Unit-3

Q11. Authentication in distributed system

- A distributed system is susceptible to a variety of threats by intruders as well as legitimate users of the system
- The entities in a distributed system, such as users, clients, etc. are referred to as principals.
- A principal can impersonate another principal and authentication becomes an important requirement.
- Authentication is the process by which one principal verifies the identity of another principal. (that the principal's identity is as claime
- In one-way authentication → Only one principal verifies the identity of the other principal.
- In mutual authentication → both communicating principals verify eachother's identity.
- Authentication is based on the possession of some secret information that is known only to the entities participating in the authentication.
- When an entity wants to authenticate another entity, the former will verify if the latter possesses the knowledge of the secret.

- In distributed systems, authentication is carried out using a protocol involving message exchanges
These protocols are called authentication protocols.

- Authentication = Identification + verification

  Identification → procedure by which an entity claims a certain identity

  verification → procedure by which that claim is checked.

- 3 main types of authentication →

→ message content authentication:
verifying that the content of a message recieved is the same as when it was sent

→ message origin authentication:
verifying that the sender of a message recieved is the same as the one recorded in the sender field of the message.

→ general identity authentication:
verifying that the principal's identity is as claimed.

Message content authentication is commonly handled by tagging a key dependant "message authentication code" (MAC) onto the message before it is sent. Message integrity can be confirmed on recieving the message by recomputing the MAC and comparing it with the one attached.

General identity authentication (GIA) is required for both authorization and accounting functions.

Message origin authentication is a subcase of GIA.
A successful general identity authentication enables the authenticating principal (verifier) to believe that the authenticated principal (claimant) possesses the claimed identity.

Types of Principals →

- Hosts : These are the addressable network level entities. They can be identified by name or by network address (eg. IP address)

- Users : These entities are ultimately responsible for all system activities.

- Processes : The system creates processes within the system boundary to represent users. The processes request and consume resources on behalf of the users.

A simple classification of authentication protocols →

Authentication protocols can be categorized based on the type of →

- cryptography → symmetric
  → asymmetric
- reciprocity of authentication → one-way
  → mutual
- key exchange
- real-time involvement of a third party → online
  → offline
- Nature of trust required from third party
- Nature of security guarantees and storage of secrets.

Symmetric cryptography is also known as "private key cryptography" as it uses a single private key to both encrypt and decrypt messages and any party that has the key can use it to encrypt / decrypt data.

Asymmetric cryptography is also known as "public key cryptography". It uses a secret key that must be kept from unauthorized users and a public key that is made public. Both the private and public keys are mathematically linked and both are unique to a communication session.

Notations used →

In symmetric key cryptography-
X_k denotes encryption of X using a symmetric key k and $Y_{k^{-1}}$ denotes the decryption of Y using symmetric key k.

In asymmetric key cryptography-
For a principal x, K_x and K^{-1}_x denote the public and private keys respectively.

A communication step of P is written as P: ...., where "..."
A communication step whereby P sends a message to Q is written as P → Q

A Login Protocol →

A login protocol is shown in the following algorithm-

U → H : U
H → U : "please enter password"
U → H : p
H : compute $y = f(p)$
: Retrieve user record $(U, f(password_U))$ from the database
: If $y = f(password_U)$, then accept, otherwise reject.

Where H = Host, U = User and f is a one-way function

Unit - 3

Q.5 Symmetric Asymmetric Cryptosystems

★ Protocols Based upon Symmetric Cryptosystems →

- In symmetric cryptosystem, knowing the shared key lets ~~a princip~~ a principal encrypt and decrypt arbitrary messages

- Hence authentication protocols can be designed according to the following principle called SVM :

"If a principal can correctly encrypt a message using a key that the verifier believes is known only to the principal with the claimed identity (outside of the verifier), this act constitutes sufficient proof of identity"

Basic protocol →

Principal P is authenticating itself to Principal Q. "k" denotes a secret key that is shared between only P and Q.

P : Create a message m = "I am P."
: Compute $m' = \{m, Q\}_k$

P → Q : m, m'
Q : Verify $\{m, Q\}_k = m'$
: if equal then accept, otherwise authentication fails.

One of the major weakness of basic protocol is it's vulnerability to replays.

