Information Security

SEMESTER PROJECT

Submitted To:

Dr. Hina Ayaz

Submitted By:

Waheed Gulzar i222526

Tayyab Attiq i222554

Shafiq Ullah i222556

Haris Khan i222673

Semester Fall 2025-SE-A

SoC - Department of Software Engineering

National University of Computer & Emerging Sciences, Islamabad

Table of content:

End-to-End Encrypted Messaging System (e2ee)

1. Introduction

This report details the design and implementation of **e2eeV1**, a secure messaging and file-sharing application designed to prioritize user privacy. Unlike traditional messaging systems where service providers retain access to user data, e2eeV1 ensures that the server acts strictly as a blind relay. By utilizing a custom Authenticated Key Exchange (AKE) protocol and robust cryptographic primitives (ECDH, ECDSA, AES-GCM), the system guarantees confidentiality, integrity, and authenticity, effectively mitigating risks associated with server-side breaches and network eavesdropping.

2. Problem Statement

In the current landscape of digital communication, users face significant privacy risks. Centralized service providers often hold decryption keys, creating single points of failure. If a provider is compromised or coerced, user data is exposed.

To address this, the e2eeV1 system was built to satisfy the following critical security requirements:

- **Confidentiality:** Data must remain opaque to all parties except the intended recipient (including the server).
- **Integrity:** Any tampering with the message payload must be detectable.
- **Identity Verification:** Users must be cryptographically bound to their messages to prevent impersonation.
- **Forward Secrecy:** Compromise of long-term keys should not expose past session data.

3. System Architecture

3.1 High-Level Design

The system operates on a client-server model. The server is trusted only for availability (routing messages) and storage (holding encrypted blobs), but not for confidentiality.

- **Clients:** React-based Single Page Applications (SPA) that handle all cryptographic operations (key generation, encryption, decryption) locally within the browser using the Web Crypto API.
- **Server:** A Node.js/Express environment that facilitates WebSocket connections for real-time delivery and connects to MongoDB for persistent storage of encrypted data.

3.2 Data Schema

The database schema is designed to store minimum necessary metadata while keeping payloads opaque.

Schema	Key Fields	Purpose
User	publicKeys (Identity & Pre-Key)	Stores public verification keys; private keys never leave the client.
Message	ciphertext, iv, aad, seq	Stores the encrypted payload and the metadata required for replay protection.
Log	event, ts, details	A tamper-evident audit trail for security events (e.g., REPLAY_DETECTED).

4. Threat Model (STRIDE Analysis)

We utilized the STRIDE methodology to analyze potential threats and implement specific defenses.

- **Spoofing:** To prevent an attacker from impersonating "Alice," all handshake parameters are signed using **ECDSA**. Public keys are strictly verified against the server's registry.
- **Tampering:** To ensure message integrity, the system uses **AES-GCM**, an authenticated encryption mode. Additionally, the handshake transcript is verified via **HMAC** to prevent man-in-the-middle tampering during session establishment.
- **Repudiation:** Digital signatures on the handshake provide non-repudiation; a user cannot deny establishing a session signed by their private Identity Key.
- **Information Disclosure:** Addressed via **E2EE (AES-256-GCM)**. The server only sees ciphertext.
- **Denial of Service:** The client enforces strict checks on Sequence Numbers (seq) and Timestamps (ts) to reject replayed or old messages instantly.
- **Elevation of Privilege:** API access is controlled via **JWT (JSON Web Tokens)**, ensuring only authenticated users can post data.

5. Cryptographic Specifications

The system implements a hybrid cryptosystem using the standard NIST curves and algorithms:

- **Key Agreement:** Elliptic Curve Diffie-Hellman (**ECDH**) on Curve P-256.
- **Signatures: ECDSA** (Curve P-256, SHA-256) for authentication.
- **Symmetric Encryption: AES-GCM** with 256-bit keys for message payloads.
- **Key Derivation: HKDF** (SHA-256) for deriving Session, Confirmation, and File keys.

6. Key Exchange Protocol

The core of the system's security is a custom 3-pass Authenticated Key Exchange (AKE). This ensures that two users can establish a shared secret that is both fresh (via nonces) and authenticated (via signatures).

The Protocol Flow:

1. Initiation (Alice $\rightarrow$ Bob): Alice generates an ephemeral key pair (Epk_A) and a nonce. She signs these parameters.
   $$Message_1 = \{ Epk_A, Nonce_A, Ts_A, Sign_A(Epk_A, Nonce_A) \}$$
2. Response (Bob $\rightarrow$ Alice): Bob verifies Alice's signature. He generates his own ephemeral key (Epk_B) and nonce, computes the shared secret via ECDH, and generates a MAC over the transcript to confirm he possesses the key.
   $$Message_2 = \{ Epk_B, Nonce_B, Ts_B, Sign_B(...), MAC_{Bob}(Transcript) \}$$
3. Confirmation (Alice $\rightarrow$ Bob): Alice verifies Bob's signature and MAC. She computes the final MAC to confirm the session to Bob.
   $$Message_3 = \{ MAC_{Alice}(Transcript) \}$$

7. Encryption Workflows

7.1 Messaging Workflow

To prevent replay attacks and binding issues, the encryption process binds metadata to the ciphertext.

