EE6032 Project

Submitted to: Prof. Thomas Newe

Submitted By: Ansh Vaid

Student Id: 24010138

(Individual Project)

Index

1. Problem Statement

To design a protocol to establish a mutually agreed session key (K_{abc}) among three entities: A, B, and C, to enable secure communication within a group chat. Your design must meet the following requirements:

1. **No Direct Communication:** Entities A, B, and C will never communicate directly.
2. **Public Key Certificates:** Each entity (A, B, C, and S) holds a Public Key Certificate (e.g., CA<<A>>, CA<<B>>, CA<<C>>, and CA<<S>>).
3. **Server Access to Certificates:** The Chat Server (S) has access to the certificates of all entities using its service and can provide these certificates to service users upon request.
4. **Authentication:** Each entity must authenticate itself to the Chat Server (S) before using its service.
5. **Integrity and Authentication:** Every step in establishing the session key (K_{abc}) must include an authenticated integrity check of the transferred data. The protocol must include both how the data is generated and how it is verified.
6. **Cryptographic Algorithms:** For each step where CIA (Confidentiality, Integrity, or Authentication) is employed, specify the algorithm used for each function. This should be included both in the protocol description and in the program/implementation comments.
7. **Legend and Description:** Provide a legend to explain the notation used and a comprehensive description of each step in the protocol for establishing the shared key (K_{abc}).

2. Protocol Diagram and its Description

The protocol designed to solve the problem statement involves various steps that include various sequential steps that have to be performed for successful initiation of end-2-end encrypted chat room between all the clients, which is a part of protocol. To develop this protocol I have used Python programming using sockets. For certificates, which are used in public key encryption, I have generated X509 certificates having RSA-2048 and SHA 512 hash algorithm. For symmetric key encryption, I have used AES 128 bit key with PKCS7 padding and CBC mode, mutually generated session key for encryption and decryption. SHA-256 hashing algorithm is used in digital signatures and mutual session key generation.

2.1 Authentication Of Clients With Server

The initiation of communication from client to server, followed by the step by step authentication of both client and server is discussed using the protocol diagram and the respective steps of communication.

2.1.1 Authentication for client A

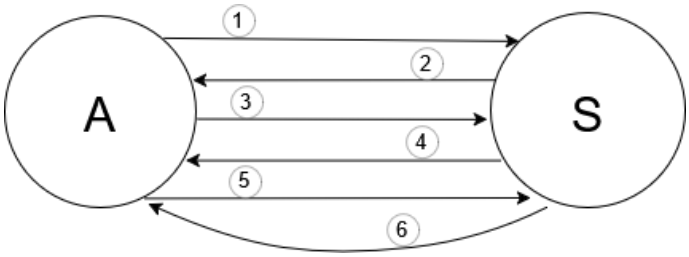

Figure 1: Authenticating client A and server S

1. **A->S**: A
   A sends identity to server S for initiation of connection

2. **S->A**: AUTH_REQUEST, $CERT_s$
   S sends Authentication Request to client A along with certificate of S containing its public key

3. **A->S**: $\{N_a\}_{Ks}$, $\{H(N_a)\}_{Ka}^{-1}$

   S: $\{\{N_a\}_{Ks}\}_{Ks}^{-1}$, $\{\{\{H(N_a)\}_{Ka}^{-1}\}_{Ka} == H(N_a)$

   A sends a random nonce to server S encrypted with public key of S for confidentiality, also send the digital signature of nonce to keep a check on integrity and authentication

S decrypts the nonce using its private key and then verifies the signature of the same by using the hash of decrypted nonce to check authenticity

4. **S->A**: $\{N_a+23\}_{Ka}$, $\{H(N_a+23)\}_{Ks}^{-1}$

   A: $\{\{N_a+23\}_{Ka}\}_{Ka}^{-1} == N_a+23$, $\{\{H(N_a+23)\}K_s^{-1}\}_{Ks} == H(N_a+23)$

   If the signature verifies with the decrypted nonce's hash then S adds 23 to the previous nonce and encrypts with public key of A for confidentiality, then creates the digital signature of the same to have integrity and authentication checks and send it to A
   A receives it and decrypts the encrypted nonce response using its private key and compares it if the operation is done on it or not, along with verification of the signature to check authenticity
   [At this point, A and S have authenticated each other with the use of digitally signed nonce from A to S and back S to A. Following steps occur ONLY if the authentication of both the parties is successful]

5. **A->S**: Ack
   A sends acknowledgement about signature and nonce matched successfully

6. **S->A**: [B, $CERT_b$], [C, $CERT_c$]

