# Authentication

Unit -5 [3Hours]

# Authentication

- Authentication is the process of verifying the identity of user or information. User authentication is the process of verifying the identity of user when that user logs into a computer system.
- Authentication is the binding of an identity to a subject i.e. it is the process of determining whether a user ( or other entity) should be allowed access to a system.
- This process consists of sending the credentials from the remote access client to the remote access server in an either plaintext or encrypted form by using an authentication protocol.
- The credentials provided are compared to those on a file in a database of the authorized user's information on a local operating system or within an authentication server.
- Only authenticated users are allowed access to system resources.
- It consists of two steps: Identification step, and Verification step
  - **Authentication**: Are you who you say you are?
  - A**uthorization**: Are you allowed to do that?

# Authentication

- The external entity must provide information to enable to the subject to confirm its identity. This information comes from one (or more) of the following:
  - What the entity knows (such as passwords or secret information)
  - What the entity has (such as a badge or card)
  - What the entity is (such as fingerprints or retinal characteristics)
  - Where the entity is (such as in front of a particular terminal)
- The authentication process consists of obtaining the authentication information from an entity, analyzing the data, and determining if it is associated with that entity. This means the computer must store some information about the entity.

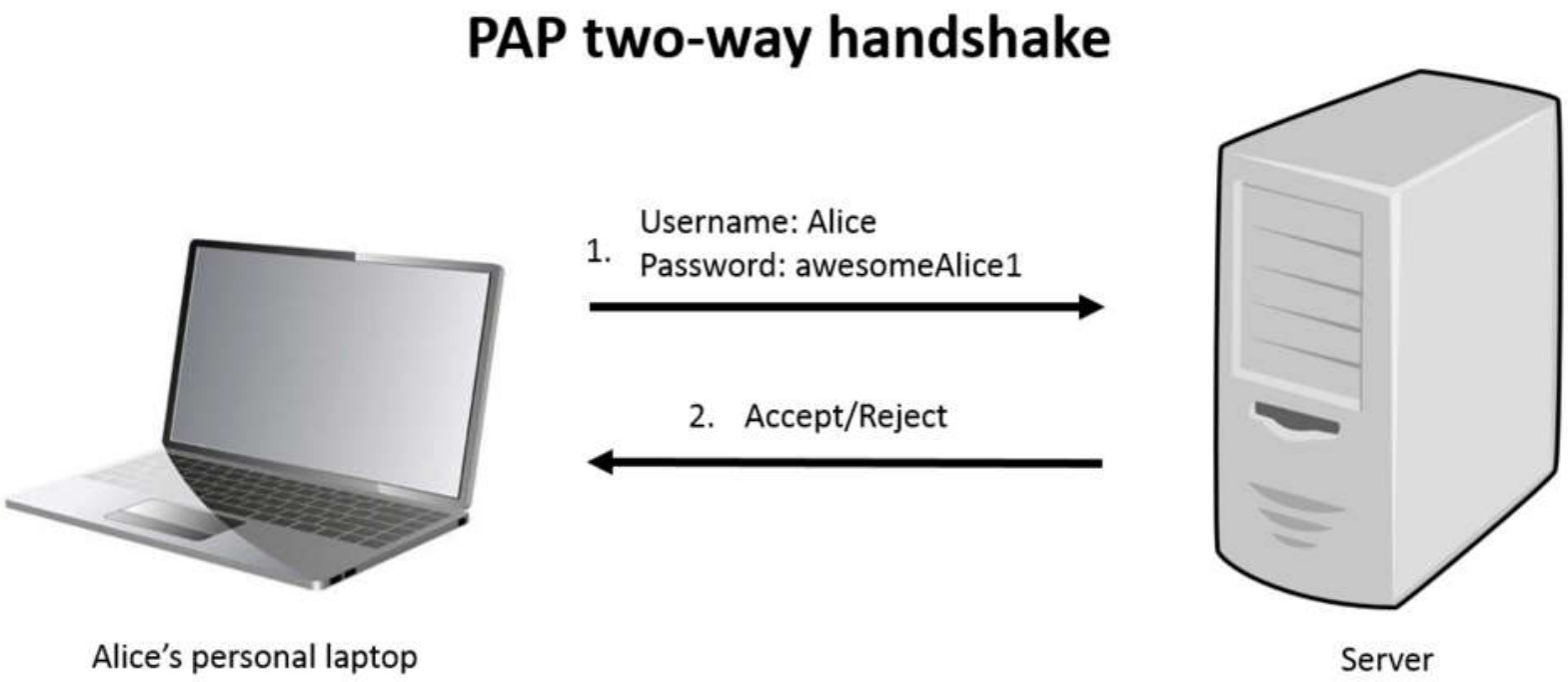
# Authentication System

- Technique that provides access control for systems by checking to see if a user's credentials match the credentials in a database of authorized users or in a data authentication severs.
- Authentication is any process by which a system verifies the identify of a user who wishes to access it.
- Four general means of authenticating a user's identity:
  1. **something the individual knows**:
     - password, PIN, answers to a prearranged set of questions
  2. **something the individual possesses**:
     - cryptographic keys, electronic keycards, smart cards, physical cards (referred to as token)
  3. **something the individual is (static biometrics)**:
     - recognition by fingerprint, retina, and face
  4. **something the individual does (dynamic biometrics)**:
     - recognition by voice pattern, handwriting characteristics, and typing rhythm

