Chapter 3

PROPOSED PKLK ARCHITECTURE
Chapter 3: PROPOSED PKLK ARCHITECTURE

3. Proposed PKLK Architecture

3.1 Public Key based Location aware Kerberos Protocol

3.2 Public Key and Context based system

3.2.1 Introduction

3.2.2 Software Context

3.2.3 Major Constraints

3.3 Proposed PKLK system

3.3.1 Usage Scenario

3.3.2 System Architecture

3.3.3 Architecture Design

3.4 Protocol Design

3.4.1 Protocol Modifications for Authentication Phase

3.4.2 Protocol Modifications for Authorization Phase

3.5 Mathematical Model
Chapter 3

PROPOSED PKLK ARCHITECTURE

The proposed system is an Authentication and Authorization combination for organizations having employees who work remotely. The system is designed to maximize the benefits of Kerberos for mobile users by providing crucial modifications in the Authorization and Authentication phases of the user. By maximizing the benefits of the Kerberos system it will meet the organizations prime need to have more secure authentication and authorization while maintaining ease of use to the users, and employees. More specifically, the system will be designed to eliminate dependency of authentication from the user password and will allow the employees to avail the benefit of single sign on. The system provides novel authorization capabilities for organizations so that it will support context based authorization to control the user’s access for a particular service based on dynamic parameters like users location.

3.1 Public key based location aware Kerberos Protocol (PKLK)

Authentication: It is the process of verifying someone by their credentials.

- The system shall provide user authentication capability based on Kerberos and Public Key Support.
• The system shall support Single Sign on (SSO) capabilities.

**Authorization:** Functionality of authorization is to find out whether a particular user is authorized to allow for particular operation on object after logging to system or service.

Functional requirements:
• The system shall provide dynamic authorization capability based on the user’s current physical location.
• The system shall process and validate the user’s current physical location to get a service ticket.
• The system shall re-validate a user’s access at the final step by extending the functionality of the application server.

**Auditability:** The log contains the object with its actions that can be used for access permission. Audit trails can be also be used for the same purpose. The relation between entities and users is managed and permissions to access services or resources may or may not be granted. The only requirement is to find out criteria for sufficient information for granting permission. Logs and Audit trials are important for following reasons:

1. Security violation detection
2. Recreation of incidents for finding loopholes

The system should be able to read the KDC Location Service’s log and display all logged-in users in the past 8 hours on a location map.
**Performance Requirements**

- Response time requirement (The time required in getting the response for the authentication request by the server should be least).
- Speed and latency requirements (High speed machines are required for fast retrieval of data).

The Kerberos Client should be able to parse all responses from the KDC server in the least possible time.

### 3.2. Public key and Context based system

In this Research work the objective is to overcome some of the important issues and enrich Kerberos functions by integrating the public key and location based security features. Figure 3.1 shows the motivation behind building the PKLK model, which is an attempt to integrate enhanced authentication and authorization capacities.
3.2.1 Introduction

Kerberos performs authentication by using conventional (shared secret key) cryptography, which has some weaknesses. Authentication is the first phase, which is password dependent. It makes the protocol vulnerable from attackers for password hacking using offline or online attacks. To overcome this risk, public key cryptography is used in the initial authentication of Kerberos. Although PKC strengthens Kerberos in terms of security there are noticeable limitations like computational burden and communication overhead, which impacts KDC performance and network bandwidth.

Authorization which is the second phase of Kerberos has few limitations. Here, it provides a service ticket to all valid users present in the system and expects the application server to take decisions on granting or rejecting user requests in the third phase. Due to this to reject an unauthorized user, the third phase of message passes becomes necessary, which is again an overhead.
to the system. It introduces the risk of replay attack in the third phase where an attacker simply attempts to reuse the request of the third phase to authenticate himself to a server.

To address the above issues we have proposed a password-less secure authentication system based on a public key, which is lighter on computation and network traffic. We have customized the second phase of Kerberos using the user’s context location as parameter.

3.2.2 Software Context

The software system will be an Authentication and Authorization system for organizations having employees that work remotely. The system will be designed to maximize the benefits of Kerberos for mobile users. More specifically, the system will be designed to eliminate dependency of authentication from the user password and will be allowing the employees to avail the benefit of single sign on. The system will provide novel authorization capabilities for organizations so that it will support context based authorization to control the user’s access for a particular service based on dynamic parameters like users location.

3.2.3 Major Constraints

One of the major constraints is to build a customized GPS based context device for the end user. A major objective of the project is to enhance the
Kerberos protocol and we have to keep building the GPS based custom device as out of scope.

**Assumptions**

- It is assumed that the design of the customized GPS system and the associated device is out of scope.
- It is assumed that the context information being read from a user folder is secured.
- It is assumed that the user’s access validation will be performed at city and/or country level.
- The database being used in this system is secured and is less susceptible to crashes.
- It’s assumed that all server components including the Authentication / Authorization server will be running on the same box.