   S sends A the certificates of client B and C which is used in future mutual key generation

2.1.2 Authentication for client B

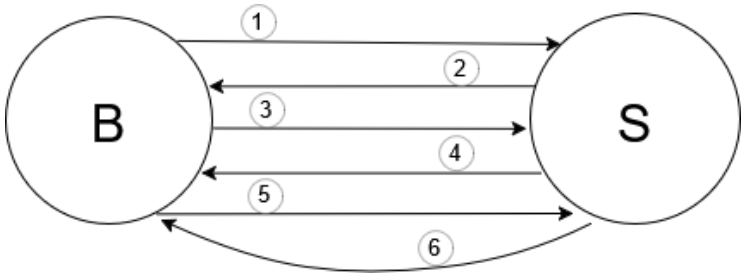

Figure 2: Authenticating client B and server S

1. **B->S**: B
   B sends identity to server S for initiation of connection

2. **S->B**: AUTH_REQUEST, $CERT_s$
   S sends Authentication Request to client B along with certificate of S containing its public key

3. **B->S**: $\{N_b\}_{Ks}$, $\{H(N_b)\}_{Kb}^{-1}$

   S: $\{\{N_b\}_{Ks}\}_{Ks}^{-1}$, $\{\{\{H(N_b)\}_{Kb}^{-1}\}_{Kb} == H(N_b)$

   B sends a random nonce to server S encrypted with public key of S for confidentiality, also send the digital signature of nonce for integrity and authentication checks

   S decrypts the nonce using its private key and then verifies the signature of the same by using the hash of decrypted nonce to check authenticity

4. **S->B**: $\{N_b+23\}_{Kb}$, $\{H(N_b+23)\}_{Ks}^{-1}$

   B: $\{\{N_b+23\}_{Kb}\}_{Kb}^{-1} == N_b+23$, $\{\{H(N_b+23)\}K_s^{-1}\}_{Ks} == H(N_b+23)$

   If the signature verifies with the decrypted nonce's hash then S adds 23 to the previous nonce and encrypts with public key of B for confidentiality, then creates the digital signature of the same for integrity and authentication checks and send it to B
   A receives it and decrypts the encrypted nonce response using its private key and compares it if the operation is done on it or not, along with verification of the signature to check authenticity
   [At this point, B and S have authenticated each other with the use of digitally signed nonce from B to S and back S to B. Following steps occur ONLY if the authentication of both the parties is successful]

5. **B->S**: Ack
   B sends acknowledgement about signature and nonce matched successfully

6. **S->B**: [A, $CERT_a$], [C, $CERT_c$]

S sends B the certificates of client A and C which is used in future mutual key generation

2.1.3 Authentication for client C

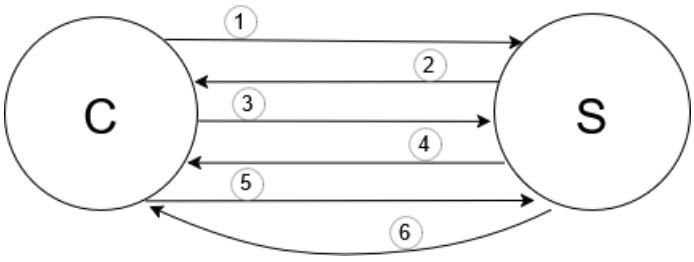

Figure 3: Authenticating client C and server S

1. **C->S**: C
   C sends identity to server S for initiation of connection

2. **S->C**: AUTH_REQUEST, $CERT_s$
   S sends Authentication Request to client C along with certificate of S containing its public key

3. **C->S**: $\{N_c\}_{Ks}$, $\{H(N_c)\}_{Kc}^{-1}$

   S: $\{\{N_c\}_{Ks}\}_{Ks}^{-1}$, $\{\{\{H(N_c)\}_{Kc}^{-1}\}_{Kc} == H(N_c)$

   C sends a random nonce to server S encrypted with public key of S, also send the digital signature of nonce

   S decrypts the nonce using its private key and then verifies the signature of the same by using the hash of decrypted nonce to check authenticity

4. **S->C**: $\{N_c+23\}_{Kc}$, $\{H(N_c+23)\}_{Ks}^{-1}$

   C: $\{\{N_c+23\}_{Kc}\}_{Kc}^{-1} == N_c+23$, $\{\{H(N_c+23)\}K_s^{-1}\}_{Ks} == H(N_c+23)$