# Password Based Authentication

- A **password** is information associated with an entity that confirms the entity's identity.
- It consists of string of alphabets, numbers and special characters, which is supposed only to the entity (usually person) that is being authenticated.
- Passwords are often stored as hash value of original password.
- Passwords are an example of an authentication mechanism based on what people know: the user supplies a password, and the computer validates it.
- If the password is the one associated with the user, that user's identity is authenticated. If not, the password is rejected and the authentication fails.
- Typically, a system that uses password-based authentication maintains a password file indexed by user ID. One technique that is typically used is to store not the user's password but a one-way hash function of the password

# Password Based Authentication



# Password Based Authentication

- The password serves to authenticate the ID of the individual logging on to the system. In turn, the ID provides security in the following ways:
  - **The ID determines whether the user is authorized to gain access to a system**. In some systems, only those who already have an ID filed on the system are allowed to gain access.
  - **The ID determines the privileges accorded to the user**. A few users may have supervisory or "superuser" status that enables them to read files and perform functions that are especially protected by the operating system. Some systems have guest or anonymous accounts, and users of these accounts have more limited privileges than others.
  - **The ID is used in what is referred to as discretionary access control**. For example, by listing the IDs of the other users, a user may grant permission to them to read files owned by that user.

# Dictionary Attack

- A dictionary attack is a type of cybersecurity attack where an attacker attempts to gain unauthorized access to a system or network by systematically trying a large number of words or phrases, often sourced from a dictionary or a list of commonly used passwords. This method relies on the assumption that many users choose weak passwords that are easily guessable.

# Best Practice to defend against dictionary and brute-force attack

To defend against dictionary attacks, organizations and individuals can take several measures:

- **Strong Password Policies**: Encourage or enforce the use of strong passwords that include a combination of uppercase and lowercase letters, numbers, and special characters. This makes it more difficult for attackers to guess passwords.
- **Multi-Factor Authentication (MFA)**: Implementing MFA adds an extra layer of security by requiring users to provide a second form of identification, such as a code sent to their mobile device, in addition to a password.
- **Account Lockout Policies**: Implement account lockout policies that temporarily lock an account after a certain number of failed login attempts. This can help prevent brute force and dictionary attacks.
- **Regular Password Changes**: Encourage users to change their passwords regularly to reduce the likelihood of successful attacks.
- **Monitoring and Logging**: Regularly monitor login attempts and maintain logs to detect and respond to suspicious activity promptly.

# Offline Dictionary Attack

- Typically, strong access controls are used to protect the system's password file.
- However, experience shows that determined hackers can frequently bypass such controls and gain access to the file.
- The attacker obtains the system password file and compares the password hashes against hashes of commonly used passwords.
- If a match is found, the attacker can gain access by that ID/password combination.
- Countermeasures include controls to prevent unauthorized access to the password file and intrusion detection measures to identify a compromise

# Challenge Response System

- A Challenge-Response System (also called *Challenge Response Authentication Mechanism*) is a security mechanism designed to authenticate or verify the identity of a user or system.
- It typically involves a back-and-forth process where one party presents a challenge, and the other party responds appropriately to prove their identity.
- Challenge-Response Systems are used in various security applications, and they provide additional level of security beyond traditional username and password authentication. One common application of challenge-response is in *two-factor authentication (2FA)* or *multi-factor authentication (MFA) systems*.

# Challenge Response System: Workflow

1. **Challenge Generation**: The system or entity seeking authentication generates a unique challenge. This challenge is a specific request or task that requires a response.
2. **Challenge Issuance**: The challenge is sent to the user or entity that needs to be authenticated.
3. **Response Generation**: The user or entity receiving the challenge must generate a response based on the challenge. This response is often generated using a secret or cryptographic key.
4. **Response Submission**: The generated response is sent back to the system that issued the challenge.
5. **Verification**: The system that issued the challenge verifies the received response. If the response is correct and matches the expected result, the authentication is successful.

# Challenge Response System: Method for Implementation

There are different methods for implementing Challenge-Response Systems, and they may include:

1. **Time-based One-Time Passwords (TOTP):** The challenge is often a unique code generated based on the current time, and the user responds with a code generated on their device using a shared secret and the current time.

2. **Public Key Cryptography**: Challenges and responses can be based on public-key cryptographic algorithms, where the challenge is encrypted with a public key, and the response is decrypted with the corresponding private key.

3. **Biometric Challenge-Response**: Challenges can involve biometric data (e.g., fingerprints, facial recognition), and the response is generated by providing the matching biometric information.