**Dependencies**

- This system is dependent on the network speed and traffic.
- The system is also dependent on well trained employees i.e. people using this system should have adequate computer knowledge.

**3.3 Proposed PKLK system**

Figure 3.2 shows the proposed block diagram of the PKLK system. At a broad level it is divided into two parts:
1) User space - Client Application and
2) Server space.

1) Users space: It consists of a User’s machine, which can be a desktop or laptop. The user also carries a separate utility device like a smart card to keep his Private Key/Public Key separately. The user is provided with a preconfigured Custom GPS Device. This GPS device carries a unique device id, which is pre-registered in the organizational database and is assigned to the user explicitly. This device acts as the user’s context, which shares data during the user authorization phase. It also shares an encrypted timestamp with the client machine during authentication.

Fig. 3.2: Proposed PKLK System Overview Diagram.
2) Server space: It consists of the Authentication Server, and Ticket Granting Server as a part of KDC. KDC uses a customized LDAP database. The database consists of user data and service data. We have introduced new fields into the database like the provision for storing the user’s public key, and user’s device key. Services are also identified as critical or non-critical. A Context Manager (CTXM) component has been introduced as shown in Figure 3.3. CTXM can interpret encrypted context information (e.g. user’s locations, which are produced by the C-GPS device). Upon request from TGS it decrypts and then validates inputs against the Context Database and verifies it with the restriction policy. If the context is found outside the policy or is invalid then it returns the error message to TGS.

### 3.3.1 Usage Scenario

The focus of a usage scenario is from the user’s point of view. The scenario is created using a real world example. The scenario describes organizations and users interaction and semantics of communication between the entities. This includes the information regarding actions events and steps while doing communication. This section focuses on the tasks, which users perform during authentication and authorization in a tabular format.

### 3.3.2 System Architecture
This section provides the complete description and the design for data, Interface and components, architecture, and framework. Software requirement specification for the Research is to be developed as a part of Research project.

### 3.3.3 Architecture Design

PKLK implements extensions to the Kerberos V5 protocol. Figure 3.3 shows a high level architecture diagram of the PKLK System. This architecture is designed to fulfill the core requirements of this Research related to authentication and authorization. Architecture is logically, divided in two parts, User part and Server part. The user side carries the Kerberos Client (KClient), Custom GPS device, and User repository. On the server side, the KDC setup consists of two service components, which are the Authentication Server and the Ticket Granting Server. To support context based authorization a new component is introduced called as Context Manager (CTXM). On the back end there is the LDAP database, and Context database to support Authentication and Authorization.
Fig. 3.3: Proposed PKLK Architecture.


The Apache Directory Kerberos server uses Apache MINA in the networking layer and the Apache Directory as the backend for storing principals and associated keys.

The Kerberos server provides:

- Authentication service
- Ticket-granting service
• Pre-authentication support (PA-ENC-TIMESTAMP)
• UDP and TCP transports

A Realm is associated with a Kerberos administrative domain. In other words, it covers everything the Kerberos server manages.

The Kerberos Principal is any entity to which the server can assign a Ticket. Typically, we can think of three kinds of Principals:

• Users
• Services
• Hosts

Each Principal is unique in the Kerberos database. This is the way we identify the entity.

A Kerberos Principal is a combination of three parts:

• The name (the primary)
• An optional instance
• The realm they are associated with

**Keys:**

The Kerberos server generates keys based on the password we provide. Those keys are stored in the server and are used for encryption and decryption of the data being exchanged with the client.
**KDC (Key Distribution Center)**

The KDC contains three components: an Authentication Service, a Ticket Granting Service, and a database (ApacheDS).

The role of KDC is to authenticate users and distribute tickets based on the information stored in its database.

The Apache Kerberos Server contains all these three components and hence is a KDC.

**Database**

This is the place where all the private keys are stored. It is very common to store all the keys in an LDAP server, and even more natural when the Kerberos server is a part of an existing LDAP server, like the Apache Directory Server.
Data flow architecture is shown in Figure 3.4. Data is transmitted between the client and the server in three phases. When the Apache Directory Server was started, it was also thought of as a repository for Kerberos keys, so we just had to develop the logic to manage those keys, and the Kerberos protocol. In other words, you have everything embedded in a single server: The LDAP server to store the keys and other related information, The Kerberos protocol, the Authentication Server, and the Ticket Granting Server.
1) **Authentication Exchange:**

During the Authentication exchange phase, the client sends AS_REQ to the Authentication Server of KDC. Once authenticated successfully, it returns the Ticket Granting Ticket in reply (AS_REP). The Authentication Server interacts with the LDAP database to validate user attributes.