Modified protocol with nonce →

To prevent replay attacks, we modify the protocol by adding a challenge and response step called nonce

$P \rightarrow Q$: "I am P"

Q : generate nonce n

$Q \rightarrow P$: n

P : Compute $m' = \{P, Q, n\}_k$

$P \rightarrow Q$: m'

Q : verify $\{P, Q, n\}_k = m'$

: if equal then accept, otherwise the authentication fails.

Wide Mouth frog protocol →

principal

A authenticates itself to principal B using a server S as follows

$A \rightarrow S$: $A, \{T_A, K_{AB}, B\}_{KAS}$

$S \rightarrow B$: $\{T_S, K_{AB}, A\}_{K_{BS}}$

★ Protocols based on Asymmetric Cryptosystem →

- In an ~~asy~~ asymmetric cryptosystem, each principal P publishes it's public key k_p and keeps private it's private key k_p^{-1}

- Only P can generate $\{m\}_{k_p^{-1}}$ for any message m by signing it using k_p^{-1}.
  The signed message can be verified by any principal with the knowledge of k_p

- Asymmetric ~~crypto~~ authentication protocols can be developed using the principle called ASYM:

  "If the principal can correctly sign a message using the private key of the claimed identity, then this act constitutes sufficient proof of identity"

Basic protocol →

P → Q : "I am P"
Q : generate nonce n
Q → P : n
P : compute $m = \{P, Q, n\}_{k_p^{-1}}$
P → Q : m
Q : verify $\{P, Q, n\} = m k_p$
: if equal then accept, otherwise authentication fails.

A modified protocol with certificate authority →

Basic protocol assumes that Q knows P's public key. Problem arises when Q does not know the public key. This problem is alleviated by a certificate authority that maintains a database of all the published public keys. If a user A does not have the public key of another user B, A can request B's public key from the CA.

Kerberos Authentication Service →

- Kerberos primarily addresses client-server authentication using a symmetric cryptosystem together with trusted third-party authentication servers.

- The basic components include authentication servers (Kerberos servers) and Ticket Granting servers (TGS)

Initial Registeration →

→ Every client / user must register with the Kerberos server by providing it's user id (u) and password (password u).

→ The Kerberos server computes a key $k_u = f(password_u)$ using a one way function f and stores this key in the database

→ k_u is a secret key that depends on the password of the user and it is shared by the client u and Kerberos server only.

The Authentication Protocol →

1. Initial authentication at login :

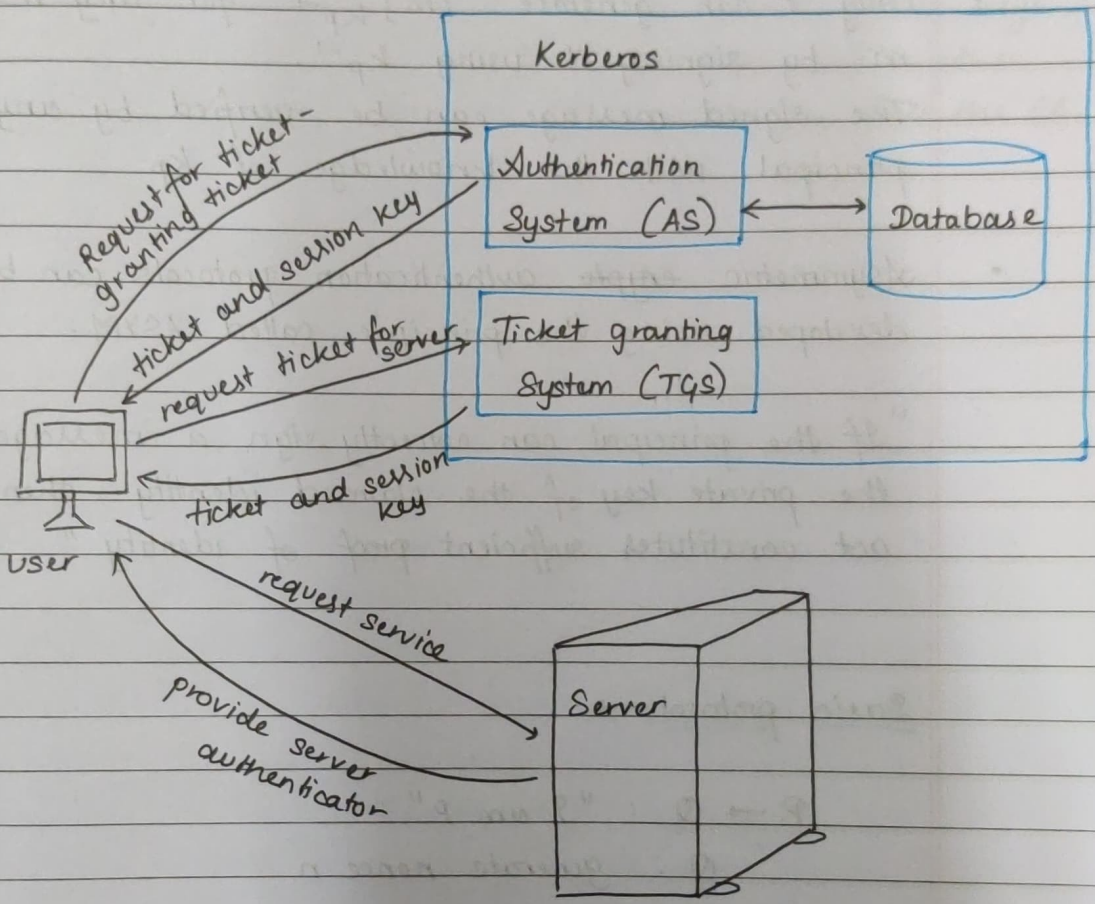
The Kerberos server authenticates user login at a host and installs a ticket for the ticket granting server (TGS) at the login host.

