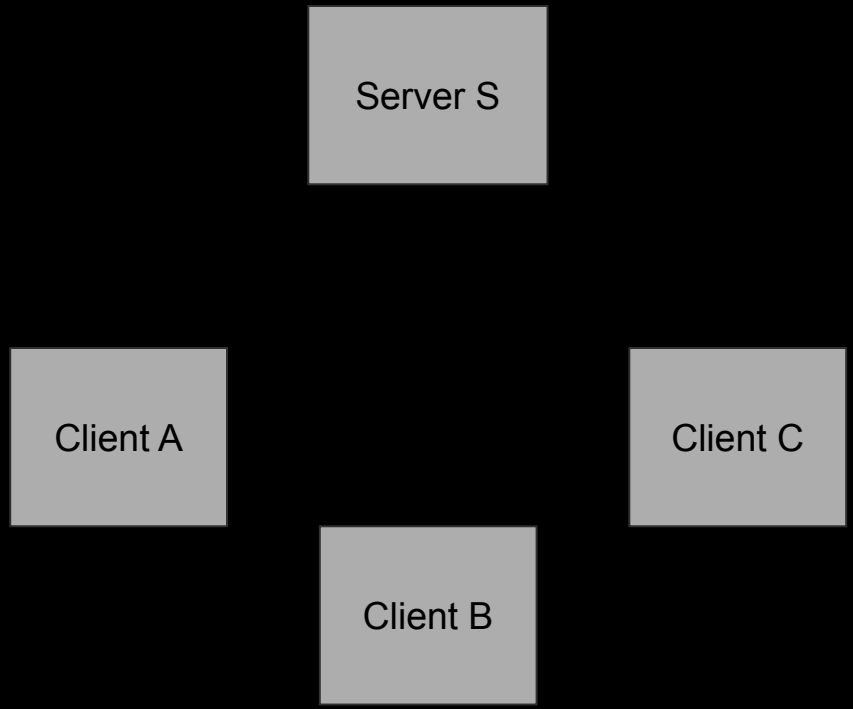
# Architecture

We will use a Certificate Authority server over a KDC, for three main reasons:

1. Server is oblivious to who the client is talking to.
2. Server does not create any kind of keys between two clients – ensuring higher privacy.
3. After the user is authenticated, and the server goes down, users can still initiate new communications as server is not required for key derivation.



# Assumptions

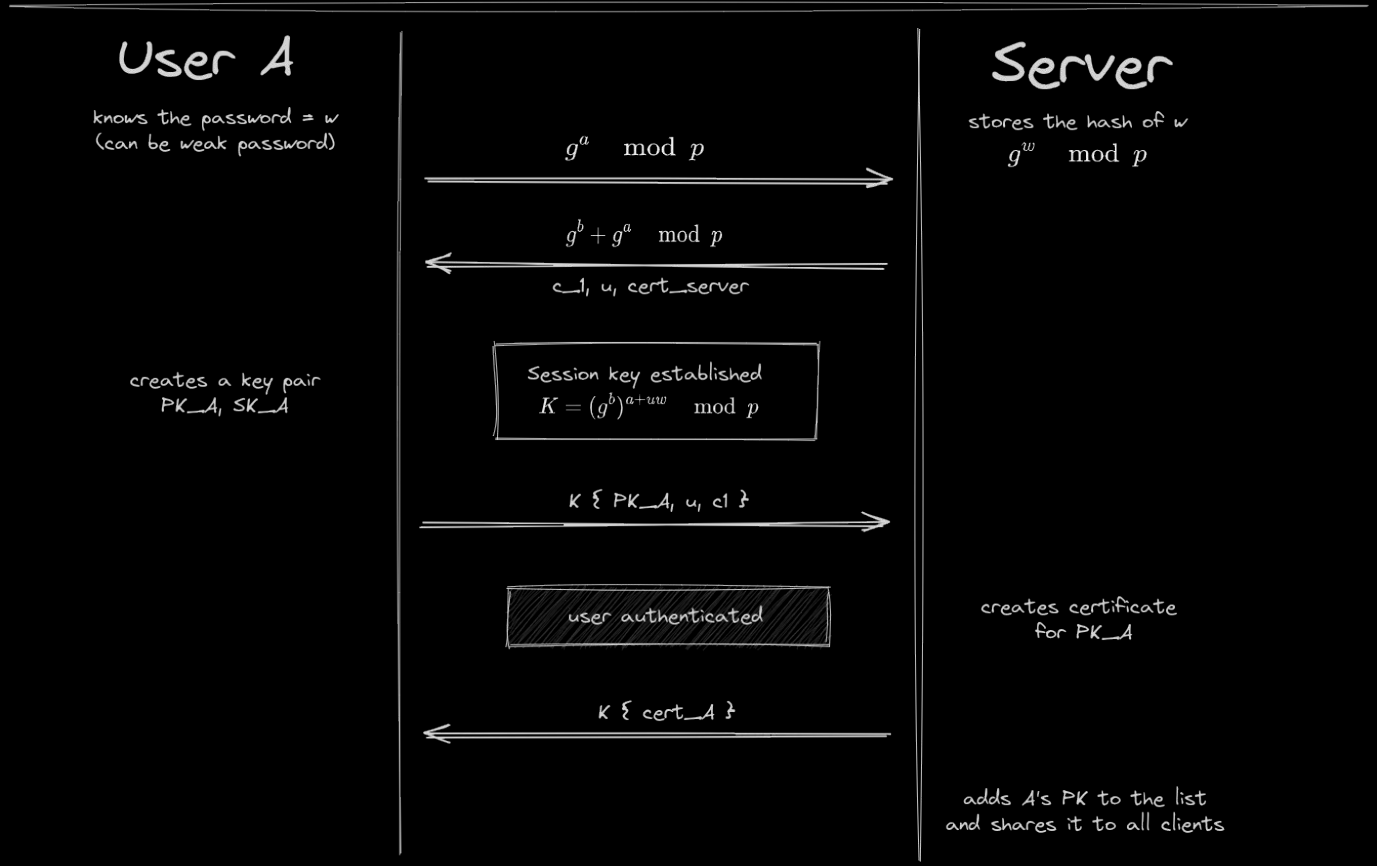
- Each client trusts the only one party and that is the server
- Each client application is pre configured to know the server’s address
- After registration, the server stores a hash of the password
- User’s password is not leaked