1. Construction: The sender constructs Additional Authenticated Data (AAD):
   $$AAD = senderId \ || \ receiverId \ || \ sessionId \ || \ seq \ || \ ts$$
2. Encryption: The ciphertext is generated using the Session Key (K_{sess}) and a random 96-bit IV.
   $$C = AES\text{-}GCM\text{-}Encrypt(K_{sess}, IV, Plaintext, AAD)$$
3. **Transmission:** The bundle $\{C, IV, AAD, seq, ts\}$ is sent to the server.

7.2 File Sharing Workflow

Large files are handled efficiently via chunking:

1. **Key Derivation:** A unique $FileKey$ is derived from the master $SessionKey$ to segregate file security from message security.
2. **Chunking:** Files are split into 64KB parts.
3. **Encryption:** Each chunk is encrypted individually. This allows for parallel uploads/downloads and partial retries without re-encrypting the whole file.

8. Security Auditing & Resilience

The system includes active defense mechanisms and logging.

Attack Demonstrations

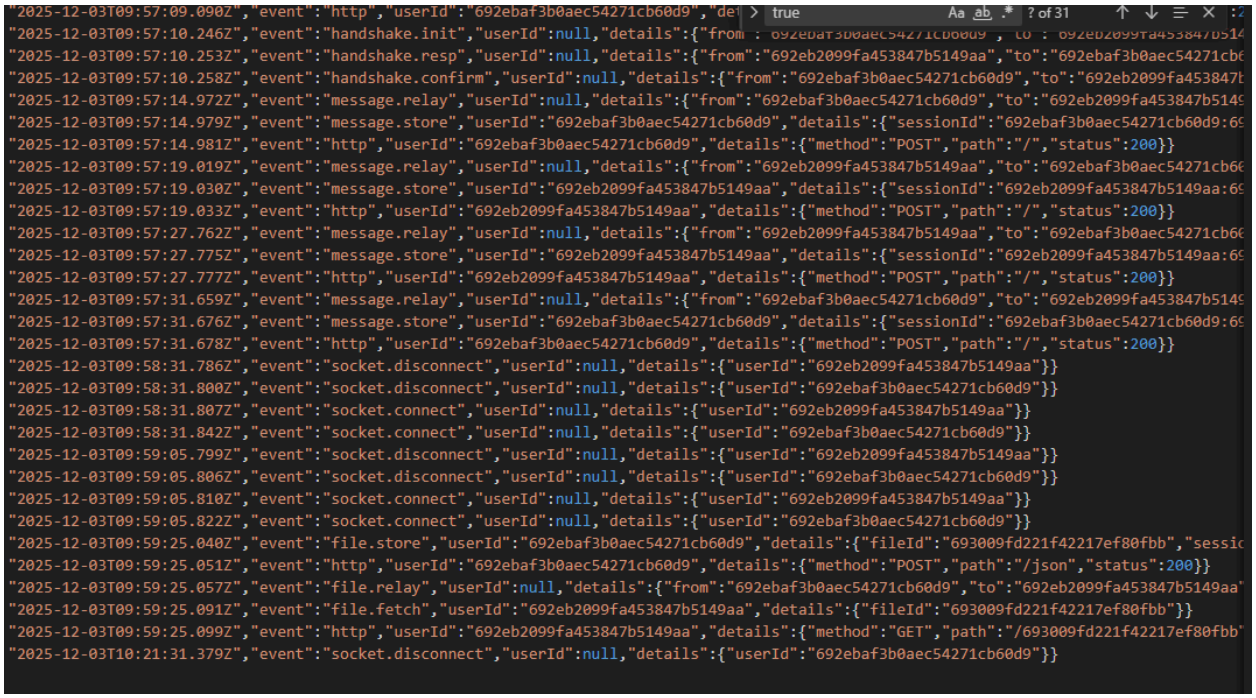
The project codebase includes proof-of-concept scripts to verify defenses:

- **MITM Defense:** The attacks/mitm_proxy.js script attempts to inject a rogue public key during the handshake. The client successfully rejects this because the attacker cannot forge the ECDSA signature corresponding to the user's Identity Key.
- **Replay Defense:** The attacks/replay_client.js script captures valid encrypted packets and re-transmits them. The receiver's logic detects the duplicate seq number and rejects the packet, logging a REPLAY_ATTACK event.