2. Obtain a ticket for the server :

Using the ticket for the ticket granting server, the client requests the ticket granting server (TGS) for a ticket for the server

3. Requesting service from the server :

The client uses the server ticket obtained from the TGS to request services from the server.



Weaknesses →

- Kerberos makes no provisions for host security, assumes that it is running on trusted hosts with an untrusted network
- Kerberos uses the principal's password as the fundamental proof of identity. If a user's Kerberos password is stolen by an attacker, they can easily impersonate the user.

Content Addressable Network (CAN):

- The Content Addressable network is a distributed, decentralized P2P infrastructure that provides a hash Table functionality on an Internet like-scale.
- Can was one of the original four distributed hash table proposals, introduce Concurrently with Chord, Pastry, & Tapestry
- Can is distributed hash table designed to map the ~~Value~~ file name to a specific network location
- Can has a more key features like:

i) Scalability:

- Can Supports the Operations like:
  1) Search
  2) insertion
  3) Deletion

- Key Features of CAN:

i) Scabbility:

Can CAN can accomodate a large number of Nodes in the system efficiently

2) Decentralization:

it Operates in a decentralized manner Where ~~eac~~ every node is responsible for small Portion of data.

3) Effecient Data retrival:

- CAN uses the distributed hash table to map the data items to their particular respective nodes based on Content Address.

4) Load Balancing:

- Data distribution across the nodes is balanced, ensures that no node is overloaded with data.

5) Fault Tolerance:

CAN Employ's Data replication and redundancy. to tolerate the ~~failure~~ Failure & ensures the Data Availability

Working of CAN: (or) How CAN works:

Co-Ordinate Space portitioning:

- The Co-ordinate Space is portitioned into zone with each node is responsible for managing the zone

Routing:

- When a data item is need to be stored or retrived, routing mechanisms determines the responsible node based on the item's Content Address

3) Data Replication :

- Data replication ensures the fault tolerance :
- Data replication across all nodes ensures fault tolerence & Availbity (or) High Availability.

4) Dynamic Node Join and Leave :

- CAN can dynamically add a node and removes a node by maintaing Data integrity and Consistency.

(or)

CAN Suppors dynamic Addition & removal of Nodes while maintaing Data integrity & Consistency.

Use Cases of CAN :

1) Distributed Storag System
2) Content Delivery Networks
3) Peer to peer ~~CAN~~ Network.

In Conclusion CAN Offerss a scalable, decentralized & fault tolerant, and robust architecture by employing a routing Mechanisms for distributed database