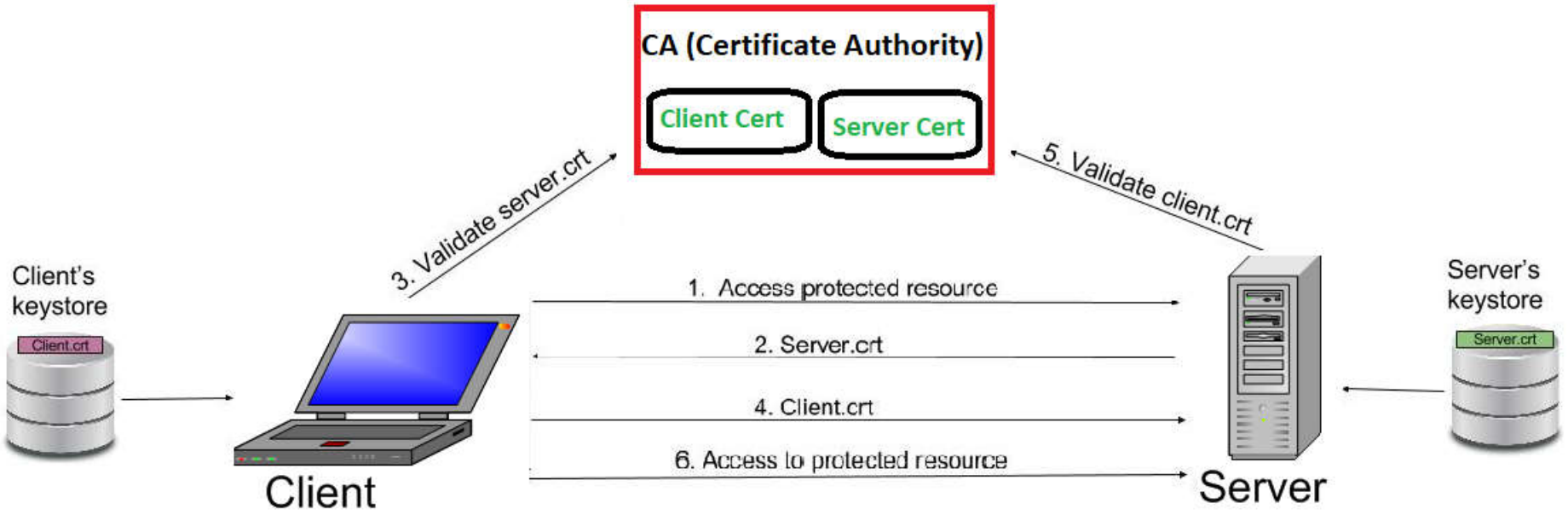
# Mutual Authentication

- Mutual authentication is a desired characteristic in verification schemes that transmit sensitive data, in order to ensure data security.
- Mutual authentication can be accomplished with two types of credentials: usernames and passwords, and public key certificates.
- Mutual authentication is often employed in the Internet of Things (IoT).
- Mutual authentication protect communication against adversarial attacks such as man in the middle attack, replay attacks, spoofing attacks, impersonation attacks.

# Mutual Authentication

- Establishing the authentication using certificate based 2-way SSL involves:
  - A client request access to a protected resource.
  - The sever presents its certificate to the client.
  - The client verifies the server's certificate.
  - If successful, the client sends its certificate to the server.
  - The server verifies the clients credentials.
  - If successful, the server grants access to the protected resource requested by the client.

# Mutual Authentication



# Mutual Authentication

Below is the high level description of the steps involved in establishment of connection and transfer of data between a client and server in case of two-way SSL:

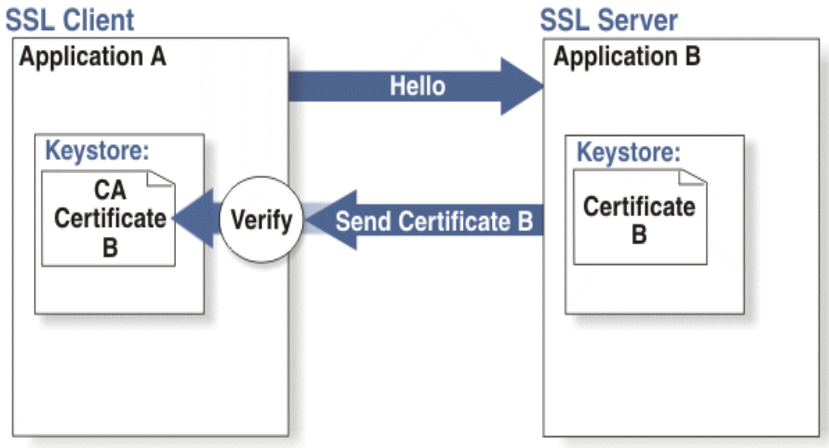
1. Client requests a protected resource over HTTPS protocol and the SSL/TSL handshake process begin.
2. Server returns its public certificate to the client along with server hello.
3. Client validates/verifies the received certificate. Client verifies the certificate through certification authority (CA) for CA signed certificates.
4. If Server certificate was validated successfully, client will provide its public certificate to the server.
5. Server validates/verifies the received certificate. Server verifies the certificate through certification authority (CA) for CA signed certificates.
6. After completion of handshake process, client and server communicate and transfer data with each other encrypted with the secret keys shared between the two during handshake.

# One Way Authentication

- One-way authentication is a process or technology in which only client authenticates server's identity before actual communication occurs.
- This is to ensure that clients are communicating exclusively with legitimate servers.
- For implementing one-way SSL, server shares its public certificate with the clients.

# One Way Authentication

- Establishing the authentication using certificate-based-1-way SSL involves:
  - A client requests access to a protected resource.
  - The server presents its certificate to the client.
  - The client verifies the server's certificate.
  - If successful, the client authenticates the server as legitimate.

