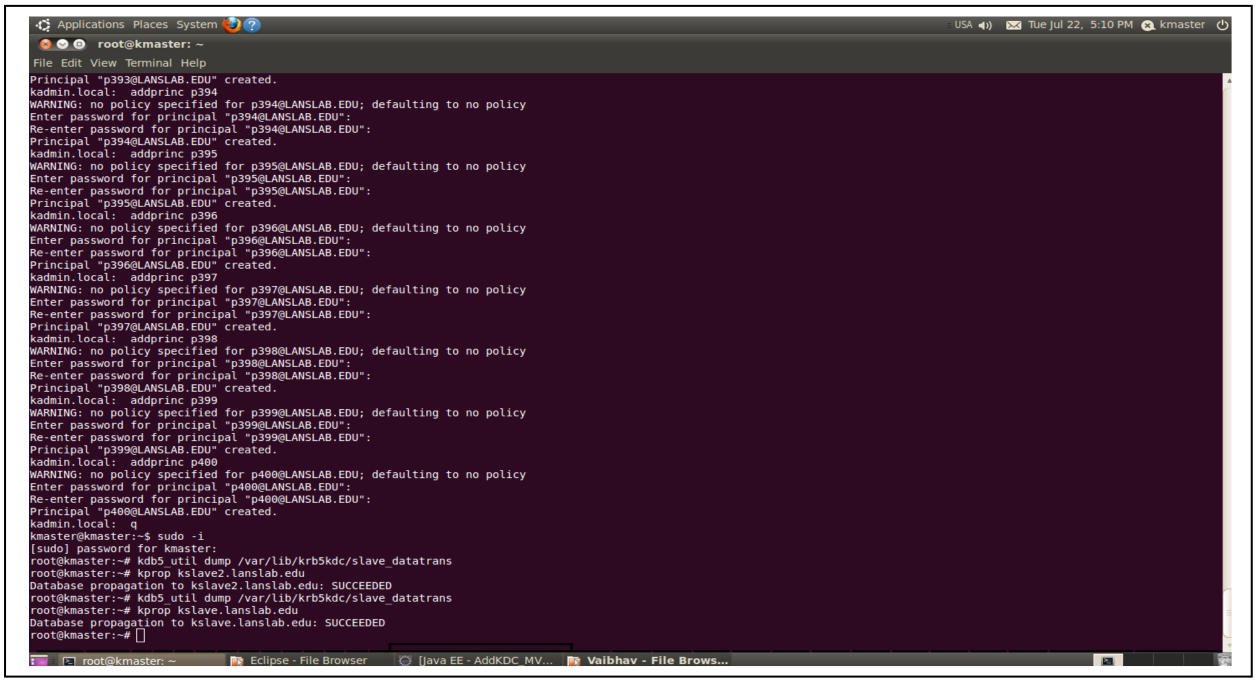
Table 3 : Configuration Details of All KDCs

8.2. Kerberos Replicated Master-Slave Servers


Applications Places System
USA Tue Jul 22, 4:46 PM kmaster
kmaster@kmaster: ~
File Edit View Terminal Help
kmaster@kmaster: ~
Eclipse - File Browser
[Java EE - AddKDC_MV1...

```
kadmin.local:  addprinc p302
WARNING: no policy specified for p302@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p302@LANSLAB.EDU":
Re-enter password for principal "p302@LANSLAB.EDU":
Principal "p302@LANSLAB.EDU" created.
kadmin.local:  addprinc p303
WARNING: no policy specified for p303@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p303@LANSLAB.EDU":
Re-enter password for principal "p303@LANSLAB.EDU":
Principal "p303@LANSLAB.EDU" created.
kadmin.local:  addprinc p304
WARNING: no policy specified for p304@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p304@LANSLAB.EDU":
Re-enter password for principal "p304@LANSLAB.EDU":
Principal "p304@LANSLAB.EDU" created.
kadmin.local:  addprinc p305
WARNING: no policy specified for p305@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p305@LANSLAB.EDU":
Re-enter password for principal "p305@LANSLAB.EDU":
Principal "p305@LANSLAB.EDU" created.
kadmin.local:  addprinc p306
WARNING: no policy specified for p306@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p306@LANSLAB.EDU":
Re-enter password for principal "p306@LANSLAB.EDU":
Principal "p306@LANSLAB.EDU" created.
kadmin.local:  addprinc p307
WARNING: no policy specified for p307@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p307@LANSLAB.EDU":
Re-enter password for principal "p307@LANSLAB.EDU":
Principal "p307@LANSLAB.EDU" created.
kadmin.local:  addprinc p308
WARNING: no policy specified for p308@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p308@LANSLAB.EDU":
Re-enter password for principal "p308@LANSLAB.EDU":
Principal "p308@LANSLAB.EDU" created.
kadmin.local:  addprinc p309
WARNING: no policy specified for p309@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p309@LANSLAB.EDU":
Re-enter password for principal "p309@LANSLAB.EDU":
Principal "p309@LANSLAB.EDU" created.
kadmin.local:  addprinc p310
WARNING: no policy specified for p310@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p310@LANSLAB.EDU":
Re-enter password for principal "p310@LANSLAB.EDU":
Principal "p310@LANSLAB.EDU" created.
kadmin.local:
```

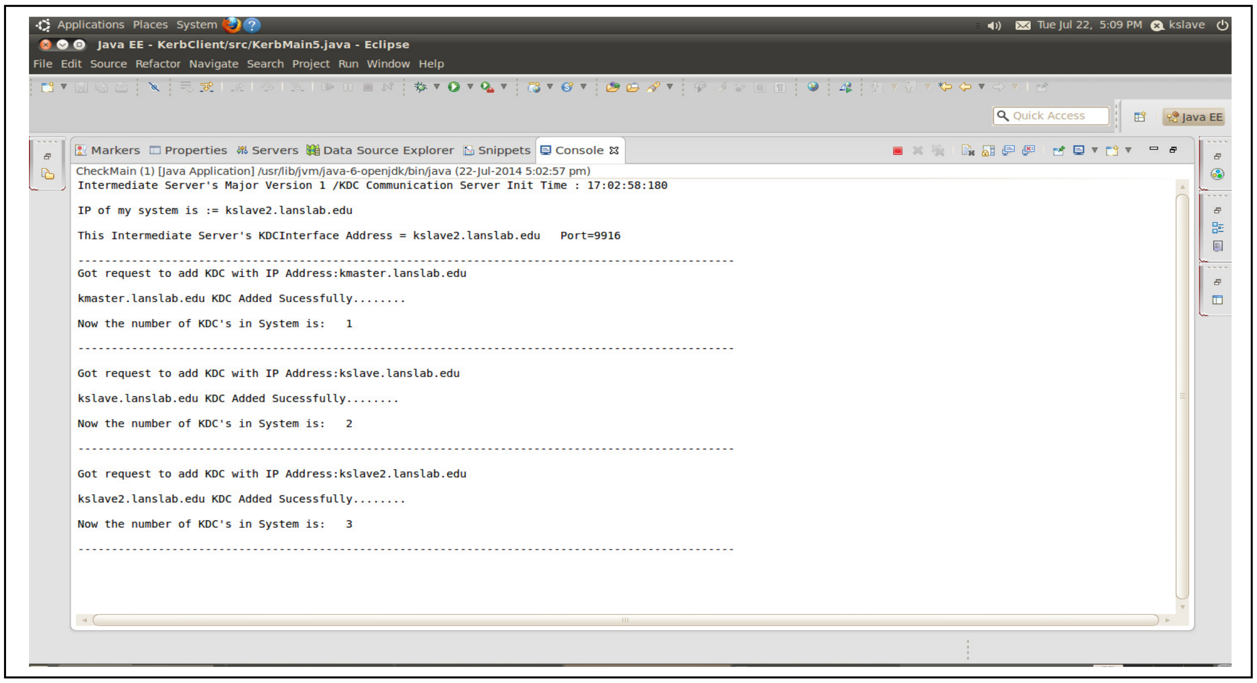
Fig. 16 : Adding Principal into Kerberos Master Database



```
Principal "p393@LANSLAB.EDU" created.
kadmin.local:  addprinc p394
WARNING: no policy specified for p394@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p394@LANSLAB.EDU":
Re-enter password for principal "p394@LANSLAB.EDU":
Principal "p394@LANSLAB.EDU" created.
kadmin.local:  addprinc p395
WARNING: no policy specified for p395@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p395@LANSLAB.EDU":
Re-enter password for principal "p395@LANSLAB.EDU":
Principal "p395@LANSLAB.EDU" created.
kadmin.local:  addprinc p396
WARNING: no policy specified for p396@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p396@LANSLAB.EDU":
Re-enter password for principal "p396@LANSLAB.EDU":
Principal "p396@LANSLAB.EDU" created.
kadmin.local:  addprinc p397
WARNING: no policy specified for p397@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p397@LANSLAB.EDU":
Re-enter password for principal "p397@LANSLAB.EDU":
Principal "p397@LANSLAB.EDU" created.
kadmin.local:  addprinc p398
WARNING: no policy specified for p398@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p398@LANSLAB.EDU":
Re-enter password for principal "p398@LANSLAB.EDU":
Principal "p398@LANSLAB.EDU" created.
kadmin.local:  addprinc p399
WARNING: no policy specified for p399@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p399@LANSLAB.EDU":
Re-enter password for principal "p399@LANSLAB.EDU":
Principal "p399@LANSLAB.EDU" created.
kadmin.local:  addprinc p400
WARNING: no policy specified for p400@LANSLAB.EDU; defaulting to no policy
Enter password for principal "p400@LANSLAB.EDU":
Re-enter password for principal "p400@LANSLAB.EDU":
Principal "p400@LANSLAB.EDU" created.
kadmin.local:  q
kmaster@kmaster:~$ sudo -i
[sudo] password for kmaster:
root@kmaster:~# kdb5_util dump /var/lib/krb5kdc/slave_datatrans
root@kmaster:~# kprop kslave2.lanslab.edu
Database propagation to kslave2.lanslab.edu: SUCCEEDED
root@kmaster:~# kdb5_util dump /var/lib/krb5kdc/slave_datatrans
root@kmaster:~# kprop kslave.lanslab.edu
Database propagation to kslave.lanslab.edu: SUCCEEDED
root@kmaster:~# 
```