2) **TGS Exchange:**

When the client needs access to a particular service it sends a service request in TGS_REQ to the Ticket Granting Server. This server interacts with the LDAP and Context Database to validate the authorization related parameters. Once authorized successfully, the TGS responds back with TGS_REP.

3) **Application Exchange:**

This is the last step of data exchange. This step is taken outside KDC where implementation is left to the target Application Server. Hence, this step is not considered during implementation. Figure 3.4 includes this step for the completeness of data flow.

**Kerberos with Public key**

Public Key Initial Authentication (PKINIT) is an extension to the original Kerberos 5 specification that provides a public-key-based replacement for a
client's initial authentication to a KDC's Authentication Service. In this modified version, the end user possesses a public-private key pair, the public component of which is pre-installed in the KDC's database. PKINIT extends the Kerberos protocol by using the pre-authentication field to house a public key certificate. PKINIT Initial Authentication Steps are mentioned below:

- $PU_c$ = Public Key of the Client
- $PR_c$ = Private Key of the Client
- $PR_{kdc}$ = Private Key of KDC
- $RK_c$ = Random Key
- $ID_c$ = ID of the Client
- $ID_{tgs}$ = ID of the Ticket Granting Server
- $Cert_c$ = Certificate of the Client
- $Cert_{kdc}$ = Certificate of the Kerberos server
- $K_{c,tgs}$ = Session Key for Client and TGS
- $TS$ = Time Stamp
- $TGT$ = Ticket Granting Ticket

**AS Request**: Client sends the Public Key Certificate $Cert_c$ and Digital Signature $[TS_c, n1]_{PR_c}$ to the Server.

$$[Cert_c, [TS_c, n1]_{PR_c}] || ID_c || ID_{tgs} || TS_1$$

**AS Response**: KDC verifies the client’s public certificate and signature. If the request is valid then the KDC generates the random key RK encrypted with its
own private key PR\textsubscript{kdc}. It uses the clients public key PU\textsubscript{c} to further encrypt the KDC certificate and the encrypted RK.

\[
[\{\text{Cert}_{kdc}, [\text{RK}, n2]_{PR_{kdc}PU_{c}}\}, \text{E}(\text{RK}, [\text{K}_{c,tgs} \mid \text{ID}_{tgs} \mid \text{TS}2 \mid \text{Lifetime2} \mid \text{TGT}_{tgs})\}]
\]

### 3.4 Protocol Design

#### 3.4.1 Protocol Modifications for Authentication Phase

In the traditional Kerberos V5 during logon, the server really does not care about the user’s identity and returns a strongly encrypted ticket, but as the encryption is finally linked with the user’s password there is a fair possibility that an attacker can perform offline attacks. To eliminate the risk associated with password hacking we have designed a new authentication phase, which is lighter on computation than PKI, but it also satisfies the safety and security needs of the Authentication Phase.

In PKLK, during the authentication phase the client C creates an authentication request called as AS\_REQ. As the device is connected with the user’s machine it encrypts a timestamp (TS) using its own device key (DV\textsubscript{c}). This AS\_REQ is then sent to the Authentication Server (AS). Upon receiving the request, the AS checks if the encrypted timestamp TS gets decrypted with the users device key (DV\textsubscript{c}) from the server’s database. If the decryption of TS is successful then the AS generates an authentication reply (AS\_REP). This reply consists of 3 important elements, which are ticket granting ticket (TGT),
Random Key, (RK) and Session Key (Kc, tgs). Here, Rk is encrypted by the AS using the client’s public key PUc.

DVc = User Device Key
PUc = Public Key of the Client
PRc = Private Key of the Client
RKc = Random Key
IDc = ID of the Client
IDtgs = ID of the Ticket Granting Server
IDv = ID of the Application Server
Kc,tgs = Session Key for Client and TGS
Kc,v = Session Key for Client and the Application Server
TS = Time Stamp
AUTHc = Authenticator
TGT = Ticket Granting Ticket
SGT = Service Granting Ticket

Client sends AS_REQ to Authentication Server

C→AS: [TS]DVc, IDc, IDtgs

Authentication Server sends AS_REP to Client

AS→C: {{[RK]PRkd}PUc, IDc, TGT, { Kc,tgs, IDtgs, TS } RK

\[ TGT=\{Kc,tgs, IDc, IDtgs, TS\}^Ktgs \]
Algorithm

For Authentication Phase

1. User Log on with Public Key
   - User starts the KClient application from his client machine.
   - System displays a login screen
   - User enters his username (principal name)
   - System performs a basic validation on the user name
   - System checks if the employee supports password-less authentication.
   - System asks for the user smart card PIN
   - System responds with “Retrieve user keys” function and passes the user principal and PIN to KClient application
   - “Retrieve user keys” function returns the public key, private key, and the device key to the KClient System
   - System responds with following sequence
     - Step 1: Invoke “Build Authentication Request”. If this step is successful then the system will invoke step 2 mentioned below.
     - Step 2: “Validate Authentication Request”. Upon successful authentication it will invoke the step 3
       - Step 3: “Build Authentication Response”.
   - If Step 3 is successful, the system collects the TGT and the session key and stores it for future use.
   - System displays the “authentication successful” message on to the screen.
2. Retrieve User Keys

- User sends requests for collecting the user secret and public key.
- System displays a PIN box to the user.
- User enters his PIN.
- System validates the user PIN.
- System retrieves the user public key, private key, and device key.
- System returns both keys to KClinet.