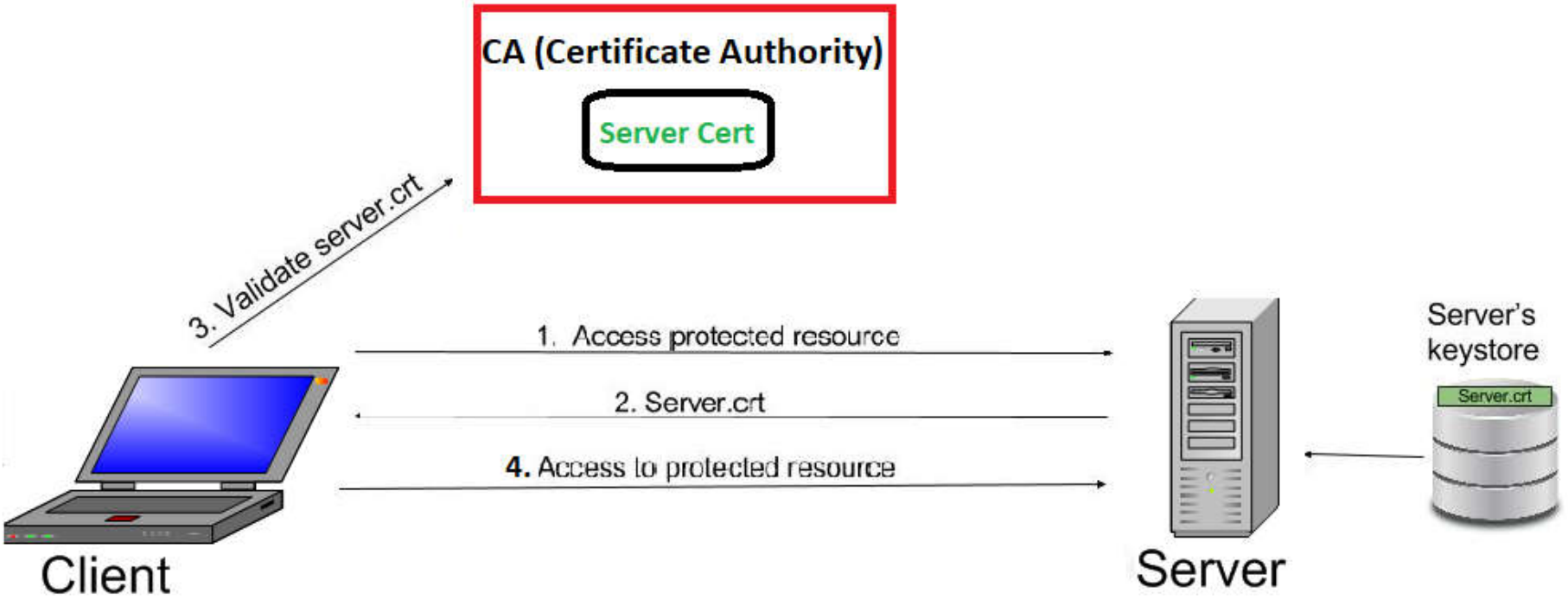


# How one-way SSL Works?

Below is the high level description of the steps involved in establishment of connection and transfer of data between a client and server in case of one-way SSL:

- Client requests for some protected data from the server on HTTPS protocol. This initiates SSL/TLS handshake process.
- Server returns its public certificate to the client along with server hello message.
- Client validates/verifies the received certificate. Client verifies the certificate through certification authority (CA) for CA signed certificates.
- SSL/TLS client sends the random byte string that enables both the client and the server to compute the secret key to be used for encrypting subsequent message data. The random byte string itself is encrypted with the server's public key.
- After agreeing on this secret key, client and server communicate further for actual data transfer by encrypting/decrypting data using this key.

# One Way Authentication



# Biometric Authentication

- A biometric authentication system attempts to authenticate an individual based on his or her unique physical characteristics.
- These include *static characteristics*, such as fingerprints, hand geometry, facial characteristics, and retinal and iris patterns; and *dynamic characteristics*, such as voiceprint and signature.
- In essence, biometrics is based on pattern recognition.
- Compared to passwords and tokens, biometric authentication is both technically more complex and expensive

# Biometric Authentication

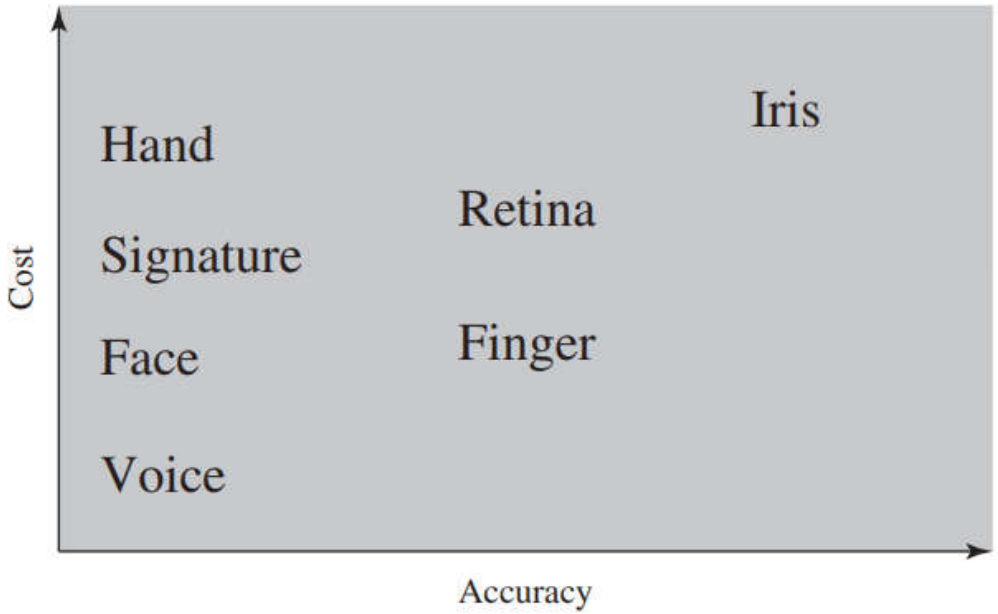
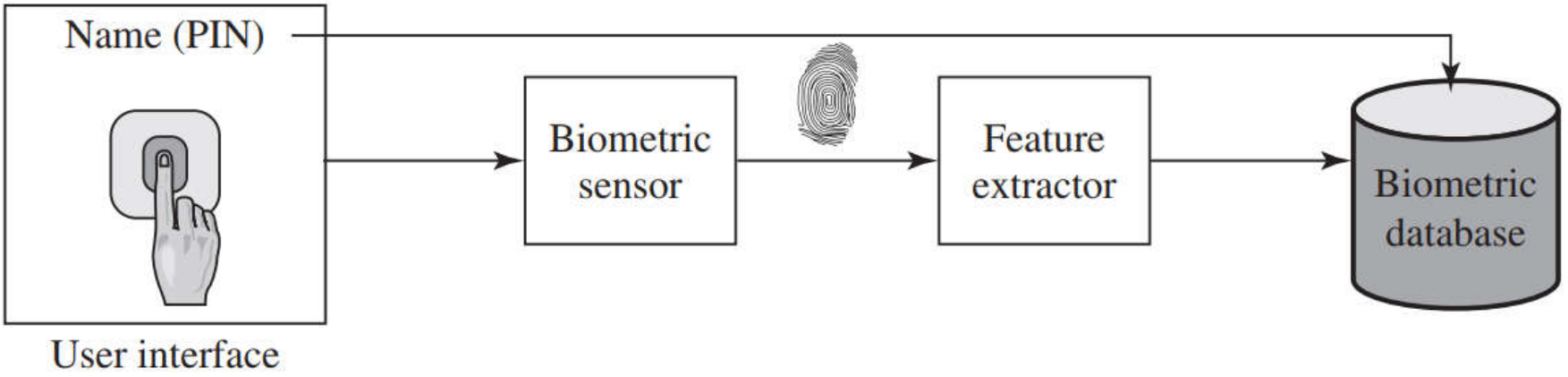