Fig. 17 : Kerberos Master Database Propagation to Slave

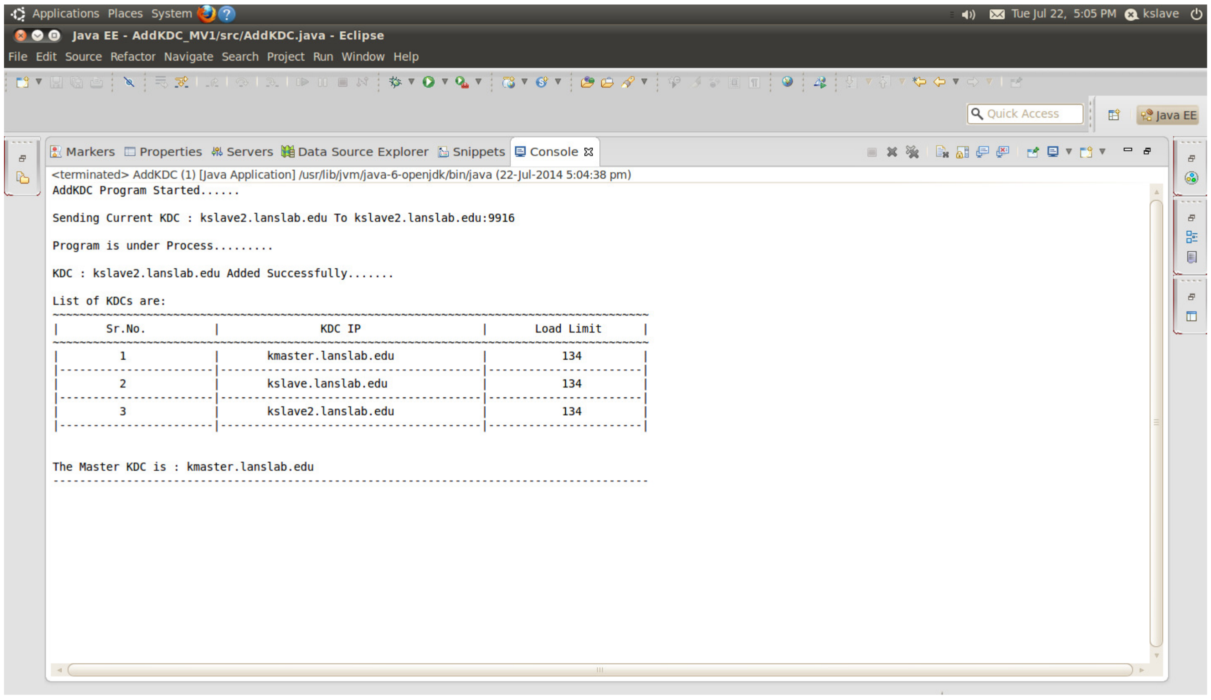
8.3. RMI Intermediate Server


Applications Places System
Tue Jul 22, 5:09 PM kslave
Java EE - KerbClient/src/KerbMain5.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Markers Properties Servers Data Source Explorer Snippets Console

```
CheckMain (1) [Java Application] /usr/lib/jvm/java-6-openjdk/bin/java (22-Jul-2014 5:02:57 pm)
Intermediate Server's Major Version 1 /KDC Communication Server Init Time : 17:02:58:180

IP of my system is := kslave2.lanslab.edu

This Intermediate Server's KDCInterface Address = kslave2.lanslab.edu   Port=9916

-----------------------------------------------------------------------------------------------
Got request to add KDC with IP Address:kmaster.lanslab.edu

kmaster.lanslab.edu KDC Added Sucessfully........

Now the number of KDC's in System is:   1

-----------------------------------------------------------------------------------------------

Got request to add KDC with IP Address:kslave.lanslab.edu

kslave.lanslab.edu KDC Added Sucessfully........

Now the number of KDC's in System is:   2

-----------------------------------------------------------------------------------------------

Got request to add KDC with IP Address:kslave2.lanslab.edu

kslave2.lanslab.edu KDC Added Sucessfully........

Now the number of KDC's in System is:   3

-----------------------------------------------------------------------------------------------
```

Fig. 18 : RMI Intermediate Server

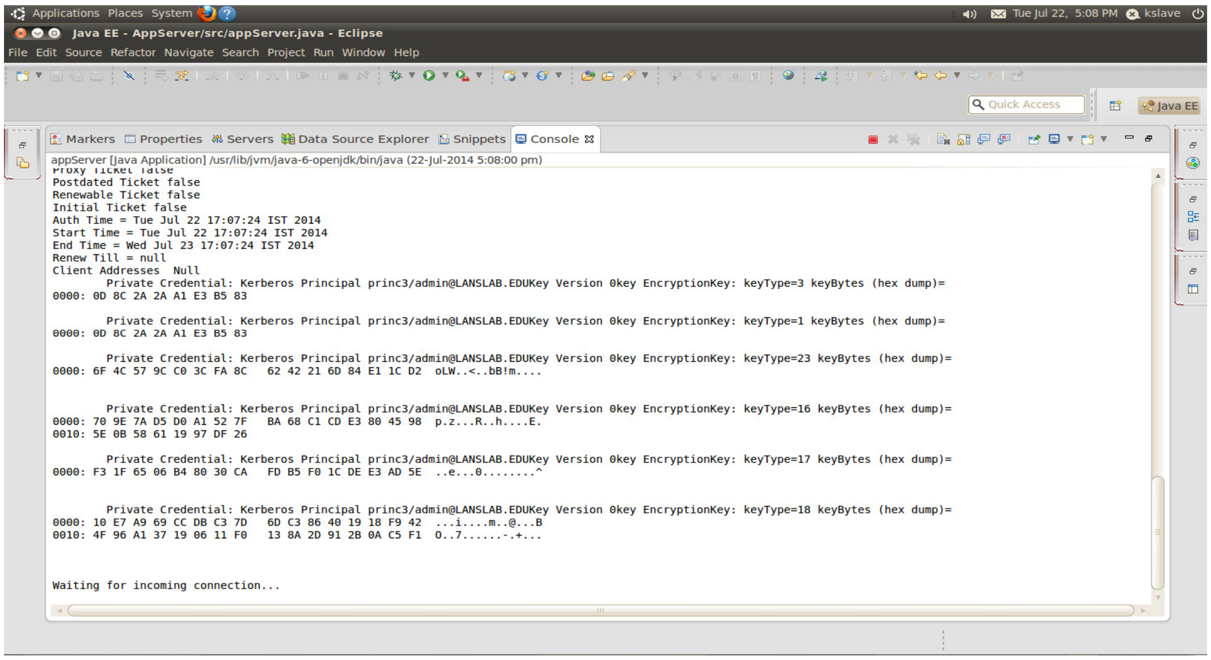
8.3. RMI Client Add KDC


Applications Places System
Tue Jul 22, 5:05 PM kslave
Java EE - AddKDC_MV1/src/AddKDC.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> AddKDC (1) [Java Application] /usr/lib/jvm/java-6-openjdk/bin/java (22-Jul-2014 5:04:38 pm)
AddKDC Program Started......

Sending Current KDC : kslave2.lanslab.edu To kslave2.lanslab.edu:9916

Program is under Process.........

KDC : kslave2.lanslab.edu Added Successfully.......

List of KDCs are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
|       Sr.No.          |              KDC IP                 |       Load Limit       |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
|         1             |       kmaster.lanslab.edu           |          134           |
|-----------------------|-------------------------------------|------------------------|
|         2             |       kslave.lanslab.edu            |          134           |
|-----------------------|-------------------------------------|------------------------|
|         3             |       kslave2.lanslab.edu           |          134           |
|-----------------------|-------------------------------------|------------------------|


The Master KDC is : kmaster.lanslab.edu
----------------------------------------------------------------------------------------
```

Fig. 19 : RMI Client Add KDC

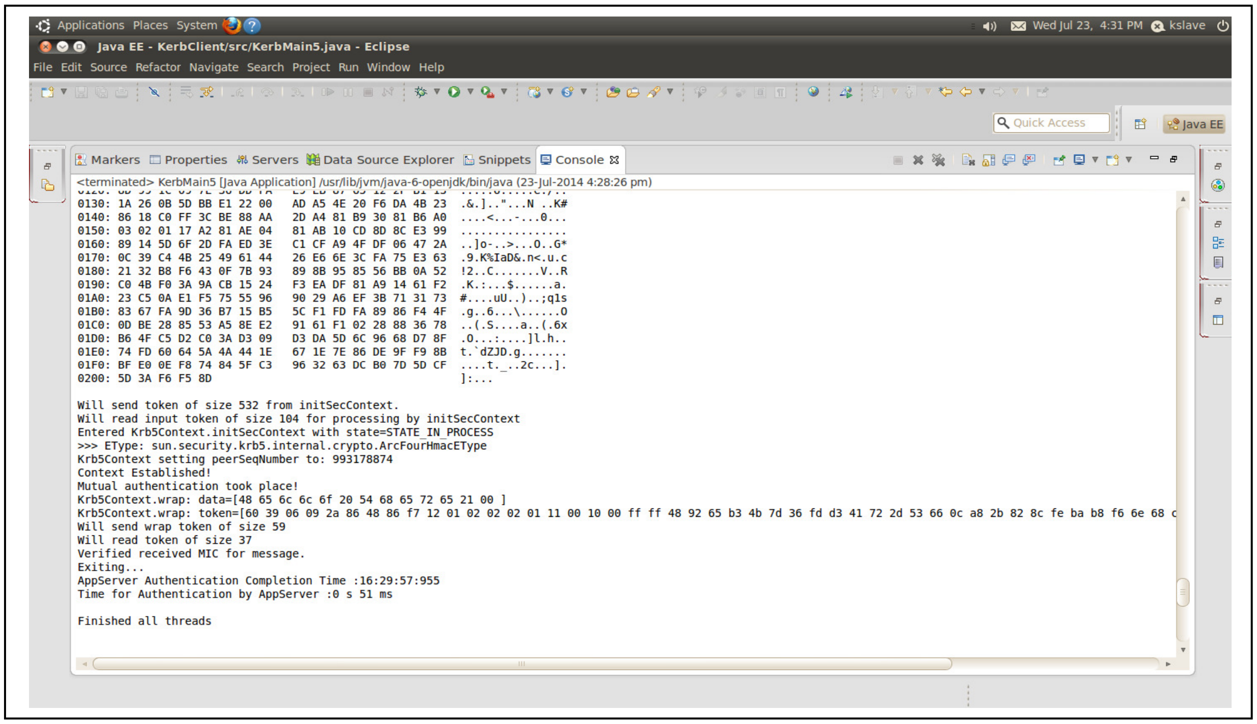
8.4. Service Server


Applications Places System
Tue Jul 22, 5:08 PM kslave
Java EE - AppServer/src/appServer.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Markers Properties Servers Data Source Explorer Snippets Console

```
appServer [Java Application] /usr/lib/jvm/java-6-openjdk/bin/java (22-Jul-2014 5:08:00 pm)
Proxy Ticket false
Postdated Ticket false
Renewable Ticket false
Initial Ticket false
Auth Time = Tue Jul 22 17:07:24 IST 2014
Start Time = Tue Jul 22 17:07:24 IST 2014
End Time = Wed Jul 23 17:07:24 IST 2014
Renew Till = null
Client Addresses  Null
        Private Credential: Kerberos Principal princ3/admin@LANSLAB.EDUKey Version 0key EncryptionKey: keyType=3 keyBytes (hex dump)=
0000: 0D 8C 2A 2A A1 E3 B5 83

        Private Credential: Kerberos Principal princ3/admin@LANSLAB.EDUKey Version 0key EncryptionKey: keyType=1 keyBytes (hex dump)=
0000: 0D 8C 2A 2A A1 E3 B5 83

        Private Credential: Kerberos Principal princ3/admin@LANSLAB.EDUKey Version 0key EncryptionKey: keyType=23 keyBytes (hex dump)=
0000: 6F 4C 57 9C C0 3C FA 8C   62 42 21 6D 84 E1 1C D2  oLW..<..bB!m....


        Private Credential: Kerberos Principal princ3/admin@LANSLAB.EDUKey Version 0key EncryptionKey: keyType=16 keyBytes (hex dump)=
0000: 70 9E 7A D5 D0 A1 52 7F   BA 68 C1 CD E3 80 45 98  p.z...R..h....E.
0010: 5E 0B 58 61 19 97 DF 26

        Private Credential: Kerberos Principal princ3/admin@LANSLAB.EDUKey Version 0key EncryptionKey: keyType=17 keyBytes (hex dump)=
0000: F3 1F 65 06 B4 80 30 CA   FD B5 F0 1C DE E3 AD 5E  ..e...0.......^


        Private Credential: Kerberos Principal princ3/admin@LANSLAB.EDUKey Version 0key EncryptionKey: keyType=18 keyBytes (hex dump)=
0000: 10 E7 A9 69 CC DB C3 7D   6D C3 86 40 19 18 F9 42  ...i....m..@...B
0010: 4F 96 A1 37 19 06 11 F0   13 8A 2D 91 2B 0A C5 F1  O..7......-.+...