3. Build Authentication Request

- User invokes a request for generating an authentication request.
- System reads all the input parameters from the Client as mentioned below.
  ✓ User id (principal name)
  ✓ User public key
  ✓ KDC port, server name
- System generates pre-authentication data as listed below.
  ✓ Encrypted Timestamp using a device key
  ✓ PA Data headers as per Kerberos RFC
  ✓ Nonce
- System encodes all the required parameters in to AS_REQ as defined in Kerberos RFC.
- System returns AS_REQ to the caller.

4. Validate Authentication Request
• User invokes the KClient system.

• System reads request AS_REQ received from the KClient.

• System validates the static inputs received in AS_REQ with the one configured in the LDAP database.
  • User id (principal name) - Lookup
  • User device key  (Compare with DB - True/False)
  • KDC port, server name (Compare with DB – True / False)
  • System then tries to decrypt pre-authentication data received in AS_REQ
    • Encrypted Timestamp – Decrypt using device key (True/False)
    • PA Data headers as per Kerberos RFC – As per Kerberos grammar check – True/False

• If the above checks are successful, the system identifies the user as a valid user.

• System calls “Build Authentication Response”.

• System returns AS_REP to the caller.

5. Build Authentication Response

• System creates a ticket granting ticket - TGT

• System generates a unique random session key.

• System generates a reply AS_REP and embeds TGT, and the session key in it.

• System returns AS_REP to the requesting service.
Challenges with Standard PKI and proposed modifications:

Standard PKINIT relies on client certificate, server certificate, digital signature, and message digest. Having digital certificates in a request requires additional parameters like certificate authority, certificate lifetime, certificate directory, and revocation list. Both the client and the KDC have a public-private key pair in order to prove their identities to each other over the open network [38]. The intention of PKI infrastructure is to address issues of managing secret keys of a large number of clients and a secure authentication procedure by building a new trust model where KDC is not the first entity to identify the user. However, for scenarios where the revocation and suspension of certificates happens, the legislation of digital signatures and related PKIs becomes a difficult and challenging task [111].

We have proposed a thin PKI model to address some of the issues related to Standard PKI. We have reduced the number of pre-authentication data elements used during AS_REQ. Thus, it keeps the message size between a request and response as minimum as possible. PKINIT expects a dataset consisting of Auth-Pack, and trusted certificates. We have not used these parameters as our framework assumes that the client and server are carrying a copy of each other’s certificate, which can be referred on demand from their database. We have eliminated computational overheads like certificate verification by both the client and server. We propose maintenance of keys, certificate and updates in LDAP as an offline activity outside the authentication phase. Table 3.1 shows the comparison between PKINIT and our proposed
PKLK.
Table 3.1: Authentication Phase Comparison between PKINIT and PKCA-

Kerberos

<table>
<thead>
<tr>
<th>Parameter</th>
<th>PKINIT</th>
<th>PKLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys Generation</td>
<td>Keys need to be generated by a certificate authority (CA) and may involve a cost behind each key-pair and certificate generation.</td>
<td>Keys can be generated by either a certificate authority (CA) or by using standard utilities, which come with Java/JDK to save the cost.</td>
</tr>
<tr>
<td>Privacy and Secrecy of user keys</td>
<td>Easy to trust, a secured and proven model in the market. E.g. VeriSign, and Entrust.</td>
<td>Either organization can invest on maintaining a proprietary setup for key generation thereby avoiding handling of key data outside an organization’s private network.</td>
</tr>
<tr>
<td>Dispatch and deployment of Private/Public Keys to the User Device</td>
<td>It introduces handling and physical transfer of keys data once generated by a vendor to the organization and then to the device. More handling adds the possibility of cloning of a private key outside the organizations reach.</td>
<td>Handling of keys limits within the organization boundary. So, a secure deployment of keys to a context device is possible to achieve with a lesser cost on processes, software’s, and people.</td>
</tr>
<tr>
<td>Publishing and Revocation of Certificates</td>
<td>Publish period for updates or revocation is usually in a weekly cycle. This means Certificate Services and the organization system may go out of sync, which may give the opportunity for a disabled user to do an intrusion within a stipulated time (i.e. time between enablement and disablement).</td>
<td>Having an LDAP database and keeping a separate attribute to track the user’s active or inactive status eliminates dependency on Certificate Service for the validity of users. This means the user’s access and authorization can be disabled or enabled instantly.</td>
</tr>
<tr>
<td>Certificate Transfer during AS Exchange</td>
<td>Copies of the User Certificate and the KDC Certificate are transferred to and fro during authentication. If an organization has 10,000+ users and assuming a user logs in two times in a day: User certificates transmits for AS exchange = 20000. KDC certificate transmits for AS exchange = 20000.</td>
<td>Certificates do not get transmitted during Kerberos request or response. The system relies on preinstalled certificates in the database and client’s device. Any updates to certificates, and keys can be done offline as and when needed. This saves the network bandwidth used for Authentication across users.</td>
</tr>
</tbody>
</table>
It means there is a transfer of 40000 certificates over wire in a day.
This definitely acts as an overhead to the system especially, when the organization has a large employee population.