**Figure**: Cost versus accuracy of various biometric characteristics in user authentication schemes

# Operation of Biometric Authentication System

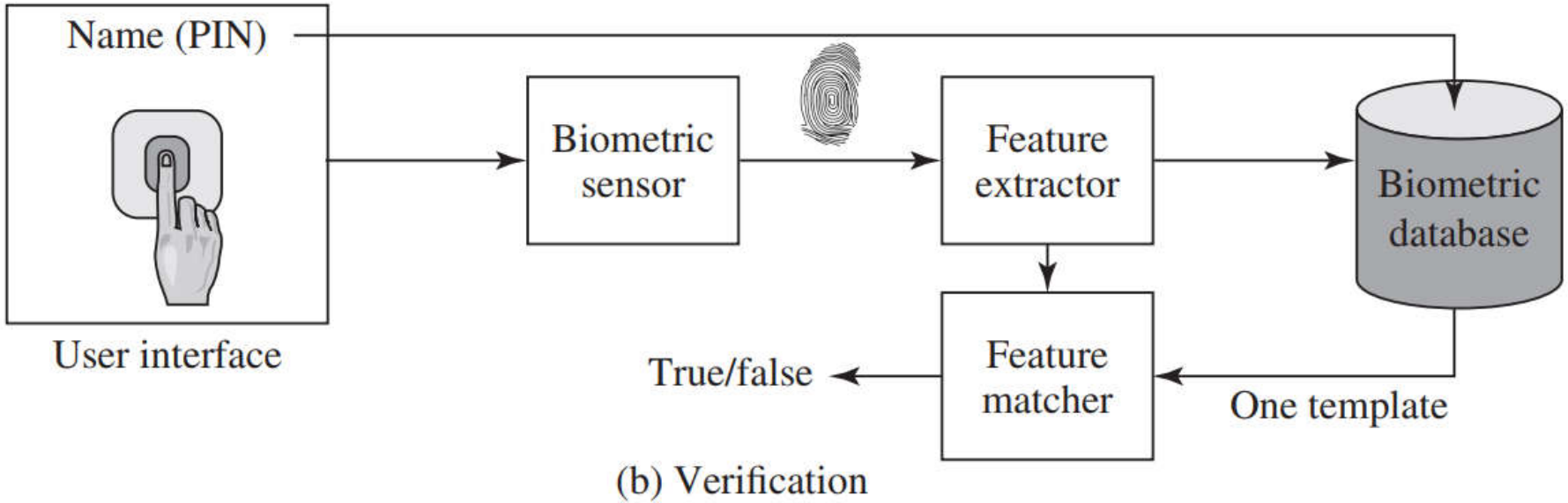
- A Generic Biometric System consists of
  - Enrollment
  - Verification
  - Identification
- Enrollment creates an association between a user and the user's biometric characteristics.
- Depending on the application, user authentication either involves verifying that a claimed user is the actual user or identifying an unknown user.