Waiting for incoming connection...
```

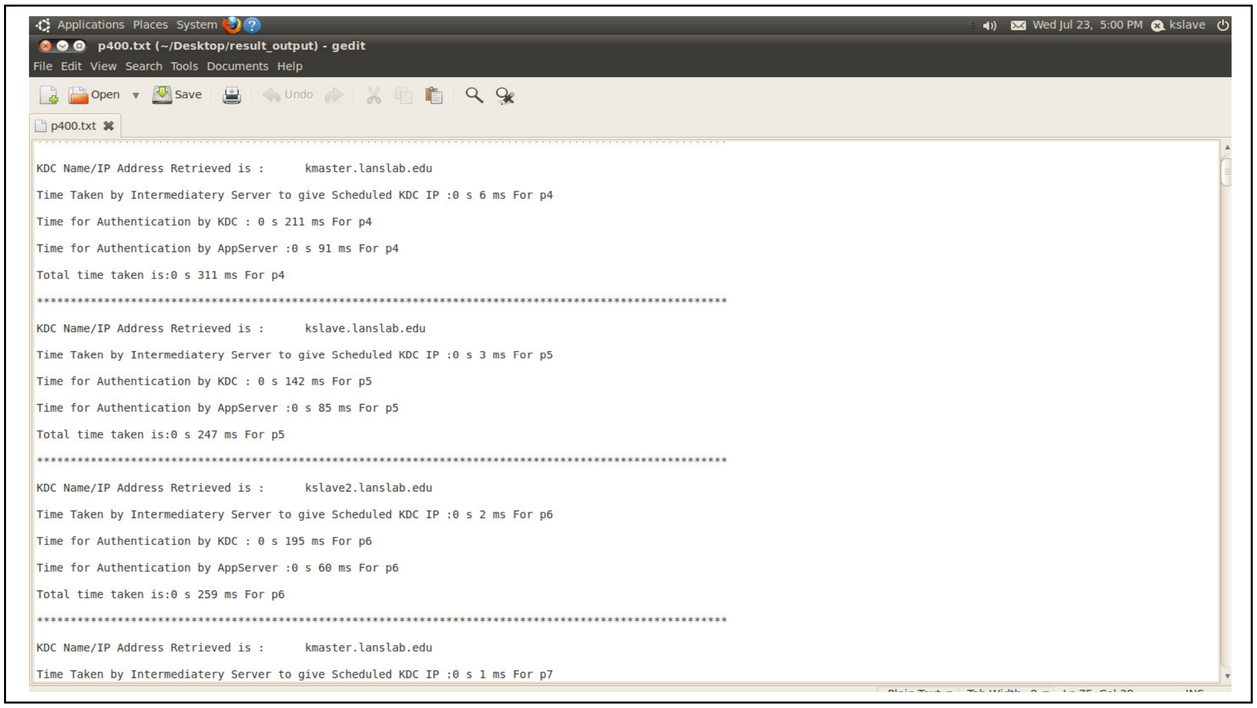
Fig. 20 : AppServer waiting for incoming connection

8.5. RMI Clients



```
<terminated> KerbMain5 [Java Application] /usr/lib/jvm/java-6-openjdk/bin/java (23-Jul-2014 4:28:26 pm)
0130: 1A 26 0B 5D BB E1 22 00   AD A5 4E 20 F6 DA 4B 23  .&.]..."...N ..K#
0140: 86 18 C0 FF 3C BE 88 AA   2D A4 81 B9 30 81 B6 A0  ....<...-...0...
0150: 03 02 01 17 A2 81 AE 04   81 AB 10 CD 8D 8C E3 99  ................
0160: 89 14 5D 6F 2D FA ED 3E   C1 CF A9 4F DF 06 47 2A  ..]o-..>...O..G*
0170: 0C 39 C4 4B 25 49 61 44   26 E6 6E 3C FA 75 E3 63  .9.K%IaD&.n<.u.c
0180: 21 32 B8 F6 43 0F 7B 93   89 8B 95 85 56 BB 0A 52  !2..C.......V..R
0190: C0 4B F0 3A 9A CB 15 24   F3 EA DF 81 A9 14 61 F2  .K.:...$......a.
01A0: 23 C5 0A E1 F5 75 55 96   90 29 A6 EF 3B 71 31 73  #....uU..)..;q1s
01B0: 83 67 FA 9D 36 B7 15 B5   5C F1 FD FA 89 86 F4 4F  .g..6...\......O
01C0: 0D BE 28 85 53 A5 8E E2   91 61 F1 02 28 88 36 78  ..(.S....a..(.6x
01D0: B6 4F C5 D2 C0 3A D3 09   D3 DA 5D 6C 96 68 D7 8F  .O...:....]l.h..
01E0: 74 FD 60 64 5A 4A 44 1E   67 1E 7E 86 DE 9F F9 8B  t.`dZJD.g.......
01F0: BF E0 0E F8 74 84 5F C3   96 32 63 DC B0 7D 5D CF  ....t._..2c...].
0200: 5D 3A F6 F5 8D                                     ]:...

Will send token of size 532 from initSecContext.
Will read input token of size 104 for processing by initSecContext
Entered Krb5Context.initSecContext with state=STATE_IN_PROCESS
>>> EType: sun.security.krb5.internal.crypto.ArcFourHmacEType
Krb5Context setting peerSeqNumber to: 993178874
Context Established!
Mutual authentication took place!
Krb5Context.wrap: data=[48 65 6c 6c 6f 20 54 68 65 72 65 21 00 ]
Krb5Context.wrap: token=[60 39 06 09 2a 86 48 86 f7 12 01 02 02 02 01 11 00 10 00 ff ff 48 92 65 b3 4b 7d 36 fd d3 41 72 2d 53 66 0c a8 2b 82 8c fe ba b8 f6 6e 68 c
Will send wrap token of size 59
Will read token of size 37
Verified received MIC for message.
Exiting...
AppServer Authentication Completion Time :16:29:57:955
Time for Authentication by AppServer :0 s 51 ms

Finished all threads
```

Fig. 21 : Testing of n-Clients at a once for Authentication