| Certificates Use | Certificate management in traditional public key infrastructure (PKI) is inefficient. | No certificates used in AS_REQ/AS_REP. Hence,  
• Low communication bandwidth.  
• No need to verify certificates online (certificate chains). |

We also verified our new framework design to see if it will be able to sustain popular attacks. While designing the new framework we tried to improve the protocol such that attacks impacting the Kerberos V5 protocol can be either minimized or eliminated.

Kerberos is vulnerable to password guessing attacks. In case if the user chooses a password, which is poor then for an attacker it is easy to attack. He
can easily mount an offline dictionary attack where he can pick up entries from the dictionary and repeatedly, attempt to decrypt the messages obtained during hacking. As the message is originally encrypted using a key derived from the password it is probably easy for the hacker to finally, get the key through trial and error. In case of a brute force attack the attacker uses an exhaustive procedure and tries all the possibilities one-by-one to crack the user’s password.

Table 3.2: Comparison of Possible Attacks.

<table>
<thead>
<tr>
<th>Possible Attacks</th>
<th>PKLK</th>
<th>Kerberos V5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password guessing</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Dictionary Attacks</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Brute Force Attack</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Reply Attack</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

#Notations: ✓ non-vulnerable; ✗ vulnerable

Password guessing and brute force attack are totally eliminated as in our implemented approach there is no need of the user’s password for authentication, which has been replaced by public key cryptography. Table 3.2 shows the comparison of possible attacks on Kerberos V5 and Proposed PKLK framework. Comparison shows that Kerberos is vulnerable to various attacks like password guessing, dictionary attacks, brute force attack and reply attack.
while all these attacks can be defended by PKLK system.

### 3.4.2 Protocol Modifications for Authorization Phase

We have modified the user authorization phase into two parts. The first part is called as the context generation phase. This phase is to generate the user’s context using Context Device Custom GPS. This context data is then embedded into TGS_REQ by the user’s Kerberos client and is sent to the server for authorization verification.

- **User Context Generation Phase:** Figure 3.5 shows a message exchange for the user context generation phase between the user machine and the CGPS device.

![Fig. 3.5: User Context Generation Phase](image)

Upon the user’s request, the Kerberos client process sends a notification to the Context Device CGPS requesting a new user context using the USER_CONTEXT_REQ message. CGPS then calls the GPS service and collects
the GPS location. To make this phase secure we assume that the CGPS device has the capability to encrypt the location and current timestamp using the device key, which is unique. This data is collected in USER_CONTEXT_REP. This data is then shared with the user’s Kerberos client.

• New context based authorization phase: Upon receiving encrypted USER_CONTEXT_REP from the CGPS device, the Kerberos client then generates a service request called as TGS_REQ as shown in Figure 3.6.

![Fig. 3.6: Information Exchange during Service ticket Granting](image)

This request mainly consists of the application server’s ID (IDv), TGT, and an Authenticator. As an authorization request needs dynamic data from the user, we need to share USER_CONTEXT_REP in TGS_REQ. As per Kerberos, RFC 4120 has provision in an authenticator to pass additional authorization data. USER_CONTEXT_REP is added to the authenticator field of
TGS_REQ, which gets encrypted using a shared session key (Kc, tgs). At the server end, TGS revives the user request. It performs the basic validations and it then checks for the presence of context data in the Authenticator field. It decrypts the authenticator field using (Kc, tgs) and retrieves USER_CONTEXT_REP. Upon successful decryption, the TGS server then sends the user principal, target server name, and USER_CONTEXT_REP to the Context Manager CTXM.

The main role of the CTXM manager is to decrypt USER_CONTEXT_REP using a secret key from the Context DB. If decryption is successful, it then checks if the context is within allowed limits as per Organization Policy Rules. If this step is successful, then the CTXM returns the success message to TGS. The TGS in turn generates response TGS_REP and returns it back to the Client.