# Operation of Biometric Authentication System

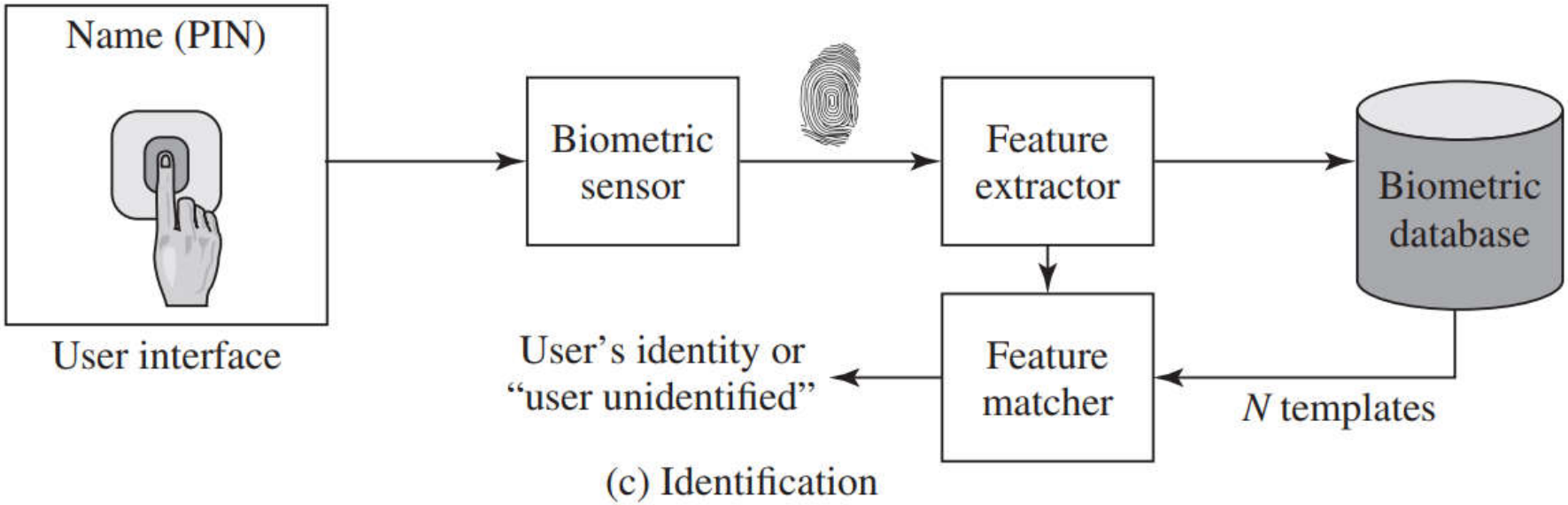



(a) Enrollment

# Operation of Biometric Authentication System



# Operation of Biometric Authentication System



# Needham-Schroeder Protocol

- The Needham-Schroeder protocol (or Needham-Schroeder key distribution protocol) is a cryptographic protocol designed for secure key exchange between two parties over an insecure network.
- It was proposed by Roger M. Needham and Michael D. Schroeder in 1978.
- The primary goal of this protocol is to enable two parties to establish a shared secret key for subsequent secure communication.
- It is the original, basic key exchange protocol.
- Used by 2 parties who both trusted a common key server, it gives one party the info needed to establish a session key with the other.
- Note that since the key server chooses the session key, it is capable of reading/forging any messages between A & B, which is why they need to trust it absolutely!

# Needham-Schroeder Scheme

- used to securely distribute a new session key for communications between A & B
- but is vulnerable to a replay attack if an old session key has been compromised
  - then message 3 can be resent convincing B that is communicating with A
- modifications to address this require:
  - timestamps (Denning 81)
  - using an extra nonce (Neuman 93)

# Needham-Schroeder Scheme

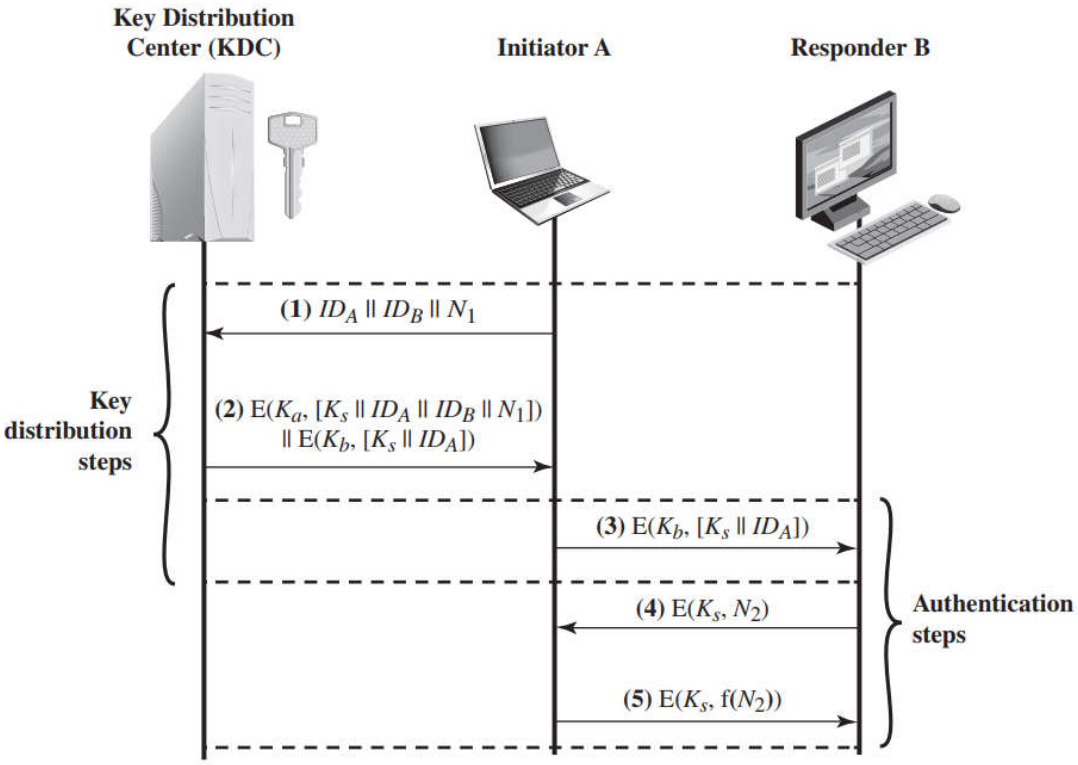



**Figure**: Key Distribution Scenario

Protocol overview is:

1. A->KDC: $ID_A$ || $ID_B$ || $N_1$
2. KDC -> A: $E_{Ka}$[Ks || $ID_B$ || $N_1$ || $E_{Kb}$[$Ks$||$ID_A$] ]
3. A -> B: $E_{Kb}$[$Ks$||$ID_A$]
4. B -> A: $E_{Ks}$[$N_2$]
5. A -> B: $E_{Ks}$[f($N_2$)]

# Needham-Schroeder Scheme

1. **Initialization:**
   - Alice and Bob share a trusted server, often referred to as the Key Distribution Center (KDC).
   - Alice wants to establish a secure communication channel with Bob.
2. **Request Phase:**
   - Alice sends a message to the KDC requesting a secure session with Bob.
   - The message typically includes Alice's identity and Bob's identity and Nonce (N1)
3. **Key Distribution:**
   - The KDC generates a new session key (Ks) for Alice and Bob.
   - The KDC encrypts the session key with Alice's key (Ka) and Bob's key (Kb) and sends the encrypted message to Alice.

# Needham-Schroeder Scheme

4. **Session Key Exchange:**
   - Alice receives the encrypted message from the KDC and decrypts it using her own key (Ka).
   - Alice re-encrypts the session key with Bob's key (Kb) and sends it to Bob.
5. **Finalization:**
   - Bob receives the message from Alice, decrypts it using his key (Kb), and obtains the session key (Ks).
   - Now, Alice and Bob both have the shared session key (Ks), which they can use for secure communication.

# Kerberos v4

- Kerberos is **a key distribution and user authentication service** developed at MIT.
- Kerberos **allows two users (or client and server) to authenticate each other over an insecure network, such as internet**.
- Named after the Greek mythological character Kerberos (or Cerberus), known in Greek mythology as being the monstrous three-headed guard dog of Hades.
- The three heads of the Kerberos protocol represent a client, a server and a Key Distribution Center (KDC), which acts as a trusted third party authentication service.
- Designed originally for Project Athena at M.I.T.
- Windows 2000/XP/Server 20003/Vista, Apple's MAC and Linux use Kerberos as their default authentication mechanism.
- Protects against eavesdropping and replay attacks.
- Uses a trusted third party **(key distribution center**) and **symmetric key cryptography (DES)**.
- Each user and service on the network is principal.
- Two versions are in use v4 and v5, thought v4 is being phased out.

# Kerberos v4

The Kerberos protocol works on the basis of tickets. Here's a simplified overview of how it works:

1. **Authentication Server (AS):** The AS performs the desired client authentication. The user authenticates with the Authentication Server and receives a `Ticket Granting Ticket (TGT)` upon successful authentication. This ticket assures the other servers that the client is authenticated

2. **Ticket Granting Server (TGS)**: When the user wants to access a specific service, they present the TGT to the Ticket Granting Server, requesting a service ticket for the desired service.

3. **Service Server (SS)**: The Ticket Granting Server provides the user with a service ticket. The user presents this service ticket to the Service Server, which then grants access to the requested service if the ticket is valid.

# Kerberos Process

- An authentication server (AS) that knows the passwords of all users and stores these in a centralized database.
- In addition, the AS shares a unique secret key with each server. These keys have been distributed physically or in some other secure manner. Consider the following hypothetical dialogue:

1) C → AS: $ID_C \parallel P_C \parallel ID_V$

2) AS → C : $Ticket$

3) C → V: $ID_C \parallel Ticket$

$Ticket = E(K_V, [ID_C \parallel AD_C \parallel ID_V])$

- where,
  - C = client
  - AS = authentication server
  - V = server
  - $ID_C$ = identifier of user on C
  - $ID_V$ = identifier of V
  - $P_C$ = password of user on C
  - ADC = network address of C
  - $K_v$ = secret encryption key shared by AS and V

# Kerberos v4: Step 1

- In this scenario, the user logs on to a workstation and requests access to server V.
- The client module C in the user's workstation requests the user's password and then sends a message to the AS that includes the user's ID, the server's ID, and the user's password.

# Kerberos v4: Step 2

- The AS checks its database to see if the user has supplied the proper password for this user ID and whether this user is permitted access to server V. If both tests are passed, the AS accepts the user as authentic and must now convince the server that this user is authentic.
- To do so, the AS creates a **ticket** that contains the user's ID and network address and the server's ID.
- This ticket is encrypted using the secret key shared by the AS and this server. This ticket is then sent back to C. Because the ticket is encrypted, it cannot be altered by C or by an opponent

# Kerberos v4: Step 3

- With this ticket, C can now apply to V for service.
- C sends a message to V containing C's ID and the ticket.
- V decrypts the ticket and verifies that the user ID in the ticket is the same as the unencrypted user ID in the message.
- If these two match, the server considers the user authenticated and grants the requested service.

# Analysis of step 3

- Each of the ingredients of message (3) is significant. The ticket is encrypted to prevent alteration or forgery.
- The server's ID ($ID_V$) is included in the ticket so that the server can verify that it has decrypted the ticket properly.
- $ID_C$ is included in the ticket to indicate that this ticket has been issued on behalf of C.
- Finally, $AD_C$ serves to counter the following threat.
  - An opponent could capture the ticket transmitted in message (2), then use the name IDC and transmit a message of form (3) from another workstation.
  - The server would receive a valid ticket that matches the user ID and grant access to the user on that other workstation.
  - *To prevent this attack, the AS includes in the ticket the network address from which the original request came. Now the ticket is valid only if it is transmitted from the same workstation that initially requested the ticket.*

# Ticket Granting Service

- A user would need a new ticket for every different service.
- If a user wished to access a print server, a mail server, a file server, and so on, the first instance of each access would require a new ticket and hence require the user to enter the password.
- The new (but still hypothetical scenario is as follows:

**Once per user logon session:**

**(1)** $C \rightarrow AS: \quad ID_C \| ID_{tgs}$

**(2)** $AS \rightarrow C: \quad E(K_c, Ticket_{tgs})$

**Once per type of service:**

**(3)** $C \rightarrow TGS: \quad ID_C \| ID_V \| Ticket_{tgs}$

**(4)** $TGS \rightarrow C: \quad Ticket_v$

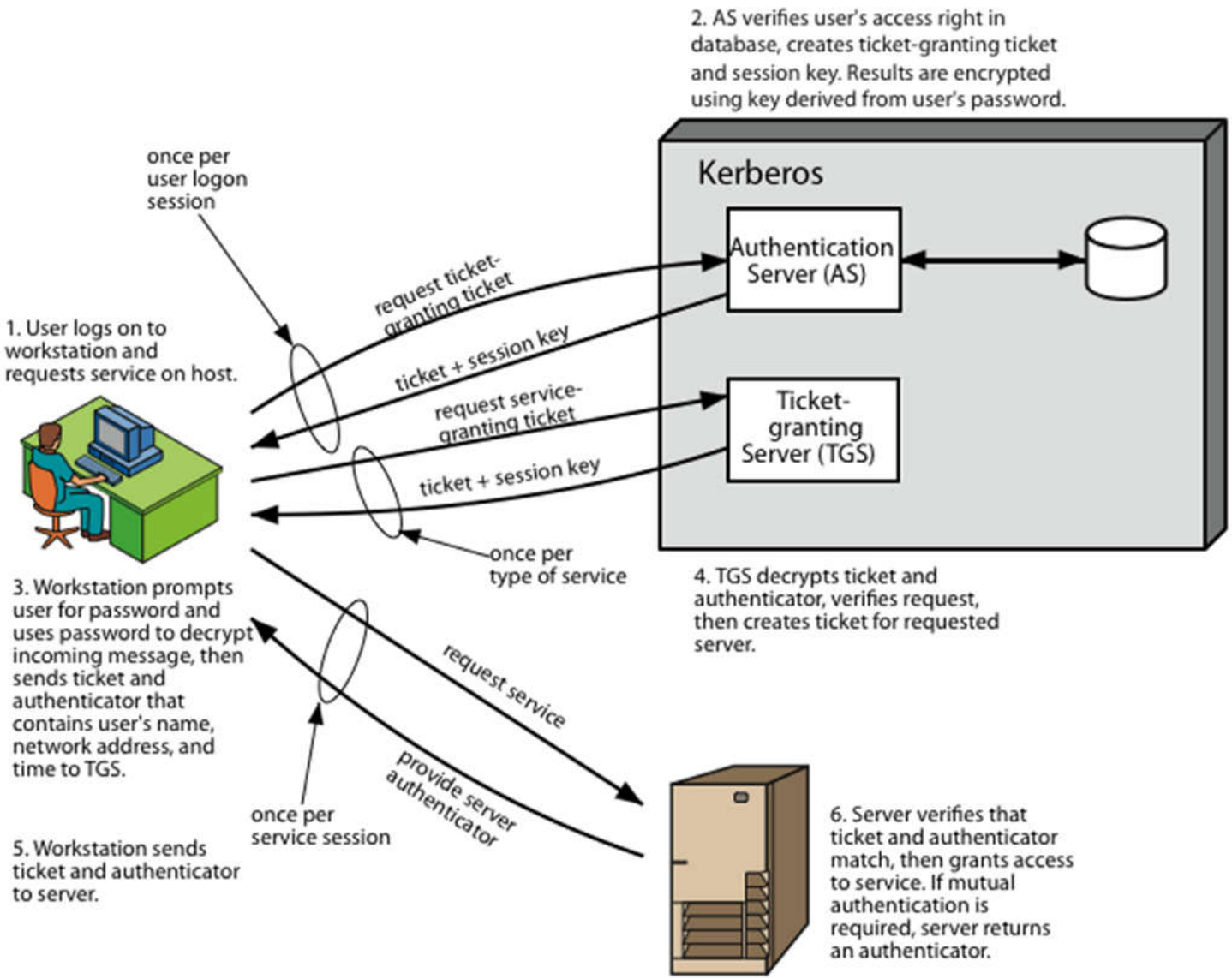
**Once per service session:**

**(5)** $C \rightarrow V: \quad ID_C \| Ticket_v$

$$Ticket_{tgs} = E(K_{tgs}, [ID_C \| AD_C \| ID_{tgs} \| TS_1 \| Lifetime_1])$$
$$Ticket_v = E(K_v, [ID_C \| AD_C \| ID_v \| TS_2 \| Lifetime_2])$$

# Ticket Granting Service

- The new service, TGS, issues tickets to users who have been authenticated to AS.
- Thus, the user first requests a ticket-granting ticket ($Ticket_{tgs}$) from the AS.
- The client module in the user workstation saves this ticket.
- Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself.
- The TGS then grants a ticket for the particular service.
- The client saves each service-granting ticket and uses it to authenticate its user to a server each time a particular service is requested.
- Because only the correct user should know the password, only the correct user can recover the ticket.
- Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5.


2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
Kerberos
Authentication Server (AS)
Ticket-granting Server (TGS)
once per user logon session
request ticket-granting ticket
ticket + session key
request service-granting ticket
ticket + session key
once per type of service
1. User logs on to workstation and requests service on host.
3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.
4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.
request service
provide server authenticator
once per service session
5. Workstation sends ticket and authenticator to server.
6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.