```
KDC Name/IP Address Retrieved is :      kmaster.lanslab.edu

Time Taken by Intermediatery Server to give Scheduled KDC IP :0 s 6 ms For p4

Time for Authentication by KDC : 0 s 211 ms For p4

Time for Authentication by AppServer :0 s 91 ms For p4

Total time taken is:0 s 311 ms For p4

*******************************************************************************************************

KDC Name/IP Address Retrieved is :      kslave.lanslab.edu

Time Taken by Intermediatery Server to give Scheduled KDC IP :0 s 3 ms For p5

Time for Authentication by KDC : 0 s 142 ms For p5

Time for Authentication by AppServer :0 s 85 ms For p5

Total time taken is:0 s 247 ms For p5

*******************************************************************************************************

KDC Name/IP Address Retrieved is :      kslave2.lanslab.edu

Time Taken by Intermediatery Server to give Scheduled KDC IP :0 s 2 ms For p6

Time for Authentication by KDC : 0 s 195 ms For p6

Time for Authentication by AppServer :0 s 60 ms For p6

Total time taken is:0 s 259 ms For p6

*******************************************************************************************************

KDC Name/IP Address Retrieved is :      kmaster.lanslab.edu

Time Taken by Intermediatery Server to give Scheduled KDC IP :0 s 1 ms For p7
```

Fig. 22 : Generated Log containing Each Authentication Time of Clients

8.6. Load Testing

For load balancing we have implemented the RoundRobin Scheduling so that it will give one Scheduled KDC Name/IP in cyclic manner. We carried out the load testing of implemented system with 3 KDCs (1 Master KDC and 2 Slave KDCs) and 200 Clients at once. For this purpose we used the Java ExecutorService. The testing results are as follows:

Principal Name	Schedule KDC Name/IP returned	Time Taken by Intermediate Server to give Scheduled KDC Name/IP (msec)	Time Taken by KDC to Authenticate the Client (msec)	Time Taken by AppServer to Authenticate Client (msec)	Total Time of Kerberos Authentication (msec)
P1	kmaster.lanslab.edu	3	182	82	269
P2	kslave.lanslab.edu	2	179	79	264
P3	kslave2.lanslab.edu	1	171	72	249
P4	kmaster.lanslab.edu	2	172	82	258
P5	kslave.lanslab.edu	2	211	81	298
P6	kslave2.lanslab.edu	2	203	84	291
P7	kmaster.lanslab.edu	2	179	103	289
P8	kslave.lanslab.edu	2	155	53	211
P9	kslave2.lanslab.edu	2	190	73	279
P10	kmaster.lanslab.edu	3	171	71	260
P11	kslave.lanslab.edu	1	179	57	249
P12	kslave2.lanslab.edu	2	147	50	200
P13	kmaster.lanslab.edu	2	180	54	239
P14	kslave.lanslab.edu	8	190	60	264
P15	kslave2.lanslab.edu	4	157	49	221
P16	kmaster.lanslab.edu	2	191	63	267
P17	kslave.lanslab.edu	9	173	57	251
P18	kslave2.lanslab.edu	2	156	70	238
P19	kmaster.lanslab.edu	2	141	57	201
P20	kslave.lanslab.edu	5	145	60	222
P21	kslave2.lanslab.edu	11	163	61	247
P22	kmaster.lanslab.edu	2	164	72	250
P23	kslave.lanslab.edu	9	163	69	251
P24	kslave2.lanslab.edu	2	193	61	261
P25	kmaster.lanslab.edu	5	173	62	247

Table 4.1 : Load Testing Result

Principal Name	Schedule KDC Name/IP returned	Time Taken by Intermediate Server to give Scheduled KDC Name/IP (msec)	Time Taken by KDC to Authenticate the Client (msec)	Time Taken by AppServer to Authenticate Client (msec)	Total Time of Kerberos Authentication (msec)
P26	kslave.lanslab.edu	2	160	44	212
P27	kslave2.lanslab.edu	2	171	55	238
P28	kmaster.lanslab.edu	3	176	65	253
P29	kslave.lanslab.edu	1	128	48	178
P30	kslave2.lanslab.edu	4	172	64	250
P31	kmaster.lanslab.edu	2	154	50	209
P32	kslave.lanslab.edu	1	183	55	249
P33	kslave2.lanslab.edu	2	160	56	231
P34	kmaster.lanslab.edu	2	161	85	249
P35	kslave.lanslab.edu	2	153	69	231
P36	kslave2.lanslab.edu	1	187	59	250
P37	kmaster.lanslab.edu	1	182	58	251
P38	kslave.lanslab.edu	2	140	56	209
P39	kslave2.lanslab.edu	2	161	65	229
P40	kmaster.lanslab.edu	2	169	60	243
P41	kslave.lanslab.edu	2	176	55	240
P42	kslave2.lanslab.edu	2	168	61	242
P43	kmaster.lanslab.edu	2	112	49	165
P44	kslave.lanslab.edu	2	154	43	200
P45	kslave2.lanslab.edu	2	165	49	230
P46	kmaster.lanslab.edu	1	154	56	221
P47	kslave.lanslab.edu	1	152	57	219
P48	kslave2.lanslab.edu	1	167	46	229
P49	kmaster.lanslab.edu	1	199	54	259
P50	kslave.lanslab.edu	1	146	43	191

Table 4.2 : Load Testing Result

Principal Name	Schedule KDC Name/IP returned	Time Taken by Intermediate Server to give Scheduled KDC Name/IP (msec)	Time Taken by KDC to Authenticate the Client (msec)	Time Taken by AppServer to Authenticate Client (msec)	Total Time of Kerberos Authentication (msec)
P51	kslave2.lanslab.edu	2	174	72	249
P52	kmaster.lanslab.edu	2	154	84	251
P53	kslave.lanslab.edu	5	169	64	238
P54	kslave2.lanslab.edu	2	182	49	241
P55	kmaster.lanslab.edu	1	183	64	259
P56	kslave.lanslab.edu	3	63	59	239
P57	kslave2.lanslab.edu	2	148	53	211
P58	kmaster.lanslab.edu	2	154	73	239
P59	kslave.lanslab.edu	2	146	49	199
P60	kslave2.lanslab.edu	1	180	48	230
P61	kmaster.lanslab.edu	2	151	45	199
P62	kslave.lanslab.edu	2	163	70	238
P63	kslave2.lanslab.edu	2	153	56	214
P64	kmaster.lanslab.edu	2	190	54	260
P65	kslave.lanslab.edu	1	183	55	250
P66	kslave2.lanslab.edu	2	185	54	250
P67	kmaster.lanslab.edu	1	192	54	257
P68	kslave.lanslab.edu	1	169	47	219
P69	kslave2.lanslab.edu	2	182	65	250
P70	kmaster.lanslab.edu	2	156	66	227
P71	kslave.lanslab.edu	1	185	52	252
P72	kslave2.lanslab.edu	1	180	58	240
P73	kmaster.lanslab.edu	1	142	66	210
P74	kslave.lanslab.edu	1	178	70	258
P75	kslave2.lanslab.edu	2	155	53	230

Table 4.3 : Load Testing Result

Principal Name	Schedule KDC Name/IP returned	Time Taken by Intermediate Server to give Scheduled KDC Name/IP (msec)	Time Taken by KDC to Authenticate the Client (msec)	Time Taken by AppServer to Authenticate Client (msec)	Total Time of Kerberos Authentication (msec)
P76	kmaster.lanslab.edu	3	178	49	230
P77	kslave.lanslab.edu	1	178	47	238
P78	kslave2.lanslab.edu	1	173	54	240
P79	kmaster.lanslab.edu	2	154	56	220
P80	kslave.lanslab.edu	1	173	54	229
P81	kslave2.lanslab.edu	5	170	51	239
P82	kmaster.lanslab.edu	1	171	64	240
P83	kslave.lanslab.edu	1	151	62	229
P84	kslave2.lanslab.edu	1	161	54	217
P85	kmaster.lanslab.edu	4	197	61	262
P86	kslave.lanslab.edu	2	165	50	221
P87	kslave2.lanslab.edu	1	163	71	239
P88	kmaster.lanslab.edu	1	166	51	229
P89	kslave.lanslab.edu	1	196	60	258
P90	kslave2.lanslab.edu	1	143	54	209
P91	kmaster.lanslab.edu	2	173	58	241
P92	kslave.lanslab.edu	1	170	45	228
P93	kslave2.lanslab.edu	1	158	57	229
P94	kmaster.lanslab.edu	1	142	65	209
P95	kslave.lanslab.edu	1	163	46	229
P96	kslave2.lanslab.edu	1	165	60	230
P97	kmaster.lanslab.edu	2	162	45	219
P98	kslave.lanslab.edu	1	173	47	229
P99	kslave2.lanslab.edu	1	162	69	249
P100	kmaster.lanslab.edu	1	173	53	230

Table. 4.4 : Load Testing Result

Principal Name	Schedule KDC Name/IP returned	Time Taken by Intermediate Server to give Scheduled KDC Name/IP (msec)	Time Taken by KDC to Authenticate the Client (msec)	Time Taken by AppServer to Authenticate Client (msec)	Total Time of Kerberos Authentication (msec)
P101	kslave.lanslab.edu	2	184	63	249
P102	kslave2.lanslab.edu	2	177	50	239
P103	kmaster.lanslab.edu	1	171	61	239
P104	kslave.lanslab.edu	1	133	42	177
P105	kslave2.lanslab.edu	3	187	53	250
P106	kmaster.lanslab.edu	1	183	53	249
P107	kslave.lanslab.edu	4	180	55	251
P108	kslave2.lanslab.edu	2	161	57	230
P109	kmaster.lanslab.edu	2	154	64	228
P110	kslave.lanslab.edu	1	174	63	240
P111	kslave2.lanslab.edu	5	144	59	221
P112	kmaster.lanslab.edu	2	168	64	238
P113	kslave.lanslab.edu	1	146	49	208
P114	kslave2.lanslab.edu	1	173	53	239
P115	kmaster.lanslab.edu	1	158	59	219
P116	kslave.lanslab.edu	1	180	52	241
P117	kslave2.lanslab.edu	2	175	51	231
P118	kmaster.lanslab.edu	2	158	79	239
P119	kslave.lanslab.edu	1	171	55	228
P120	kslave2.lanslab.edu	1	174	69	249
P121	kmaster.lanslab.edu	1	172	47	231
P122	kslave.lanslab.edu	2	167	54	227
P123	kslave2.lanslab.edu	2	185	92	281
P124	kmaster.lanslab.edu	1	168	47	218
P125	kslave.lanslab.edu	1	149	50	201

Table. 4.5 : Load Testing Result

Principal Name	Schedule KDC Name/IP returned	Time Taken by Intermediate Server to give Scheduled KDC Name/IP (msec)	Time Taken by KDC to Authenticate the Client (msec)	Time Taken by AppServer to Authenticate Client (msec)	Total Time of Kerberos Authentication (msec)
P126	kslave2.lanslab.edu	1	170	65	242
P127	kmaster.lanslab.edu	3	166	50	223
P128	kslave.lanslab.edu	2	160	45	219
P129	kslave2.lanslab.edu	1	171	65	240
P130	kmaster.lanslab.edu	1	139	47	189
P131	kslave.lanslab.edu	1	173	46	228
P132	kslave2.lanslab.edu	1	165	55	232
P133	kmaster.lanslab.edu	2	168	56	238
P134	kslave.lanslab.edu	1	153	54	209
P135	kslave2.lanslab.edu	2	187	49	239
P136	kmaster.lanslab.edu	1	176	53	239
P137	kslave.lanslab.edu	2	166	53	230
P138	kslave2.lanslab.edu	3	172	58	237
P139	kmaster.lanslab.edu	5	168	53	241
P140	kslave.lanslab.edu	2	176	56	245
P141	kslave2.lanslab.edu	2	167	50	224
P142	kmaster.lanslab.edu	2	172	47	232
P143	kslave.lanslab.edu	1	188	48	249
P144	kslave2.lanslab.edu	7	216	63	291
P145	kmaster.lanslab.edu	2	210	62	286
P146	kslave.lanslab.edu	2	164	62	229
P147	kslave2.lanslab.edu	1	149	49	200
P148	kmaster.lanslab.edu	2	172	60	234
P149	kslave.lanslab.edu	2	168	75	245
P150	kslave2.lanslab.edu	1	158	51	211