- Client sends TGS_REQ to the Ticket Granting Server
- C→TGS: ID_v, TGT, AUTH_C

- The Ticket Granting Server sends TGS_REP to the Client
- TGS→C: ID_c, SGT, {Kc,v, ID_v, TS }_{Kc,tgs}
  - SGT={ Kc,v, ID_c, ID_v, TS }_{Kv}
  - AUTH_C={ID_c,TS,CGPS}_{Kc,tgs}

**Overcoming limitations in Traditional Kerberos Authorization Phase**
The Traditional Kerberos authorization phase takes minimal part in authorization. If the user sends a valid SGT_REQ then the KDC generates a service ticket with the SGT and returns it back to the user. Here, it is expected that the target Application Server is capable of taking authorization related validations.

In case an organization wants to tightly control access for critical services using the traditional Kerberos system, then the key authorization checks need to be applied on the Application Servers. But the Application Server comes into action only in the last phase of Kerberos. This means that if the organization has ‘n’ critical services then all of these services will be deploying authorization checks at their end. Having multiple servers involved makes it difficult in maintaining an authorization policy for critical services of the organization.

In this PKLK we have overcome the above problem by enhancing phase-II of Kerberos. Here, the Ticket Granting Server and the Context-Manager works together to validate the user’s context, and performs additional checks like verifying if the user’s context is within a restricted zone. This makes the 2\textsuperscript{nd} phase stronger and addresses the organization’s key need to centralize important authorization decisions. If the user is unable to prove its context, TGS will not provide a service ticket SGT. Not having the SGT in hand makes it impossible for the user to access the target service because his request will be immediately rejected in the 3rd phase of Kerberos.
In traditional Kerberos as TGS returns the SGT to all legitimate users, it increases the possibility of ‘Replay’ attack in the 3\textsuperscript{rd} phase. However, in our proposed framework, the SGT is not returned to the user unless he passes through a critical authorization validation. Not receiving a SGT makes it difficult for the hacker to do a replay attack in the 3\textsuperscript{rd} phase of Kerberos.

**Authorization Phase algorithm**

1. **Request for Service Ticket**
   - The user selects “request for service ticket” option of the Client application.
   - System displays a list of services to the user.
     - TELNET (S1)
     - FTP (S2)
     - HTTP (S3)
   - User selects TELNET service
   - System detects it as a critical service
   - System requests user permission to connect to the context device.
   - User confirms by clicking “OK” button
   - System invokes “Obtain Device Location”.
   - System receives an encrypted context E(CLC).
   - System calls “Build Authorization Request” and provides E(CLC).
   - Above step gives TGS_REQ to the system.
   - Client sends the TGS_REQ to the Authorization Server by invoking “Validates Authorization Request”.
   - Client receives the TGS_REP from the Authorization Server.
• Client decodes TGS_REP and stores the session key and SGT for use in the next phase.
• Client displays a service request of a successful message to the user.

2. Obtain Device Location
• This starts from “Request for service ticket” when the Client sends a request to the User’s GPS Device.
• System (Context device) fetches the current location context (CLC) data from the A-GPS or GPS mechanism (whichever is available and configured).
• System also detects a timestamp when the user’s location was fetched.
• System then encrypts the CLC data and the current timestamp using a device key into a new value called as user’s encrypted context.
• System then shares this encrypted context E(CLC) back to the calling Client.

3. Build Authorization Request
• This process starts from “Request for Service Ticket” function when the Client wants to build TGS_REQ.
• System shares the below key inputs for generating TGS_REQ
  ▪ User’s current context E(CLC)
  ▪ Target Service Name
  ▪ Nonce
  ▪ TGT
  ▪ Session key
• System generates the PA-Data request parameters as specified in Kerberos RFC
• System builds a complete TGS_REQ and returns it to the calling service.

4. Validate Authorization Request
• This function starts when the client calls “Request for Service Ticket” and shares TGS_REQ for an authorization check.
• System validates the static inputs received in TGS_REQ with the LDAP database.
  ▪ User id (principal name) - Lookup
  ▪ Target service name (Compare with DB – True / False)
  ▪ TGT (decrypt using TGS private key – success / fail)
• System then tries to decrypt PA-data received in TGS_REQ
  ▪ Encrypted Timestamp – Decrypt using session key (True/False)
  ▪ PA Data headers as per Kerberos RFC – As per Kerberos grammar check – True/False
• If the above checks are successful, the system identifies the user as a valid user requesting a service ticket.
• System detects E(CLC) in the PA-Data field.
• System calls for “Validate user context”.
• System receives a success message from the above service.
• System calls “Build Authorization Response”.
5. **Validate User Context**

- This steps starts when the Authentication Server (TGS) requests for validating the user’s context.
- TGS shares E(CLC) and the user’s principle information to the Context Manager (CTXM).
- Context Manager (CTXM) fetches a device key from the LDAP database for a given user.
- Context Manager decrypts D(CLC) using a device key and reads CLC successfully.
- Context Manager compares the CLC time with the system time to see if the delay is within the accepted time limit (time difference should be less than 40 seconds).
- Context Manager invokes Dynamic Policy Checker Component PCC.
- Policy Checker Component PCC checks CLC against the given service S1 and verifies if this CLC is within allowed metrics.
- PCC finds CLC within acceptable limits of the organization.
- PCC returns the success flag to the Context Manager.

