Requirements and Fulfillments

1. Message Exchange Functionality:

- We have used UDP instead of TCP protocol
- This allows us to dynamically change the port to which the messages can be sent

2. Authentication:

- Used RSA key-based authentication to implement
- communication can be encrypted using RSA cryptography using both the public and the private keys. The opposite key from the one that was used to encrypt a message is used to decrypt it.

3. Confidentiality:

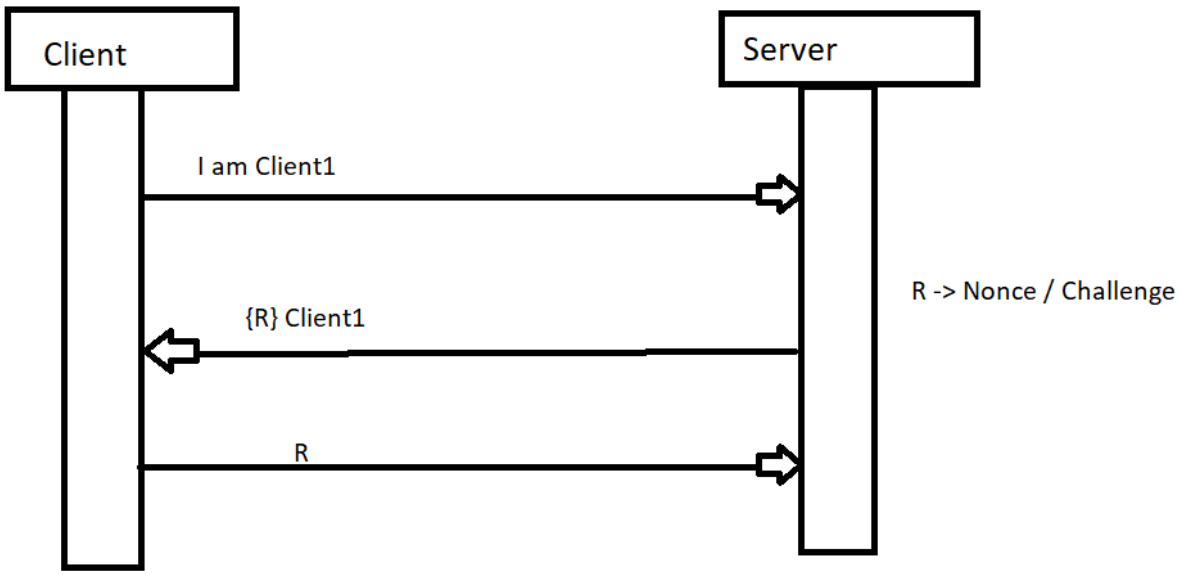
- Implemented using AES public key cryptography
- Client1 sends the encrypted message to Client2 during the communication
- On the receiver side, the encrypted message is decrypted to get back the original message

4. Integrity:

- Implemented using the HMAC-SHA256 algorithm
- Applying this, a message digest is created by combining the shared key between the two clients and a shared Initialization vector and using the SHA-256 function to encrypt it.
- The plaintext message is sent along with the hashed message to each side of the communication. The sender and receiver can check the authenticity of the sent message.

Message Flow:

1)Authentication:



- Message 1:
    - Client sends a message to Server by sharing its public key path and name
- Message 2:
    - Server encrypts a nonce with the public key and sends it to the client requested for authentication
- Message 3:
    - Client decrypts the nonce with its private key to get the nonce and sends it back to sever
- Message 4:
    - The server verifies the received nonce and sends the list of available clients as per the client's request to initiate communication

2)Confidentiality -

- Client1 creates a bit stream using the following steps
    - B = MD(K12 || IV) where IV is the initialization vector and MD is any hash function
    - Ciphertext, C = P XOR B, where P is the plain text and B is the hashed data that is generated
    - C (cipher-text) is encrypted with shared AES and is sent to client2
    - The mode of encryption used is AES EAX with the AES library in Python
    - Client1 sends this encrypted message to Client2

- Client2 will create an identical bit stream using the following
    - On the receiving client side, B = MD(K12 || IV)
    - We are able to get the plain text back by performing decryption as follows: P = C XOR B

In this way we are able to achieve confidentiality by using keyed hash functions.

3)Integrity

- The communication between both the clients has been implemented in a way to achieve integrity.
- Plaintext will be hashed using SHA-256 and sent along the message (plaintext). Upon receiving, the client will hash the plaintext and compare with the hash value present in the message.
- h(P) performs the integrity check.
- In Python, the library used to perform the hashing is Hashlib.
- Therefore, each message will consist of plaintext and hashed (plaintext) and be encrypted with the shared secret key between two clients.

4)Freshness

- B will send a nonce to A encrypting the nonce with the secret key shared between A and B K(AB).
- Then A will decrypt the nonce and generate a new key concatenating with K(AB) and nonce and with this A will encrypt the message and send it to B.
- As B already has the nonce and K(AB) it will concatenate and make the same key and decrypt the message.
- This way we will prevent the Replay attack as k=K(AB)||Nonce is a shared key only between A and B and C cannot come in the middle and forward it to B from A. In this way, we achieve freshness.

Possible Attacks:

Passive Attacks:

1. Eavesdropping: an attacker cannot obtain any valuable information since each message is encrypted using either a private key of clients or shared secret key between clients.

Active Attacks:

1. Reflection attack: Since there is only one server, reflection attack does not come into picture.
2. Replay attack: We must ensure to use a random number generator to generate random nonce R to prevent replay attacks.