Table. 4.6 : Load Testing Result

Principal Name	Schedule KDC Name/IP returned	Time Taken by Intermediate Server to give Scheduled KDC Name/IP (msec)	Time Taken by KDC to Authenticate the Client (msec)	Time Taken by AppServer to Authenticate Client (msec)	Total Time of Kerberos Authentication (msec)
P151	kmaster.lanslab.edu	2	169	50	228
P152	kslave.lanslab.edu	4	169	48	231
P153	kslave2.lanslab.edu	1	170	57	239
P154	kmaster.lanslab.edu	2	166	60	229
P155	kslave.lanslab.edu	2	149	67	219
P156	kslave2.lanslab.edu	2	161	58	222
P157	kmaster.lanslab.edu	2	159	49	217
P158	kmaster.lanslab.edu	2	169	46	221
P159	kslave.lanslab.edu	2	150	62	218
P160	kslave2.lanslab.edu	1	183	49	248
P161	kmaster.lanslab.edu	1	177	61	240
P162	kslave.lanslab.edu	1	168	61	238
P163	kslave2.lanslab.edu	1	186	56	250
P164	kmaster.lanslab.edu	2	203	42	259
P165	kslave.lanslab.edu	3	204	43	269
P166	kslave2.lanslab.edu	1	176	58	250
P167	kmaster.lanslab.edu	2	182	55	249
P168	kslave.lanslab.edu	1	174	62	242
P169	kslave2.lanslab.edu	2	182	52	247
P170	kmaster.lanslab.edu	1	186	57	250
P171	kslave.lanslab.edu	2	168	63	245
P172	kslave2.lanslab.edu	3	149	58	230
P173	kmaster.lanslab.edu	2	178	47	228
P174	kslave.lanslab.edu	1	183	55	240
P175	kslave2.lanslab.edu	1	175	47	238

Table. 4.7 : Load Testing Result

Principal Name	Schedule KDC Name/IP returned	Time Taken by Intermediate Server to give Scheduled KDC Name/IP (msec)	Time Taken by KDC to Authenticate the Client (msec)	Time Taken by AppServer to Authenticate Client (msec)	Total Time of Kerberos Authentication (msec)
P176	kmaster.lanslab.edu	1	169	58	229
P177	kmaster.lanslab.edu	1	112	51	169
P178	kslave.lanslab.edu	1	118	49	199
P179	kslave2.lanslab.edu	3	159	56	219
P180	kmaster.lanslab.edu	1	160	55	220
P181	kslave.lanslab.edu	2	173	63	241
P182	kslave2.lanslab.edu	1	155	60	218
P183	kmaster.lanslab.edu	1	169	52	229
P184	kslave.lanslab.edu	1	166	60	228
P185	kslave2.lanslab.edu	2	173	54	231
P186	kmaster.lanslab.edu	1	162	63	230
P187	kmaster.lanslab.edu	4	151	50	218
P188	kslave.lanslab.edu	1	155	49	210
P189	kslave2.lanslab.edu	4	161	74	240
P190	kmaster.lanslab.edu	1	170	58	239
P191	kslave.lanslab.edu	2	159	57	219
P192	kslave2.lanslab.edu	2	141	55	202
P193	kmaster.lanslab.edu	1	141	43	187
P194	kslave.lanslab.edu	2	167	53	231
P195	kslave2.lanslab.edu	2	176	61	240
P196	kmaster.lanslab.edu	4	182	73	268
P197	kslave.lanslab.edu	2	153	47	209
P198	kslave2.lanslab.edu	2	170	49	224
P199	kmaster.lanslab.edu	2	187	48	238
P200	kslave.lanslab.edu	1	163	47	219
	Avg. Time	**1.98**	**167.345**	**57.54**	**234.18**

Table. 4.8 : Load Testing Result

The above Table. clearly shows that the introduction of intermediate server only adds approximately 2 msec in overall Kerberos authentication. But the load balanced nature of proposed architecture will share the equal load on each KDC which improves the Kerberos Authentication Time significantly.

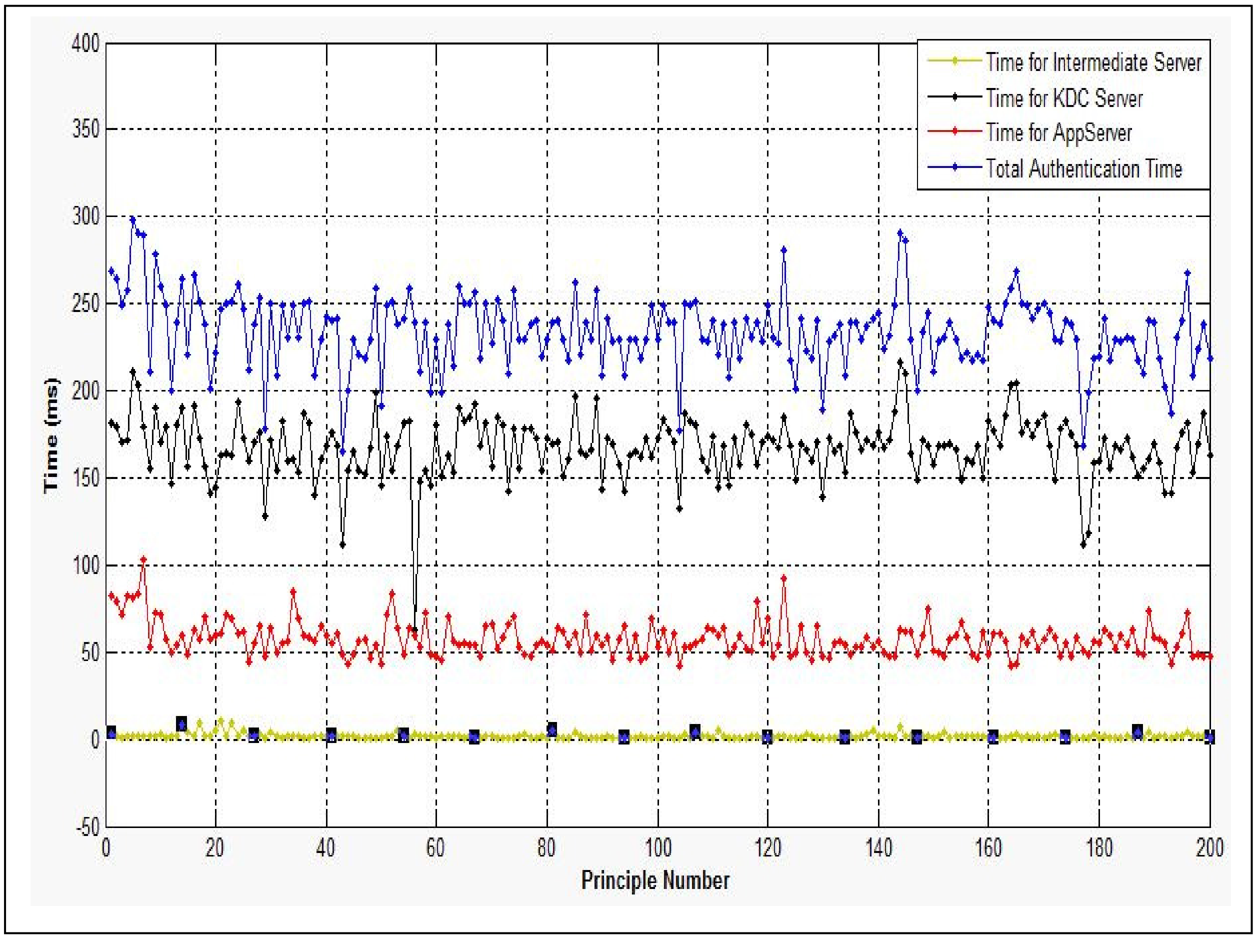


Fig. 23 : Principal Number VS Authentication Times

8.7. Comparison of Average KDC Authentication Time

The comparison is made on the basis of Average KDC Authentication Time for two scenario:

1. Without Load Balancing
2. With Load Balancing

The following Table shows this comparison upto 200 clients and using 2 KDCs:

Sr. No.	No. of Clients	Avg. KDC Authentication Time with Load Balancing (msec)	Avg. KDC Authentication Time without Load Balancing (msec)
1	10	220.3	220.1
2	50	182.48	196.92
3	75	174.02	184.3
4	100	167.97	177.48
5	125	165.96	174.15
6	150	165.77	173.37
7	175	165.45	170.97
8	200	165.63	169.56

Table 5 : Avg. KDC Authentication Time Comparison

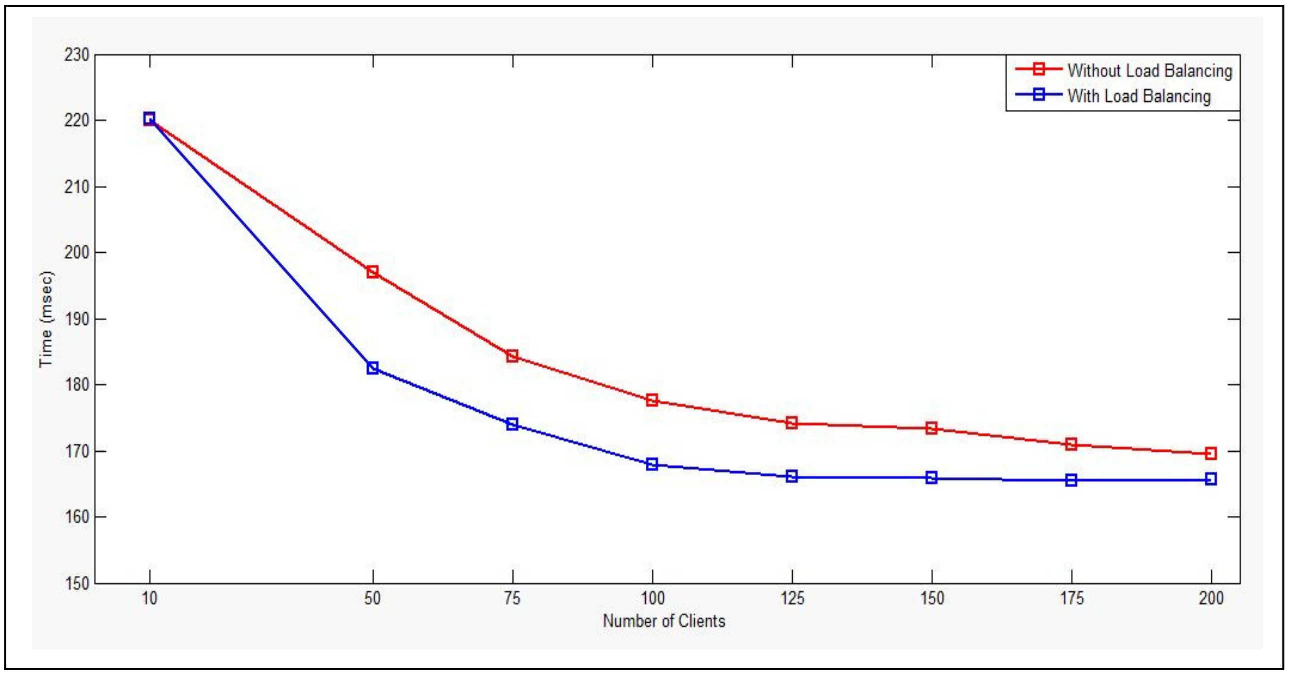


Fig. 24 : No. of Clients VS Avg. KDC Authentication Time

CHAPTER 9 : CONCLUSION AND FUTURE WORK

9.1. Conclusion

Kerberos is symmetric, any two parties can authenticate to prevent phishing and other malicious attacks. Also, Kerberos makes mutual authentication simple. While Kerberos solve most modern distributed systems requirements, it is architecturally sound which makes it easy to embed into other systems. In current days, Kerberos is integral part of IT infrastructure.

The serious problem with giving all authentication request on single KDC is if the current KDC goes down, then there would not be any entity to authenticate the users. The Replicated Kerberos implemented with Master-Slave KDCs solves this problem. But, again the slave KDCs are only used when the Master KDC is unavailable or down. With use of Load Balancing implemented in Intermediate Server makes whole Authentication Model fast and reliable.

Use of RMI solves the problem of opening and closing of sockets with respective user request and it makes authentication even simpler. It also shifts processing load from Client to Intermediate Server, since client has only request to Intermediate Server, the Scheduled KDC Name/IP to which it can send the authentication request.

The experimental results of this project shows that the introduction of Intermediate Server only adds 2 msec in whole Kerberos authentication process, but it adds reliability through replication and load balancing, hence increased the availability of authenticated servers.

9.2. Future Work

In future, to solve the Intermediate Server's bottleneck problem we want to extend it to Rigorous Binary Tree Code Algorithm [25]. Also, The Kerberized Java Client can be integrated with Doctor Application for proper user of Kerberos authentication. Also, RBAC [20] can also be integrated with Java Application Server for managing user permissions and to implement group access control.

There is possibility to deploy this Replicated, Load Balanced Master-Slave Kerberos Model on CloudStack[21] Cloud to give Authentication As A Service. This will make this architecture even more fast, reliable and scalable on demand.

It can also be extended to generate soft tokens for authenticated users for two-factor authentication security.[22]

REFERENCES

[1] Dr. Shilekh Mittal, " Eye Opening Aspect of Telemedicine in Punjab", Sciknow Publication, Health Care, 2013.

[2] Asad Amir Pirzada et.al., 2004, "Kerberos Assisted Authentication in Mobile Ad-hoc Networks", ACSC '04 Proceedings of the 27th Australian conference on Computer Science.

[3] Sanjeev Kumar Pippal et.al., 2011, "CTES based Secure approach for Authentication and Authorization of Resource and Service in Clouds", International Conference on Computer and communication Technology (ICCCT) - 2011.

[4] SeungJun Bang et.al, 2007, "Implementation and Performance Evaluation of Socket and RMI based Java Message Passing Systems", Fifth International Conference on Software Engineering Research, Management and Applications, IEEE, 2007.

[5] Kerberos: The Network Authentication Protocol
http://web.mit.edu/kerberos/

[6] Kerberos Protocol
http://en.wikipedia.org/wiki/Kerberos_(protocol)

[7] Charlie Kaufman et.al., Book, Network Security, PRIVATE Communication in PUBLIC World, Second Edition.

[8] Jason Garman, Book, Kerberos : The Definitive Guide, O'Reilly Publication, 2003

[9] Essentials of Kerberos Authentication
http://consultingblogs.emc.com/markwilson/archive/2005/06/06/1541.aspx

[10] Kerberos Authentication Explained
http://consultingblogs.emc.com/markwilson/archive/2005/06/06/1541.aspx

[11] MIT Consortium, Why is Kerberos a Credible Security Solution?
http://www.kerberos.org/software/whykerberos.pdf

[12] Kerberos Server Replication
http://tldp.org/HOWTO/Kerberos-Infrastructure-HOWTO/server-replication.html

[13] Ubuntu 10.04 Server Guide
https://help.ubuntu.com/10.04/serverguide/serverguide.pdf

[14] JAAS Guide by Oracle
http://docs.oracle.com/javase/7/docs/technotes/guides/security/jaas/JAASRefGuide.ht
ml

[15] Scalable Java Security with JAAS
http://www.javaworld.com/article/2074873/java-web-development/all-that-jaas.html

[16] Anders Fongen, RMI Lab Tutorial
http://www.java.no/web/files/moter/tuplespace.pdf

[17] Kerberos Server Guide
https://help.ubuntu.com/10.04/serverguide/kerberos.html
[18] Kerberos Master Slave Replication Guide
www.rjsystems.nl/en/2100-d6-kerberos-slave.php
[19] GSS-API Programming Guide
http://docs.oracle.com/cd/E19455-01/806-3814/6jcugr7d6/index.html
[20] Role Based Access Control
http://en.wikipedia.org/wiki/Rbac
[21] Apache CloudStack Website
http://cloudstack.apache.org/
[22] Pkinit Configuration and Soft Tokens
http://k5wiki.kerberos.org/wiki/Pkinit_configuration
[23]Generic Security Services - Application Program Interface
http://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interfa
ce
[24]Introduction to JAAS and Java GSS-API Tutorials
http://docs.oracle.com/javase/7/docs/technotes/guides/security/jgss/tutorials/index.htm
l
[25] Hongjun Liu et.al., 2007, "A distributed expansible authentication model based on Kerberos", ScienceDirect, Journal of Network and Computer Applications 31 (2008) 472-486
[26] AskUbuntu Website
http://askubuntu.com/
[27] Stackoverflow Website
http://stackoverflow.com/

APPENDIX

Appendix 1

Propagation Script on Master KDC:

Create file name /etc/cron.hourly/krb5-prop with following contents:

```
#!/bin/sh

# Distribute KDC database to slave servers
slavekdcs=”kslave.lanslab.edu kslave2.lanslab.edu kslave3.lanslab.edu”

/usr/sbin/kdb5_util dump /var/lib/krb5kdc/dump
error=$?
if [ $error -ne 0 ]; then
        echo "Kerberos database dump failed"
        echo "with exit code $error. Exciting."
        exit 1
fi
for kdc in $slavekdcs; do
        /usr/sbin/kprop $kdc >
        error=$?
        if [ $error -ne 0 ]; then
                echo "Propagation of database to host $kdc"
                echo "failed with exit code $error."
        fi
done
exit 0
```

Appendix 2

Intermediate Java RMI Server's Source Code:

1. InterMServer.java

```
import java.net.InetAddress;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Iterator;

import java.util.LinkedHashMap;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;


public class InterMServer extends UnicastRemoteObject implements KDCInterface {

String servername,KDC,masterKDC;

Registry registry;

int key_count = 0;//rschcount=0; // storing the count of setipKDCMap's count

public static int rrcount=0; // for cycle through KDC list

public static SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss:SSS");

public static String KDCinitTime; /* KDC's init Time i.e. initialisation
time/start/addtime of KDC */
```

```
Map<String, Integer> ipKDCLLMap = new LinkedHashMap<String, Integer>();// HashMap<String, Integer>();

/*-----------------------------------------------------------------------------------------------------
---------------*/

/*-----------------------------------------------------------------------------------------------------
---------------*/

public InterMServer() throws RemoteException{

try {

        KDCinitTime = sdf.format(new Date());

        System.out.println("Intermediate Server's Major Version 1 /KDC Communication Server Init Time : "+KDCinitTime+"\n");

        InetAddress IP=InetAddress.getLocalHost();

        servername = IP.getCanonicalHostName();

        System.out.println("IP of my system is := "+servername+"\n");

} catch (Exception e) {

        // TODO: handle exception

        System.out.println("Can't get inet address....\n");

}

int port = 9916;

System.out.println("This Intermediate Server's KDCInterface Address = "+ servername + "\tPort="+ port+"\n");

try {

registry = LocateRegistry.createRegistry(port);

registry.rebind("InterMServer", this);

} catch (Exception e) {

        // TODO: handle exception

        System.out.println("Remote Exception"+e);
        }
}
/*-----------------------------------------------------------------------------------------------------
---------------*/
@Override
public void addKDC(String x) throws RemoteException {
// TODO Auto-generated method stub
System.out.println("Got request to add KDC with IP Address:"+x+"\n");
int y = 134; // load balancing purpose
ipKDCLLMap.put(x, y);
++key_count;
System.out.println(x+" KDC Added Sucessfully........\n");
System.out.println("Now the number of KDC's in System is:\t"+key_count+"\n");
System.out.println("--------------------------------------------------------------------------------
------------------\n");
}
@Override
public Map<String, Integer> getCurrentKDCs() throws RemoteException {
// TODO Auto-generated method stub
return ipKDCLLMap;
}
@Override
public String getMasterKDC() throws RemoteException {
// TODO Auto-generated method stub
Iterator<Entry<String, Integer>> iter = ipKDCLLMap.entrySet().iterator();
```

```
Entry<String, Integer> next = iter.next();

return next.getKey();//masterKDC;

}

@Override

public String getSchKDCip() throws RemoteException {

// TODO Auto-generated method stub

/*This implementation of Interface gives the IP of KDC from Intermediate server to
the client to next authentication process*/

rrcount++;

System.out.println("Value of RRCount is:"+rrcount);

String rrSchip = RoundRSch(ipKDCLLMap,rrcount);

return rrSchip;

}

private String RoundRSch(Map<String, Integer> ipKDCLLMap2, int rrcount2) {

// TODO Auto-generated method stub

int countKDC = 0;

String rrschIP="";

countKDC = ipKDCLLMap2.size();

System.out.println("No of KDC's in System are:"+countKDC);

/*Iterator to rotate through the KDCs IP*/

Iterator<Entry<String, Integer>> iter = ipKDCLLMap.entrySet().iterator();

Entry<String, Integer> next1 = null;

/*Following code is actual scheduling the KDC IP*/

// 3rd solution

int jump = rrcount2 % countKDC;
```

```
System.out.println("Jump Value is :"+jump);
if (jump != 0) {
while(iter.hasNext()){
        for(int i=0;i<jump;i++){
                next1=iter.next();
                System.out.println(next1.getKey());
                }
        break;
        }
} else {
while(iter.hasNext()){
        for(int i=0;i<countKDC;i++){
                next1=iter.next();
                System.out.println(next1.getKey());
                        }
        break;
                }
}
rrschIP = next1.getKey();
System.out.println("Scheduled KDC at first is:"+rrschIP);
/*Uptil now the scheduled KDC IP is returned now update the load value of scheduled
KDC IP's LoadLimit....*/
/*Check the load limit value and if it is greater than zero assign the respective IP to
client
 * and update the corresponding Load limit of KDC IP*/
```

```
if(next1.getValue() != 0){
ipKDCLLMap.put(rrschIP,ipKDCLLMap.get(rrschIP)-1);
}else {
        while(iter.hasNext()){
                next1 = iter.next();
                }
        rrschIP = next1.getKey();
        ipKDCLLMap.put(rrschIP,ipKDCLLMap.get(rrschIP)-1);
        }

        if(ipKDCLLMap.get(rrschIP)<0){
        //return "Load limit of Scheduled KDC exceeds...Try Again";
        ipKDCLLMap.put(rrschIP, 0);
        return null;
        }
        else
        return rrschIP;
        }
}
```

2. RMI Interface (Same for Java RMI Server and its Clients):

KDCInterface.java

```
import java.rmi.Remote;

import java.rmi.RemoteException;

import java.util.List;

import java.util.Map;

public interface KDCInterface extends Remote {

        void addKDC(String x) throws RemoteException;

        public Map<String, Integer> getCurrentKDCs() throws RemoteException;

        String getMasterKDC() throws RemoteException;

        //This is Experimental with Client

        String getSchKDCip() throws RemoteException;

}
```

Appendix 3

Source Code of Addition of KDC's to Intermediate Server's KDC List :

1. AddKDC.java

```
import java.rmi.NotBoundException;

import java.rmi.RMISecurityManager;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

public class AddKDC {

public static void main(String args[]){

KDCInterface IServer;

Registry registry;

String serverAddress = args[0];

String serverPort = args[1];

String text = args[2];

System.out.println("AddKDC Program Started......\n");

System.out.println("Sending Current KDC : "+text+" To
"+serverAddress+":"+serverPort+"\n");

try {

System.setSecurityManager(new RMISecurityManager());

registry = LocateRegistry.getRegistry(serverAddress,(new Integer(serverPort)).intValue());
IServer = (KDCInterface)(registry.lookup("InterMServer")); /* for CheckMaster.java file its CheckMasterServer & for InServer.java file its IntServer */

//call the remote method

System.out.println("Program is under Process.........\n");

IServer.addKDC(text);

System.out.println("KDC : "+text+" Added Successfully.......\n");

Map<String, Integer> KDClist = IServer.getCurrentKDCs();

System.out.println("List of KDCs are:\t");//+KDClist);

System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

System.out.println("|\tSr.No.\t\t|"+"\t\tKDC IP\t\t\t|"+"\tLoad Limit\t|");

System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

Iterator<Entry<String, Integer>> iter = KDClist.entrySet().iterator();

int j=1;		// Used for Sr. No. Field

	while (iter.hasNext()) {

		Entry<String, Integer> next = iter.next();

		Integer value = next.getValue();

		System.out.println("|\t  "+j+"\t\t|\t"+next.getKey()+"\t\t|\t    "+value+"\t\t|\t");

		System.out.println("|-----------------------|---------------------------------------|-----------------------|\t");

		j++;

		}

System.out.println("\n");

System.out.print("The Master KDC is : ");

String masterkdc = IServer.getMasterKDC();
```

```
System.out.println(masterkdc);

System.out.println("--------------------------------------------------------------------\n");

System.out.println("");


// Getting Scheduled IP for Processing

/*String schKDC = IServer.getSchKDCip();

System.out.println("Scheduled KDC is:\n"+schKDC);

*/

} catch (RemoteException e) {

        // TODO: handle exception

        e.printStackTrace();

                }

        catch (NotBoundException e) {

        // TODO: handle exception

        System.err.println(e);

                }

        }

}
```

2. RMI Interface (Same for Java RMI Server and its Clients):

KDCInterface.java

```
import java.rmi.Remote;

import java.rmi.RemoteException;

import java.util.List;

import java.util.Map;
```

```
public interface KDCInterface extends Remote {

        void addKDC(String x) throws RemoteException;

        public Map<String, Integer> getCurrentKDCs() throws RemoteException;

        String getMasterKDC() throws RemoteException;

        //This is Experimental with Client

        String getSchKDCip() throws RemoteException;

}
```

Appendix 4

Application Server's Source Code :

1. appServer.java

```
import org.ietf.jgss.*;

import java.io.*;

import java.net.Socket;

import java.net.ServerSocket;

import java.security.PrivilegedAction;

import javax.security.auth.Subject;

import javax.security.auth.callback.Callback;

import javax.security.auth.callback.CallbackHandler;

import javax.security.auth.callback.NameCallback;

import javax.security.auth.callback.PasswordCallback;

import javax.security.auth.callback.UnsupportedCallbackException;

import javax.security.auth.login.LoginContext;

import javax.security.auth.login.LoginException;
```

```
public class appServer  {


static Subject serviceSubject = new Subject();

static LoginContext context = null;

int localPort=9990;

public static void main(String[] args) throws IOException, GSSException {

System.setProperty( "sun.security.krb5.debug", "true");

System.setProperty( "java.security.krb5.realm","LANSLAB.EDU");

System.setProperty( "java.security.krb5.kdc","172.16.6.18");

System.setProperty( "java.security.auth.login.config", "./jaas.conf");

System.setProperty("javax.security.auth.useSubjectCredsOnly","false");

final String username = "princ3/admin";

final String password = "princ3";

final int localPort = 9990;

// Create LoginContext with a callback handler

try {

context = new LoginContext("server", new LoginCallbackHandler(username , password));

// Perform authentication

context.login();

} catch (LoginException e) {

        // TODO Auto-generated catch block

        e.printStackTrace();

        }

System.out.println(context.getSubject());
// Perform action as authenticated user
serviceSubject = context.getSubject();
ServerSocket ss = new ServerSocket(localPort);
while(true)
        {
        System.out.println("Waiting for incoming connection...");
        Socket socket = ss.accept();
        (new ServerThread(socket,serviceSubject)).start();
        }
    }
}
```

2. LoginCallbackHandler.java

```
import java.io.IOException;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.callback.UnsupportedCallbackException;

class LoginCallbackHandler implements CallbackHandler {
private String password;
private String username;
public LoginCallbackHandler() {
          super();
}



public LoginCallbackHandler( String name, String password) {
    super();
    this.username = name;
    this.password = password;
}
public void handle( Callback[] callbacks)
     throws IOException, UnsupportedCallbackException {
    for ( int i=0; i<callbacks.length; i++) {
            if ( callbacks[i] instanceof NameCallback && username != null) {
            NameCallback nc = (NameCallback) callbacks[i];
            nc.setName( username);
            }
else
            if ( callbacks[i] instanceof PasswordCallback) {
            PasswordCallback pc = (PasswordCallback) callbacks[i];
            pc.setPassword( password.toCharArray());
            }
        }
     }
}
```

3. ServerAction.java

```
import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.net.ServerSocket;

import java.net.Socket;

import java.security.PrivilegedAction;

import org.ietf.jgss.GSSContext;

import org.ietf.jgss.GSSCredential;

import org.ietf.jgss.GSSManager;

import org.ietf.jgss.GSSName;

import org.ietf.jgss.MessageProp;

import org.ietf.jgss.Oid;


public class ServerAction implements PrivilegedAction{

        DataInputStream inStream;

        DataOutputStream outStream;

        public ServerAction(DataInputStream inStream,DataOutputStream outStream){

                this.inStream=inStream;

                this.outStream=outStream;

        }

        public Object run(){

        try{
                GSSManager manager = GSSManager.getInstance();


        /*
         * Create a GSSContext to receive the incoming request
         * from the client. Use null for the server credentials
         * passed in. This tells the underlying mechanism
         * to use whatever credentials it has available that
         * can be used to accept this connection.
         */