6. **Build Authorization Response**

- This starts when “Validate Authorization Request” invokes.
- System creates a service granting ticket - SGT
- System generates a unique random session key.
- System generates a reply TGS_REP and embeds SGT, and a session key in
to it.

• System returns TGS_REP to the requesting service.

7. **Create Service Request with User Context**

• Client presents a list of available services to the user as mentioned below.
  • TELNET (S1)
  • FTP (S2)
  • HTTP (S3)
• User selects TELNET service
• System detects it as a critical service
• System asks user to connect the context device
• User connects the context device
• Context device fetches the current location context (CLC) data from the A-GPS or GPS mechanism (whichever is available and configured).
• The CLC data is then encrypted using a context key in the context device and is called as the user’s current context UCC.
• Context device shares this encrypted context E(CLC) back to the KClient.
• KClient encodes the user’s current context E(CLC) using Kerberos’s standard encoding practice into the PA-Data field of TGS_REQ
• KClient builds a complete TGS_REQ and sends it to the Ticket Granting Server (TGS).
• Ticket Granting Server performs the basic validations on user inputs
• TGS detects E(CLC) in the PA-Data field.
• TGS shares E(CLC) and the user’s principle information to CTXM
• CTXM fetches a device key from the LDAP database.
• Context Manager decrypts D(CLC) using a device key and reads CLC successfully.
• Context Manager compares the CLC time with the system time to see if the delay is within the accepted time limit (time difference should be less than 40 seconds).
• Context Manager invokes Dynamic Policy Checker Component PCC.
• Policy Checker Component PCC checks CLC against the given service S1 and verifies if this CLC is within allowed metrics.
• PCC finds CLC within acceptable limits of the organization.
• PCC returns the success flag to the Context Manager.
• Context Manager returns the success message back to the TGS server.
• TGS server builds a TGS_REP and sends it to the KClient.
• KClient decodes TGS_REP and stores the session key and SGT for use in the next phase.

**Comparison between proposed PKLK and PKINIT based authorization systems.**

This research developed a framework to address some of the limitations in the Kerberos protocol. In PKINIT Authorization is distributed across the application server whereas in PKLK it is centralized to the KDC. Dynamic authorization capability is not provided in PKINIT. We are capturing users run
time context parameter which is dynamic in nature and is used for dynamic authorization. PKINI implementation is heavy-weighted in terms of computation and may be burden to the system sometimes but results shows that PKLK is not that heavy-weighted as expected. Dependency on application server is also removed in PKLK.

Table 3.3 presents a detailed comparison between standard PKINIT and PKLK framework against various parameters.

**Table 3.3: Comparison of Authorization Phase between PKINIT and PKCA-Kerberos**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>PKINIT</th>
<th>PKLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization approach</td>
<td>Distributed across Application Servers.</td>
<td>Centralized in to the KDC Domain.</td>
</tr>
<tr>
<td>Supported functions with TGS Server</td>
<td>Basic only. TGS grants SGT to all legitimate users.</td>
<td>Enhanced. TGS performs detailed authorization checks.</td>
</tr>
<tr>
<td>Dynamic authorization capabilities</td>
<td>No</td>
<td>Yes – As system captures the user’s run time context securely.</td>
</tr>
<tr>
<td>Dependency on Application Server during Authorization</td>
<td>Necessary. It’s expected that the Application Server handles the required authorization checks in phase III.</td>
<td>Independent of Application Server. TGS and the Context Manager together can restrict users from granting service</td>
</tr>
<tr>
<td>Implementing new authorization policies for critical services</td>
<td>Costlier and time consuming as it is decentralized. It impacts all critical services. All services need to be modified and tested to fulfill new authorization requirements.</td>
<td>It’s less costly comparatively. This is because many authorization decisions can be centrally managed at TGS and the Context Manager irrespective of the number of critical services present in the network.</td>
</tr>
</tbody>
</table>

Replay attacks are usually, very dangerous as the attacker gains direct access to resources without tampering the data. This means that to safeguard critical services, it is important to protect the information from middle men and hackers. Kerberos V5 cannot prevent a replay attack in the 3rd phase because in the 3rd phase only the client and target service or the target application server comes in to the picture. The client holds SGT and here the hacker can replay a request and possibly get access to the critical service. For organizations this becomes a critical problem.
In our implemented approach we have introduced context based authorization, which is a check on the user’s context (like location) and accordingly, a decision is taken if the SGT should be returned to the user or not. As the user will not be able to acquire SGT in the second phase it stops an unauthorized user to avail SGT. Not having SGT eliminates the opportunity of unauthorized users accessing the target application server.

