Password-Based Login Protocol Design

1. How the Server Stores Passwords

- The server will store passwords using salted hashes to enhance security. Each user's password will be processed as follows:

- Salt Generation: For each user, a unique salt (a random value) will be generated when the user registers.
- Hashing: The password concatenated with the salt will be hashed using a secure hashing algorithm (e.g., SHA-256 , bcrypt).
- Storage Format: The server will store each user’s record in a text file in the following format:

[User_name],[Salt],[Salted_Hash_of_password]

2. Message Flow of the Protocol

- Here’s the sequence of steps in the login protocol:

a. Client Initiates Login: The client sends a login request to the server.
Client -> Server: LOGIN_REQUEST {username}

b. Server Responds with Nonce: The server generates a nonce (a unique random value) and sends it back to the client.
Server -> Client: NONCE {nonce}

c.Client Calculates Hash: The client retrieves the user's password, combines it with the received nonce, and sends the hash to the server.
Client -> Server: LOGIN_HASH {username, hash(password + nonce)}

d. Server Verifies Credentials:
The server retrieves the user's record (salt and stored hash) from the database.
It computes the expected hash using the stored salt and compares it with the hash received from the client.

Server Response:
If the hashes match, the server sends a success message.
If they do not match, it sends a failure message.
Server -> Client: LOGIN_RESPONSE {success/failure}

3. Server Verification Process

The server performs the following steps to verify the login attempt:

- Retrieve the user's record from the password database using the username.
- Extract the salt and stored hash.
- Compute the hash of the password concatenated with the nonce - using the stored salt.
- Compare the computed hash with the hash received from the client.
- Return a success or failure response based on the comparison.

Security Analysis

1. Resistance to Offline Dictionary Attacks:
Since passwords are hashed with a unique salt, an attacker cannot utilize precomputed rainbow tables, making offline dictionary attacks significantly harder.
Hashing functions (like bcrypt or Argon2) are designed to be slow, which further impedes brute force attempts.

2. Nonce Usage:
The use of nonces protects against replay attacks. Even if an attacker captures a successful login response, they cannot reuse it later since the nonce will be different for each session.

3. Mitigation of Man-in-the-Middle Attacks:
If SSL/TLS is used to encrypt messages between the client and server, it will help prevent interception of sensitive data.

4. Password Security:
Storing salted hashes instead of plain text passwords ensures that even if the database is compromised, passwords remain secure.

5. Confidentiality:
The password is never transmitted in plaintext. The client computes a hash that includes the nonce, making it difficult for an eavesdropper to extract the password or reuse the hash.