   If the signature verifies with the decrypted nonce's hash then S adds 23 to the previous nonce and encrypts with public key of C for confidentiality, then creates the digital signature of the same and send it to C for integrity and authentication checks
   A receives it and decrypts the encrypted nonce response using its private key and compares it if the operation is done on it or not, along with verification of the signature to check authenticity
   [At this point, C and S have authenticated each other with the use of digitally signed nonce from C to S and back S to C. Following steps occur ONLY if the authentication of both the parties is successful]

5. **C->S**: Ack
   C sends acknowledgement about signature and nonce matched successfully

6. **S->C**: [A, $CERT_a$], [B, $CERT_b$]

   S sends C the certificates of client A and B which is used in future mutual key generation

Once all the clients authenticate with the server successfully, they move to the next phase, i.e. generation of mutual session key, where each client securely transfers a randomly generated number to other two clients with proper integrity and authenticity checks feature for the other two receivers. This part is discussed in detail in the next section.

2.2 Mutual Session Key Generation Between The Clients

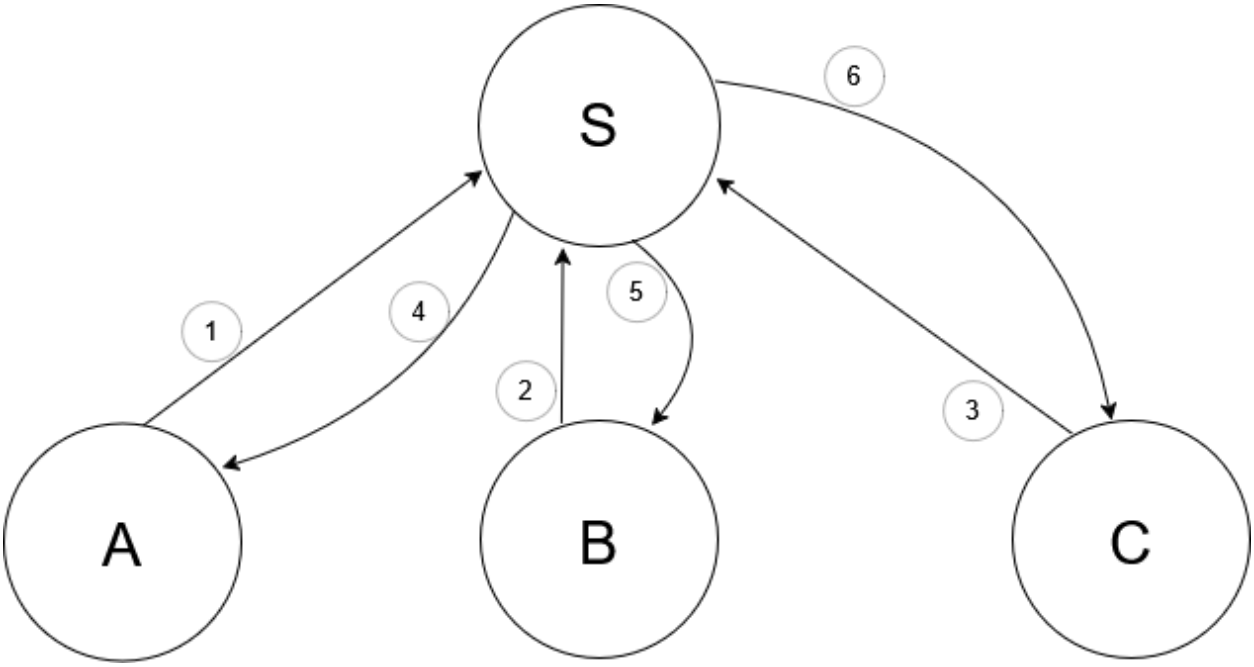

Figure 4: Sharing nonce and generating session key

1. **A->S**: [B,$\{N_a\}_{Kb}$], [C,$\{N_a\}_{Kc}$], $\{H(N_a)\}_{Ka}^{-1}$
   A sends S a randomly generated nonce (secret key part) encrypted using public key of B and C for confidentiality and also sends digitally signed nonce with it for integrity and authentication checks

2. **B->S**: [A,$\{N_b\}_{Ka}$], [C,$\{Nb\}_{Kc}$], $\{H(N_b)\}_{Kb}^{-1}$
   B sends S a randomly generated nonce (secret key part) encrypted using public key of A and C for confidentiality and also sends digitally signed nonce with it for integrity and authentication checks

3. **C->S**: [A,$\{N_c\}_{Ka}$], [B,$\{Nc\}_{Kb}$], $\{H(N_c)\}_{Kc}^{-1}$
   C sends S a randomly generated nonce (secret key part) encrypted using public key of A and B for confidentiality and also sends digitally signed nonce with it for integrity and authentication checks