Attack scenario at server side: Single point of failure - It requires continuous availability of a central server. In our approach we have used Kerberos DB for storing the user’s public key, which is required for verification of the user’s identity. When the Kerberos server is down, no one can log in. Mirroring of server can be done to address this issue.

3.5 Mathematical model

In the proposed system, Users are the key elements. The objective of this research is to provide authentication of users using public key based Kerberos. The proposed system has following assumptions:

- A fixed size key is used for encryption
- KDC is online all the time
- No limit on the number of users arriving in the system

In the sequel, when any number of users arrives to be authenticated using proposed PKLK system, time required for a single user authentication is constant. In order to analyze the performance of PKLK system queuing model happens to be an appropriate mathematical tool. There are multiple queue
models like M/M/1, M/D/1 and M/M/n for the proposed system. Encryption time by device key is constant and M/D/1 is used as the basic queue model. High level queue model of PKLK is depicted in Figure 3.7. As per the basics of queuing theory time span is divided into two primitives that is service time and request time.

The proposed PKLK model consists of a trusted KDC server, which is responsible for distributing a ticket and granting a ticket during authentication and a service ticket during authorization to the end user. Clients approaching the KDC for service are managed in a queue. Figure 3.7 shows the system, where $\lambda$ is the arrival rate of requests.

The inter-arrival time for client requests is exponentially distributed. Thus, arrival rate follows the person’s arrival process. Our system can be modeled as a queuing model with a constant service rate and one server. To evaluate the system performance, we model the sojourn time, that is, the total time spent in the transaction by the client in the system. The expectation of waiting time for authentication requests in the queue can be given as,
\[ E[W_q] = N_q \times E[S] + E[R] \]

Where \( N_q \) = mean number of authentication requests in the queue.
\( E[S] \) = service time of KDC
\( E[R] \) = residual time

Thus, by Little’s formula, mean queue length is given by,

\[ N_q = \lambda \cdot E[W_q] \]

Therefore,

\[ E[W_q] = \frac{E[R]}{1 - \rho_{KDC}} \]

Where utilization of KDC is,

\[ \rho_{KDC} = \lambda \cdot E[S] \]

The residual time, is the service time remaining to the end user being served when the authentication request arrives at the queue. The mean residual time can be calculated as below:

\[ E[R] = \frac{1}{t} \int_0^t R(t) \, dt = \frac{1}{t} \sum_{i=1}^n \frac{1}{2} [S_i^2] \]

\[ = \frac{n}{t} \cdot \frac{1}{n} \sum_{i=1}^n \frac{1}{2} [S_i^2] \]

\[ \frac{n}{t} \to \lambda \sum_{i=1}^n \frac{1}{2} [S_i^2] \to \frac{1}{2} E[S^2] \]

\[ E[R] = \frac{\lambda \cdot E[S^2]}{2} \]

\[ E[W_q] = \frac{\lambda \cdot E[S^2]}{2(1 - \rho_{KDC})} \]
Now, the total time spent for authentication in the system (sojourn time) is

\[ E[T] = E[W_q] + E[S] \]

\[ E[T] = \frac{\lambda \cdot E[S^2]}{2(1 - \rho_{KDC})} + E[S] \]

The total service time comprises of two factors, expectation and variance. The variance is the difference between the mean of squares of the values and square of mean of values. Therefore,

\[ \text{Var}[S] = E[S^2] - E[S]^2 \]

For M/D/1 system, as the service time is constant variance.

\[ \therefore E[S^2] = E[S]^2 \]

Thus,

\[ E[T] = \frac{\lambda \cdot E[S^2]}{2(1 - \rho_{KDC})} + E[S] \]

\[ E[T] = \left(1 + \frac{\rho_{KDC}}{2(1 - \rho_{KDC})}\right) \cdot E[S] \]

By Little's formula the mean queue length, and mean number of authentication requests in a queue is given by,

\[ N_q = \lambda \cdot E[W_q] \]

\[ N_q = \frac{\lambda^2 \cdot E[S]^2}{2(1 - \rho_{KDC})} \]

\[ N_q = \frac{\rho_{KDC}^2}{2(1 - \rho_{KDC})} \]
Thus, from the above equations it can be concluded that the total time spent for a user’s authentication request in the system is the function of the service time and the utilization of KDC.