Secure Multi-Party Chat System

# 📄 Project Description

This project implements a Secure Multi-Party Chat protocol where Clients (A, B, and C) establish a secure session key among themselves using a trusted Server (S).

Each Client exchanges encrypted nonces, authenticates using digital signatures (RSA-PSS), and derives a shared session key (Kabc) using HKDF over collected nonces. Clients then chat securely using AES-GCM encryption.

The Server acts purely as a relay and cannot decrypt any message or nonce.

# 🛠️ Technologies Used

- Python 3

- cryptography library

- sockets for communication

- JSON for structured messaging

- RSA-PSS for digital signatures

- AES-GCM for authenticated encryption

- HKDF-SHA256 for key derivation

- Mermaid.js for protocol diagrams

# 🔒 Security Concepts Implemented

Confidentiality: Nonces encrypted with recipient's public key

Authentication: RSA-PSS signature on nonce exchanges

Integrity: Signature verification on all messages

Perfect Forward Secrecy: Derived fresh session keys each run

Server Blindness: Server cannot decrypt any payload

# 👉 Protocol Sequence Diagram

(Insert your Mermaid sequence diagram here or link to the generated image.)

Legend:

{Nx}Kx = Nonce encrypted with public key of X

SIG\_X = RSA-PSS signature by X

Kx = RSA public key, Kx⁻¹ = RSA private key

HKDF(...) = Key derived from H(Na||Nb||Nc)

AES-GCM(Kabc, Msg) = Authenticated encryption of chat messages

# 🤔 How Protocol Works

Key Exchange:

- Clients generate fresh nonces (Na, Nb, Nc).

- Encrypt nonces using peers' public keys.

- Sign nonce and participant IDs.

Session Key Derivation:

- Each Client decrypts received nonces.

- Verifies signatures.

- Derives Kabc using HKDF over all nonces.

Secure Chat Messaging:

- Clients encrypt messages using AES-GCM with Kabc.

- Server forwards encrypted messages without decrypting.

# 🔢 Detailed Verification Process

During the key exchange phase, each client needs to verify the authenticity and integrity of the nonces received from peers.

Step-by-step Verification:

1. Decrypt Received Nonce:

- Decrypt using the client's own private key (K\_self⁻¹).

2. Recalculate Hash Input:

- Combine peer\_nonce || sorted(participant IDs).

3. Verify Signature:

- Verify with the peer’s public key (K\_peer) using RSA-PSS and SHA-256.

4. Keys Used:

- Decryption: Client's private key (K\_self⁻¹)

- Verification: Peer’s public key (K\_peer)

5. Purpose:

- Ensure authenticity and integrity. Discard if verification fails.

# 📂 Project Structure

server.py: Server handling certificate requests and message relay

client\_A.py, client\_B.py, client\_C.py: Client programs

key\_exchange.py: Nonce generation, signing, verification

secure\_chat.py: AES-GCM encryption/decryption

key\_manager.py: Certificate loading

auth\_manager.py: Certificate authority simulation (optional)

Certs/: Certificates for each entity

# 🚀 How to Run

1. Install dependencies:

pip install cryptography

2. Run the Server:

python server.py

3. Start each Client in a separate terminal:

python client\_A.py

python client\_B.py

python client\_C.py

4. Chat securely!

# 📄 Certificates

Each Client (A, B, C) and Server (S) has a valid X.509 certificate. Server can supply peer certificates on request during key exchange.

# ✨ Credits

Project Lead: (Your Name)

Special thanks: ChatGPT assistance for explanations and code structure improvements.