Oid krb5Oid = new Oid("1.2.840.113554.1.2.2");
GSSName serviceName = manager.createName("princ3/admin@LANSLAB.EDU",
GSSName.NT_USER_NAME);
GSSCredential serviceCredentials = manager.createCredential(serviceName,
GSSCredential.INDEFINITE_LIFETIME, krb5Oid, GSSCredential.ACCEPT_ONLY);
GSSContext context = manager.createContext(serviceCredentials);
// Do the context eastablishment loop
byte[] token = null;
        while (!context.isEstablished()) {
          token = new byte[inStream.readInt()];
           System.out.println("Will read input token of size "
           + token.length+ " for processing by acceptSecContext");
           inStream.readFully(token);
           System.out.println("Properties set server 0");
           token = context.acceptSecContext(token, 0, token.length);
```

```
    System.out.println("Properties set server 1");
    // Send a token to the peer if one was generated by
    // acceptSecContext
    if (token != null) {
        System.out.println("Will send token of size "
                            + token.length+ " from acceptSecContext.");
        outStream.writeInt(token.length);
        outStream.write(token);
        outStream.flush();
    }
}

System.out.print("Context Established! ");
System.out.println("Client is " + context.getSrcName());
System.out.println("Server is " + context.getTargName());
/*
 * If mutual authentication did not take place, then
 * only the client was authenticated to the
 * server. Otherwise, both client and server were
 * authenticated to each other.
 */
if (context.getMutualAuthState())
    System.out.println("Mutual authentication took place!");


/*
```

```
 * Create a MessageProp which unwrap will use to return
 * information such as the Quality-of-Protection that was
 * applied to the wrapped token, whether or not it was
 * encrypted, etc. Since the initial MessageProp values
 * are ignored, just set them to the defaults of 0 and false.
 */
MessageProp prop = new MessageProp(0, false);

/*
 * Read the token. This uses the same token byte array
 * as that used during context establishment.
 */
token = new byte[inStream.readInt()];
System.out.println("Will read token of size "
                   + token.length);
inStream.readFully(token);

byte[] bytes = context.unwrap(token, 0, token.length, prop);
String str = new String(bytes);
System.out.println("Received data \""
                   + str + "\" of length " + str.length());

System.out.println("Confidentiality applied: "
                   + prop.getPrivacy());
```

```
/*
 * Now generate a MIC and send it to the client. This is
 * just for illustration purposes. The integrity of the
 * incoming wrapped message is guaranteed irrespective of
 * the confidentiality (encryption) that was used.
 */

/*
 * First reset the QOP of the MessageProp to 0
 * to ensure the default Quality-of-Protection
 * is applied.
 */
prop.setQOP(0);

token = context.getMIC(bytes, 0, bytes.length, prop);

System.out.println("Will send MIC token of size "
                   + token.length);
outStream.writeInt(token.length);
outStream.write(token);
outStream.flush();

context.dispose();
outStream.close();
```

```
        }catch (Exception e){
                        e.printStackTrace();
        }
                return null;
        }
}
```

4. ServerThread.java

```
import java.io.*;
import java.net.Socket;
import java.security.PrivilegedAction;
import java.security.PrivilegedActionException;
import javax.security.auth.Subject;

public class ServerThread extends Thread{
        Socket socket;
        Subject serviceSubject;
         public ServerThread(Socket socket,Subject serviceSubject)
         {
                this.socket=socket;
                this.serviceSubject=serviceSubject;
         }

         public void run()
         {
                try
                {
         describeConnection(socket);
        DataInputStream inStream = new DataInputStream(socket.getInputStream());
        DataOutputStream outStream = new DataOutputStream(socket.getOutputStream());

        PrivilegedAction action = new ServerAction(inStream,outStream);
        Subject.doAs(serviceSubject, action);
         }catch (Exception e){
                            e.printStackTrace();
                }
      try
      {
      socket.close();
      }catch(IOException ex){
                 System.out.println("IOException occurred when closing socket.");
                }
        }

         void describeConnection(Socket client)
         {
                   String destName = client.getInetAddress().getHostName();
                   String destAddr = client.getInetAddress().getHostAddress();
                   int destPort = client.getPort();
```

```
                    System.out.println("Accepted connection to "+destName+" ("+destAddr+")"+"
on port "+destPort+".");

          }

}
```

Appendix 5

Kerberized RMI Clients Source Code:

1. kerbClient.java

```
import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.rmi.NotBoundException;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.security.PrivilegedAction;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Iterator;

import java.util.Map;

import java.util.Map.Entry;

import java.util.StringTokenizer;

import javax.security.auth.Subject;

import javax.security.auth.login.LoginContext;

import javax.security.auth.login.LoginException;
import org.ietf.jgss.GSSException;


public class kerbClient {

        public static StringBuilder sbuilder = new StringBuilder();

        public static SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss:SSS");

        public static String intservaddr = "";//"127.0.0.1";    // IP address of Intermediate
Server

        public static String username = "";             // Username to be authenticate

        public static String passwd = "";               // Password of respective user

        public static Registry intservRegistry;

        public static KDCInterface IServer;

        public static int intServPort = 9916;

        public static String schAuthServ;               //to store scheduled KDC's IP retrieved
from intermediate server

        public static String
startTime,endTime,reqTime,ipRetrieveTime,kdcauthReqTime,kdcauthStartTime,kdcauthCom
pletionTime,appServAuthReqTime,appServAuthStartTime,appServAuthCompletionTime;
        //inTimegetAuthServ;

        public static String totalinterMServTime, totalKDCServTime, totalAppServAuthTime,
totalAuthTime;


        private static Subject subject = new Subject();

        private static LoginContext lc = null;


        public static void Client(String[] args) throws IOException, GSSException{

                startTime=sdf.format(new Date());

                System.out.println("Kerberos Client Program is Starting.......\n");
```

```
			System.out.println("Kerberos Client Init Time is:\t"+startTime+"\n");
			System.out.println("--------------------------------------------------------------------
-------------\n");


			/*Kerberos System Property Settings for Authentication*/
			System.setProperty("sun.security.krb5.debug", "true");
			System.setProperty("java.security.krb5.realm", "LANSLAB.EDU");
			System.setProperty("java.security.auth.login.config", "./jaasc.conf");
			// Parse arguments
    if (args.length!=2 && args.length!=3)
      invalidOptions();
    else
    {
        username = args[0];
        passwd = args[1];
        System.out.println("UserName:\t"+username+"\nPassword:\t"+passwd);
        if (args.length == 3)
       intservaddr = args[2];
    }
    try {
intservRegistry = LocateRegistry.getRegistry(intservaddr,(new
Integer(intServPort)).intValue());
reqTime = sdf.format(new Date());
System.out.println("Sending request to "+intservaddr+" : "+intServPort+"\t at
Time"+reqTime+"\n");
System.out.println("----------------------------------------------------------------------------------
\n");

IServer = (KDCInterface)(intservRegistry.lookup("InterMServer")); /* for*/

System.out.println("Connected to Intermediate Server\t"+intservaddr+"\n");

/*Now get the Scheduled KDC IP i.e. the Retrieve the IP of Authentication Server to be used
with respective Client*/

schAuthServ = IServer.getSchKDCip();//"192.168.50.10";//

ipRetrieveTime = sdf.format(new Date());

System.out.println("Time when Authentication Server IP assigned to Client is :
\t"+ipRetrieveTime+"For"+username+"\n");

totalinterMServTime = timeDiff(reqTime, ipRetrieveTime);

System.out.println("Time Taken by Intermediatery Server to give Scheduled KDC IP
:"+totalinterMServTime+"\n");

if(schAuthServ != null){

System.out.println("IP Address Retrieved is :\t"+schAuthServ+"\n");

System.out.println("----------------------------------------------------------------------------------
\n");

                    }

          else

          {

          System.out.println("******Error/Warning:Load Limit of All KDC's exceeds...Try
Again....\n");

          }

/*Now we have retrieved the scheduled IP lets check the Updated LoadLimit of Each
KDC...*/

Map<String, Integer> KDClist = IServer.getCurrentKDCs();

System.out.println("List of KDCs are:\t");//+KDClist);
System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

System.out.println("|\tSr.No.\t\t|"+"\t\tKDC IP\t\t\t|"+"\tLoad Limit\t|");

System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

Iterator<Entry<String, Integer>> iter = KDClist.entrySet().iterator();

        int j=1;

        while (iter.hasNext()) {

        Entry<String, Integer> next = iter.next();

        Integer value = next.getValue();

        System.out.println("|\t  "+j+"\t\t|\t"+next.getKey()+"\t\t|\t    "+value+"\t\t|\t");

        System.out.println("|----------------------|-------------------------------------|----------------
-------|\t");

        j++;

            }

} catch (RemoteException e) {

        // TODO: handle exception

        e.printStackTrace();

        } catch (NotBoundException e) {

        // TODO Auto-generated catch block

        e.printStackTrace();

}

/*Now we have retrieved the scheduled KDC's IP now the authentication process will start*/

System.setProperty("java.security.krb5.kdc", schAuthServ);

/*Authentication Server is Set for further process...*/
```

```
/*First Step of JAAS Authentication "Create LoginContext with optionally passing
CallbackHandler to LoginContext for gathering or processing authentication data...."*/

        try {

                kdcauthReqTime = sdf.format(new Date());

                System.out.println("\nKDC Authentication Request Time :
"+kdcauthReqTime+"\n");

                lc = new LoginContext("JaasSample", new LoginCallbackHancler(username,
passwd));

        // Authentication successful, we can now continue.

        } catch (LoginException e) {

                // TODO: handle exception

                e.printStackTrace();

                System.exit(-1);

                } catch (SecurityException se) {

                        // TODO: handle exception

                        se.printStackTrace();

                        System.exit(-1);

                        }

        // the user has 3 attempts to authenticate successfully

            int i;

            for (i = 0; i < 3; i++) {

            try {

                 kdcauthStartTime = sdf.format(new Date());

                 System.out.println("KDC Authentication Start Time
:"+kdcauthStartTime+"\n");

                // attempt authentication
```

```
                    lc.login();

                    kdcauthCompletionTime = sdf.format(new Date());

                    System.out.println("\nKDC Authentication Completion
Time:"+kdcauthCompletionTime+"\n");

                    totalKDCServTime = timeDiff(kdcauthReqTime,kdcauthCompletionTime);

                    System.out.println("Time for Authentication by KDC :
"+totalKDCServTime+"\n");

                    break;

        } catch (LoginException le) {

            System.err.println("Authentication failed:");

            System.err.println("  " + le.getMessage());

            try {

                Thread.currentThread().sleep(3000);

            } catch (Exception e) {

                e. printStackTrace();

            }

        }

}

  // did they fail three times?

    if (i == 3) {

    System.out.println("Sorry....\n");

    System.exit(-1);

    }

 System.out.println("Authentication succeeded!\n");

subject = lc.getSubject();
```

```
appServAuthReqTime = sdf.format(new Date());

PrivilegedAction action = new ClientAction(schAuthServ, username);

appServAuthStartTime = sdf.format(new Date());

System.out.println("AppServer Authentication Start Time:"+appServAuthStartTime);

Subject.doAs(subject, action);

appServAuthCompletionTime = sdf.format(new Date());

System.out.println("AppServer Authentication Completion Time
:"+appServAuthCompletionTime);

totalAppServAuthTime = timeDiff(appServAuthReqTime, appServAuthCompletionTime);

System.out.println("Time for Authentication by AppServer :"+totalAppServAuthTime);

endTime=sdf.format(new Date());

totalAuthTime=timeDiff(reqTime,endTime);

logTime(sbuilder,username);

 }

private static void invalidOptions(){

		System.out.println("MyClient <username password> [server]");

		System.exit(0);