4. **S->A**: [B, A, $\{N_b\}_{Ka}$, $\{H(N_b)\}_{Kb}^{-1}$], [C, A, $\{N_c\}_{Ka}$, $\{H(N_c)\}_{Kc}^{-1}$]
   A: [$\{\{N_b\}_{Ka}\}_{Ka}^{-1}$, $\{\{H(N_b)\}_{Kb}^{-1}\}_{Kb}$ == $H(N_b)$] ; [$\{\{N_c\}_{Ka}\}_{Ka}^{-1}$, $\{\{H(N_c)\}_{Kc}^{-1}\}_{Kc}$ == $H(N_c)$]

S sends the combination of digitally signed secret key and encrypted secret for the A from client B and C. A decrypts the nonce from B using its private key, verifies the signature of B using the public key of B and verifies the hash of nonce sent to it. Similarly, A decrypts the nonce from C using its private key, verifies the signature of C using the public key of C and verifies the hash of nonce sent to it.
Ascending order sorting(N_a, N_b, N_c) and SHA256(N_a || N_b || N_c) to generate mutual session key at client A

5. **S->B**: [A, B, $\{N_a\}_{Kb}$, $\{H(N_a)\}_{Ka^{-1}}$], [C, B, $\{N_c\}_{Kb}$, $\{H(N_c)\}_{Kc^{-1}}$]
B: [$\{\{N_a\}_{Kb}\}_{Kb^{-1}}$, $\{\{H(N_a)\}_{Ka^{-1}}\}_{Ka}$ == $H(N_a)$] ; [$\{\{N_c\}_{Ka}\}_{Kc^{-1}}$, $\{\{H(N_c)\}_{Kc^{-1}}\}_{Kc}$ == $H(N_c)$]
S sends the combination of digitally signed secret key and encrypted secret for the B from client A and C. B decrypts the nonce from A using its private key, verifies the signature of A using the public key of A and verifies the hash of nonce sent to it. Similarly, B decrypts the nonce from C using its private key, verifies the signature of C using the public key of C and verifies the hash of nonce sent to it.
Ascending order sorting(N_a, N_b, N_c) and SHA256(N_a || N_b || N_c) to generate mutual session key at client B

6. **S->C**: [A, C, $\{N_a\}_{Kc}$, $\{H(N_a)\}_{Ka^{-1}}$], [B, C, $\{N_b\}_{Kc}$, $\{H(N_b)\}_{Kb^{-1}}$]
C: [$\{\{N_a\}_{Kc}\}_{Kc^{-1}}$, $\{\{H(N_a)\}_{Ka^{-1}}\}_{Ka}$ == $H(N_a)$] ; [$\{\{N_b\}_{Kc}\}_{Kc^{-1}}$, $\{\{H(N_b)\}_{Kb^{-1}}\}_{Kb}$ == $H(N_b)$]
S sends the combination of digitally signed secret key and encrypted secret for the C from client A and B. A decrypts the nonce from A using its private key, verifies the signature of A using the public key of A and verifies the hash of nonce sent to it. Similarly, C decrypts the nonce from B using its private key, verifies the signature of B using the public key of B and verifies the hash of nonce sent to it.
Ascending order sorting(N_a, N_b, N_c) and SHA256(N_a || N_b || N_c) to generate mutual session key at client C

Therefore, session key is mutually generated as

K_{abc}= SHA256(N_a || N_b || N_c), if $N_a<N_b<N_c$;

or else **K_{abc}= SHA256(N_a || N_c || N_b)**, if $N_a<N_c<N_b$;

or else **K_{abc}= SHA256(N_b || N_c || N_a)**, if $N_b<N_c<N_a$;

or else **K_{abc}= SHA256(N_b || N_a || N_c)**, if $N_b<N_a<N_c$;

or else **K_{abc}= SHA256(N_c || N_b || N_a)**, if $N_c<N_b<N_a$;

or else **K_{abc}= SHA256(N_c || N_a || N_b)**, if $N_c<N_a<N_b$;

Once the mutual session key is generated, the server goes into relay mode and each client starts a sender and receiver thread to receive messages that are relayed by the server from any of the clients, and send the message to other clients relayed by the server. This part is discussed in detail in the next section.