# Services

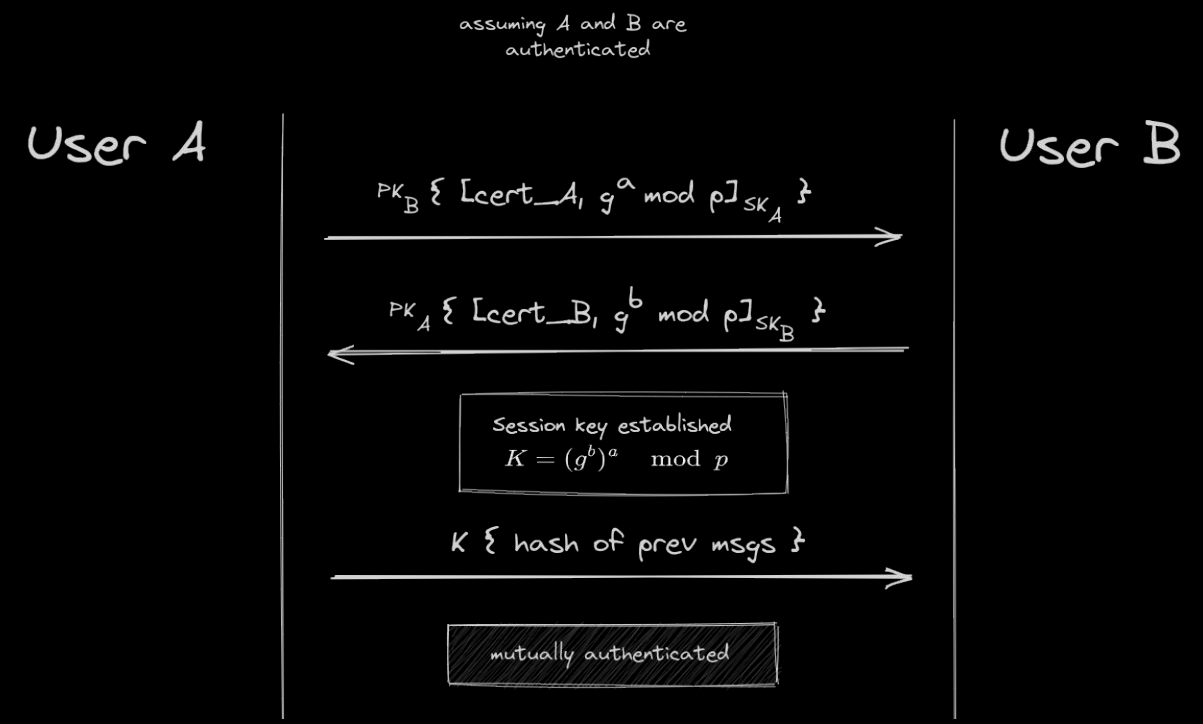
This design will provide:

- perfect forward secrecy
- protection against weak passwords
- protection from the server decrypting messages between users
- Protection from the server knowing the any ongoing communication

# Authentication Protocol (Login)



# Key Establishment Protocol



# Messaging Protocol

1. Once A and B mutually authenticate themselves, they can start communicating using key K.
2. Each message has the following contents:
    a. Message length:
    b. Sender ID:
    c. Receiver ID:
    d. Cipher text: {sender + receiver + timestamp + plaintext} encrypted with key K_ab
    e. Signature : hash {plaintext + timestamp + sender + receiver}
3. To create the cipher text, we encrypt the plaintext with K using AES256 in CTR (counter) mode. So K is a 256-bit key.
4. Time stamp inside cipher text is provided to stop replay attacks.
5. Signature allows the receiver to be sure that the message has not been tampered with. The inclusion of timestamp again ensures no replays, but sender and receiver ensures that each signature is unique.
6. NOTE: Receiver always checks if the message length and the actual length match

# Logout Protocol

1. Client initiates the logout and the application sends logout request to the server.
2. The request includes user and timestamp, encrypted with K.
3. The server uses this session key to verify the client and session.
4. Once verified server terminates the user’s session key, and forgets b.
5. **Server revokes the certificate** and sends logout confirmation to user.
6. Client application updates the local state, forgets a and the key pair.
7. This provides PFS, as the attacker in the future cannot get K for that session, without both a and b.