}


private static void logTime(StringBuilder sbuild,String user){

sbuilder.append("For User/Client\t"+user+"\n");

sbuilder.append("\n");

sbuilder.append("KDC Name/IP Address Retrieved is :\t"+schAuthServ+"\n");

sbuilder.append("\n");
sbuilder.append("Time Taken by Intermediatery Server to give Scheduled KDC IP
:"+totalinterMServTime+" For "+user+"\n");

sbuilder.append("\n");

sbuilder.append("Time for Authentication by KDC : "+totalKDCServTime+" For
"+user+"\n");

sbuilder.append("\n");

sbuilder.append("Time for Authentication by AppServer :"+totalAppServAuthTime+" For
"+user+"\n");

sbuilder.append("\n");

sbuilder.append("Total time taken is:"+totalAuthTime);

sbuilder.append("\n");

sbuilder.append("*************************************************************
******************************************\n");

sbuilder.append("\n");

Log(sbuilder,username);

}


// Log files generator

static void Log(StringBuilder logData, String fileNameExt) {

        try {

                String user[]=username.split("/");

                String folder="/home/kslave/Desktop/LoadTest";

                String filename=user[0] + ".txt";

                File f = new File(folder,filename);

                if (!f.exists())

                        f.createNewFile();
        String data = new String(logData);

        FileOutputStream fout = new FileOutputStream(f);

        fout.write(data.getBytes());

        fout.flush();

        fout.close();

        } catch (IOException e) {

                // TODO Auto-generated catch block

                e.printStackTrace();

        }

}

static String timeDiff(String time1,String time2)

{

        StringTokenizer tokens1 = new StringTokenizer(time1, ":");

        StringTokenizer tokens2 = new StringTokenizer(time2, ":");

        int hr1 = Integer.parseInt(tokens1.nextToken());

        int min1 = Integer.parseInt(tokens1.nextToken());

        int sec1 = Integer.parseInt(tokens1.nextToken());

        int mili1 = Integer.parseInt(tokens1.nextToken());

        int hr2 = Integer.parseInt(tokens2.nextToken());

        int min2 = Integer.parseInt(tokens2.nextToken());

        int sec2 = Integer.parseInt(tokens2.nextToken());

        int mili2 = Integer.parseInt(tokens2.nextToken());


        int totalMili1 = (hr1 * 60 * 60 * 1000) + (min1 * 60 * 1000)

                        + (sec1 * 1000) + mili1;
        int totalMili2 = (hr2 * 60 * 60 * 1000) + (min2 * 60 * 1000)
                + (sec2 * 1000) + mili2;

        int diff = totalMili2 - totalMili1;

        int diffMili = diff % 1000;
        int diffSec = (diff / 1000) % 60;

        return (diffSec + " s " + diffMili
                + " ms");
    }
}
```

2. KDCInterface.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;
import java.util.Map;

public interface KDCInterface extends Remote {
    void addKDC(String x) throws RemoteException;
    public Map<String, Integer> getCurrentKDCs() throws RemoteException;
    String getMasterKDC() throws RemoteException;
    //This is Experimental with Client
    String getSchKDCip() throws RemoteException;
}
```

3. LoginCallbackHandler.java

```
import java.io.IOException;

import javax.security.auth.callback.Callback;

import javax.security.auth.callback.CallbackHandler;

import javax.security.auth.callback.NameCallback;

import javax.security.auth.callback.PasswordCallback;

import javax.security.auth.callback.UnsupportedCallbackException;


public class LoginCallbackHancler implements CallbackHandler {

        private String username, passwd;


        public LoginCallbackHancler(){

                super();

        }

public  LoginCallbackHancler(String name, String passwd) {

                super();

                this.username = name;

                this.passwd = passwd;

        }

        @Override

        public void handle(Callback[] callbacks) throws IOException,

                                UnsupportedCallbackException {

                // TODO Auto-generated method stub
```

```
                    for ( int i=0; i<callbacks.length; i++) {
                         if ( callbacks[i] instanceof NameCallback && username != null) {
                           NameCallback nc = (NameCallback) callbacks[i];
                           nc.setName( username);
                         }
                         else if ( callbacks[i] instanceof PasswordCallback) {
                           PasswordCallback pc = (PasswordCallback) callbacks[i];
                           pc.setPassword( passwd.toCharArray());
                         }
               }
        }
}
```

4. ClientAction.java

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.PrintWriter;
import java.net.Socket;
import java.security.PrivilegedAction;
import java.util.Date;
import org.ietf.jgss.GSSContext;
import org.ietf.jgss.GSSCredential;
import org.ietf.jgss.GSSManager;
import org.ietf.jgss.GSSName;
import org.ietf.jgss.MessageProp;
import org.ietf.jgss.Oid;


public class ClientAction implements PrivilegedAction{
        String hostName;
        String username;
        int port;
        public ClientAction(String schAuthServ,String username){
        hostName="172.16.6.22";
        port=9990;
        this.username=username;
}
public Object run(){
        try{
     /*
      * This Oid is used to represent the Kerberos version 5 GSS-API
      * mechanism. It is defined in RFC 1964. We will use this Oid
      * whenever we need to indicate to the GSS-API that it must
      * use Kerberos for some purpose.
      */


     Oid krb5Oid = new Oid("1.2.840.113554.1.2.2");
     GSSManager gssManager = GSSManager.getInstance();


     /*
      * Create a GSSName out of the server's name. The null
```

```
         * indicates that this application does not wish toimport java.security.PrivilegedAction;
make

         * any claims about the syntax of this name and that the

         * underlying mechanism should try to parse it as per whatever

         * default syntax it chooses.

         */

                GSSName clientName = gssManager.createName(username,
GSSName.NT_USER_NAME);

                GSSName serviceName =
gssManager.createName("princ3/admin@LANSLAB.EDU", null);


        /*

         * Create a GSSContext for mutual authentication with the

         * server.

         *    - serverName is the GSSName that represents the server.

         *    - krb5Oid is the Oid that represents the mechanism to

         *      use. The client chooses the mechanism to use.

         *    - null is passed in for client credentials

         *    - DEFAULT_LIFETIME lets the mechanism decide how long the

         *      context can remain valid.

         * Note: Passing in null for the credentials asks GSS-API to

         * use the default credentials. This means that the mechanism

         * will look among the credentials stored in the current Subject

         * to find the right kind of credentials that it needs.

         */
```

```
                    GSSCredential clientCredentials = gssManager.createCredential(clientName,
8*60*60, krb5Oid, GSSCredential.INITIATE_ONLY);

                    GSSContext context = gssManager.createContext(serviceName, krb5Oid,
clientCredentials, GSSContext.DEFAULT_LIFETIME);


        // Set the desired optional features on the context. The client

        // chooses these options.


        context.requestMutualAuth(true);  // Mutual authentication

        context.requestConf(true);  // Will use confidentiality later

        context.requestInteg(true); // Will use integrity later


       // String Time4=kerbClient.sdf.format(new Date());

        //String TimeKDC=kerbClient.timeDiff(kerbClient.Time3,Time4);

        //kerbClient.sbuilder.append("Time for KDC: "+TimeKDC+"\n");

        //writer.flush();

                    //writer.write(kerbClient.IP);

                    //writer.flush();

                    //writer.close();

                    //// Do the context eastablishment loop


        byte[] token = new byte[0];


                    //String Time5=kerbClient.sdf.format(new Date());
        Socket socket = new Socket(hostName, port);

        DataInputStream inStream = new DataInputStream(socket.getInputStream());

        DataOutputStream outStream = new DataOutputStream(socket.getOutputStream());



        while (!context.isEstablished()) {


            // token is ignored on the first call

          token = context.initSecContext(token, 0, token.length);

          // Send a token to the server if one was generated by

          // initSecContext

          if (token != null) {

              System.out.println("Will send token of size "+ token.length + " from
initSecContext.");

              outStream.writeInt(token.length);

              outStream.write(token);

              outStream.flush();

          }


          // If the client is done with context establishment

          // then there will be no more tokens to read in this loop

          if (!context.isEstablished()) {

              token = new byte[inStream.readInt()];

              System.out.println("Will read input token of size " + token.length + " for processing
by initSecContext");
```

```
            inStream.readFully(token);
          }
        }

        //String Time6=kerbClient.sdf.format(new Date());

        //String TimeServerAuthentication=kerbClient.timeDiff(Time5, Time6);

        //kerbClient.sbuilder.append("Time for authentication by
server"+TimeServerAuthentication+"\n");

        System.out.println("Context Established! ");

        /*
         * If mutual authentication did not take place, then only the
         * client was authenticated to the server. Otherwise, both
         * client and server were authenticated to each other.
         */
        if (context.getMutualAuthState())
          System.out.println("Mutual authentication took place!");

        byte[] messageBytes = "Hello There!\0".getBytes();

        /*
         * The first MessageProp argument is 0 to request
```

```
 * the default Quality-of-Protection.
 * The second argument is true to request
 * privacy (encryption of the message).
 */
MessageProp prop =  new MessageProp(0, true);

/*
 * Encrypt the data and send it across. Integrity protection
 * is always applied, irrespective of confidentiality
 * (i.e., encryption).
 * You can use the same token (byte array) as that used when
 * establishing the context.
 */

token = context.wrap(messageBytes, 0, messageBytes.length, prop);
System.out.println("Will send wrap token of size " + token.length);
outStream.writeInt(token.length);
outStream.write(token);
outStream.flush();

/*
 * Now we will allow the server to decrypt the message,
 * calculate a MIC on the decrypted message and send it back
 * to us for verification. This is unnecessary, but done here
 * for illustration.
 */


token = new byte[inStream.readInt()];

System.out.println("Will read token of size " + token.length);

inStream.readFully(token);

context.verifyMIC(token, 0, token.length,

                messageBytes, 0, messageBytes.length,

                prop);


System.out.println("Verified received MIC for message.");


System.out.println("Exiting...");

context.dispose();

socket.close();

    }catch(Exception e){


              /*if(!IS.isClosed())

              {

                        writer.write(kerbClient.IP);

                        try{

                        IS.close();

                        }catch(Exception ec)

                        {

                                  ec.printStackTrace();

                        }
                }
                */
                e.printStackTrace();
        }
                return hostName;
    }
}
```

5. KerbMulThreads.java

```
import java.io.IOException;
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.ArrayList;
import org.ietf.jgss.GSSException;


public class KerbMulThreads {
        private static final int MYTHREADS = 1;
        private static String intermURL = "172.16.6.22";
        private static int noOfInstances = 201;
        private static String user;
        private static String pass;
        //private static String p = "p";
        public static void main(String args[]) throws Exception {

                String [] usernames = new String[noOfInstances];

                String [] passwds = new String[noOfInstances];

                ExecutorService executor = Executors.newFixedThreadPool(MYTHREADS);


                for(int i=1;i<noOfInstances;i++){

                        usernames[i]= "p"+Integer.toString(i);

                        System.out.println(usernames[i]);

                        passwds[i]= "p"+Integer.toString(i);

                        System.out.println(passwds[i]);

        System.out.println("########################################################
######################");



                }
List<Object> list= new ArrayList<Object>();//new MyRunnable(usernames, passwds,
intermURL);

for(int i=1;i<noOfInstances;i++){

System.out.println("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@");

System.out.println("Thread :\t"+i+"\t submitted");

MyRunnable obj= new MyRunnable(usernames[i],passwds[i],intermURL);

executor.submit(obj);

list.add(obj);

}

System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
```

```
System.out.println("All Tasks are Submitted.................. ");
executor.shutdown();
// Wait until all threads are finish
while (!executor.isTerminated()) {


    }
    System.out.println("\nFinished all threads");


//kerbClient kerb = new kerbClient();
}
public static class MyRunnable implements Runnable {
        private final String user,pass,intermURL;
        kerbClient kerb = new kerbClient();
        public MyRunnable(String user, String pass, String intermURL) {
                // TODO Auto-generated constructor stub
                this.user = user;
                this.pass = pass;
                this.intermURL = intermURL;


        }
@Override
public void run() {
        // TODO Auto-generated method stub
        String[] args = {user,pass,intermURL};
        try {
```

```
        System.out.println("Executing Thread's Run Body..........................");
        kerb.Client(args);


        } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        } catch (GSSException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
                }
        }
    }
}
```