2.3 Encrypted Communication Between Clients Relayed Via Server

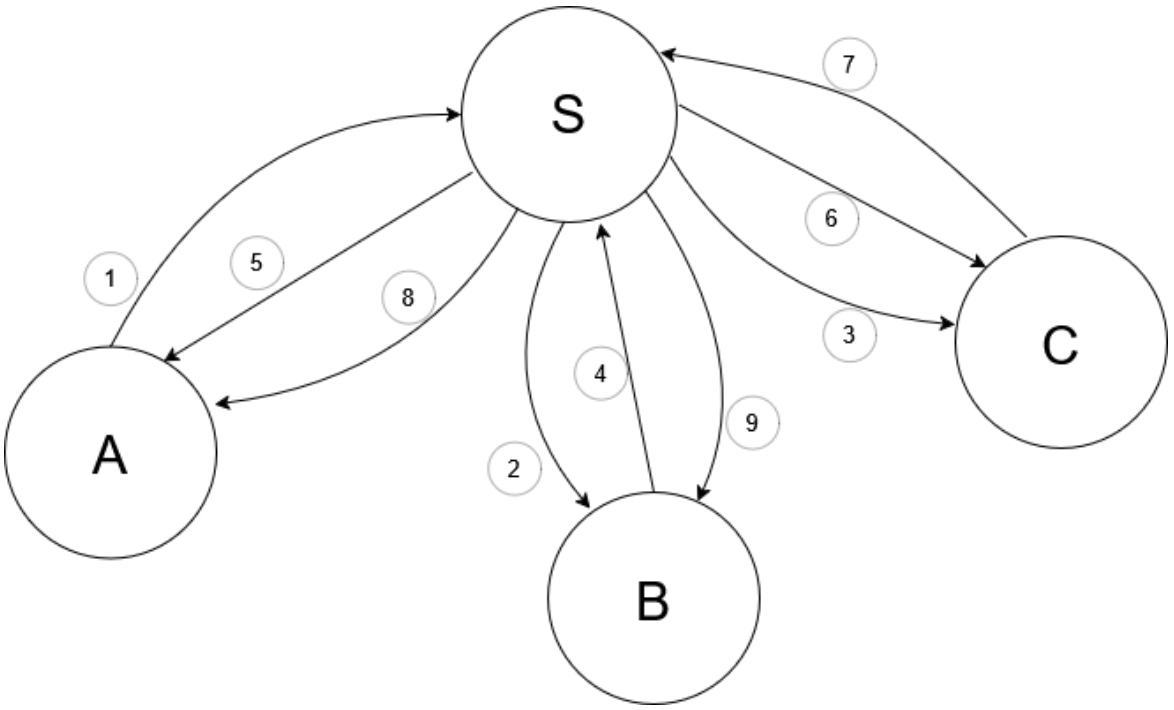

Figure 5: Session key encrypted communication between the clients through S

1. **A->S**: $\{M_1\}_{Kabc}$
   A sends the message encrypted using mutually generated session key K_{abc} to S

2. **S->B**: $\{M_1\}_{Kabc}$
   S relays the encrypted message to client B, sent from client A

3. **S->C**: $\{M_1\}_{Kabc}$
   S relays the encrypted message to client C, sent from client A

4. **B->S**: $\{M_2\}_{Kabc}$
   B sends the message encrypted using mutually generated session key K_{abc} to S

5. **S->A**: $\{M_2\}_{Kabc}$
   S relays the encrypted message to client A, sent from client B

6. **S->C**: $\{M_2\}_{Kabc}$
   S relays the encrypted message to client C, sent from client B

7. **C->S**: $\{M_3\}_{Kabc}$
   C sends the message encrypted using mutually generated session key K_{abc} to S

8. **S->A**: $\{M_3\}_{Kabc}$
   S relays the encrypted message to client A, sent from client C

9. **S->B**: $\{M_3\}_{Kabc}$
   S relays the encrypted message to client B, sent from client C

3. Legends

A= Entity A
B= Entity B
C= Entity C
S= Server S
$CERT_a$= Certificate of entity A
$CERT_b$= Certificate of entity B
$CERT_c$= Certificate of entity C
$CERT_s$= Certificate of server S
K_a= Public key of entity A
K_a^{-1}= Private key of entity A
K_b= Public key of entity B
K_b^{-1}= Private key of entity B
K_c= Public key of entity C
K_c^{-1}= Private key of entity C
K_s= Public key of server S
K_s^{-1}= Private key of server S
N_a= Nonce generated by entity A
N_b= Nonce generated by entity B
N_c= Nonce generated by entity C
K_{abc}= Mutually generated session key
M_1= Message 1
M_2= Message 2
M_3= Message 3
H()= Hash function
->= Send/Transmission/Transfer
{}= Encryption/Decryption