9. Conclusion

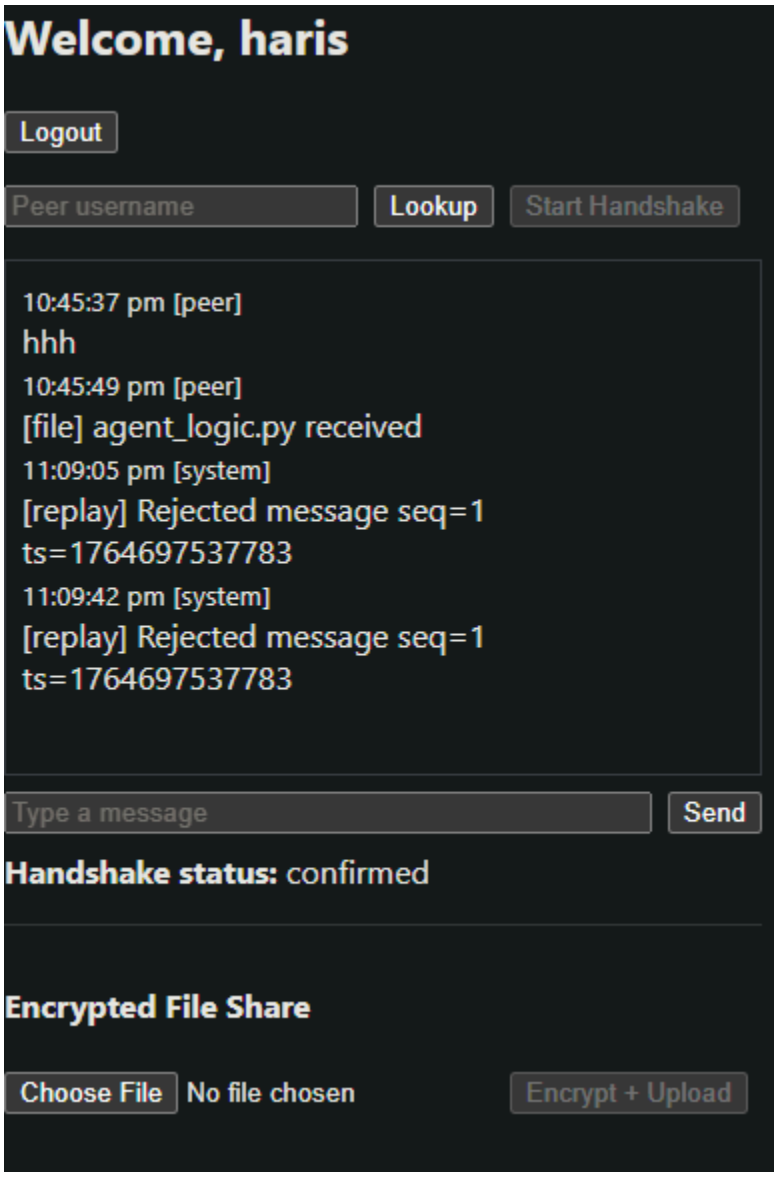
The **e2eeV1** system successfully demonstrates a secure, modern approach to private communication. By combining ECDH for forward secrecy, ECDSA for strong authentication, and AES-GCM for tamper-proof encryption, the system meets the high-security standards required for confidential data exchange in hostile network environments.

Logging & Security Auditing

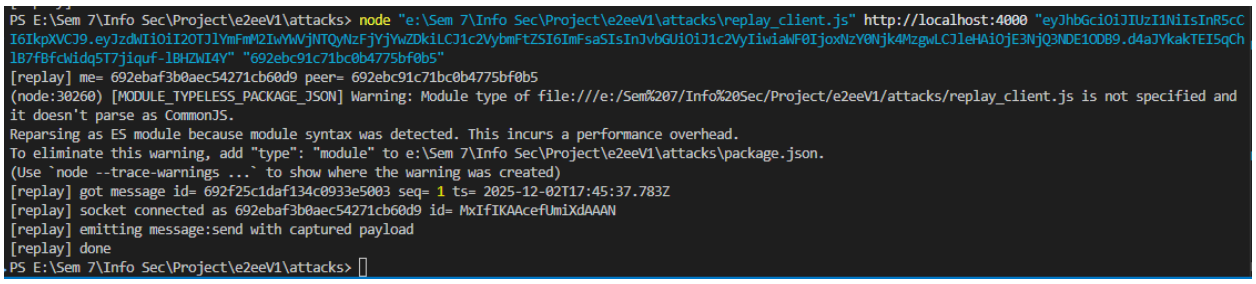


```
                                                    react-dom.development.js:29895
Download the React DevTools for a better development experience:
https://reactjs.org/link/react-devtools
[socket] creating socket for user 692ebaf3b0aec54271cb60d9          Chat.jsx:44
[socket] connected, id= em1axRyrJmezEZl8AAAz                        Chat.jsx:60
[kex] looking up peer tayyab                                       Chat.jsx:81
[kex] lookup result                                                Chat.jsx:83
▶ {userId: '692eb2099fa453847b5149aa', username: 'tayyab', publicKeys: {…}}
[kex] starting handshake with                                      Chat.jsx:87
▶ {userId: '692eb2099fa453847b5149aa', username: 'tayyab', publicKeys: {…}}
[kex] startHandshake                                               Chat.jsx:90
[kex] meId: 692ebaf3b0aec54271cb60d9                               Chat.jsx:92
[kex] peerInfo:                                                    Chat.jsx:99
▶ {userId: '692eb2099fa453847b5149aa', username: 'tayyab', publicKeys: {…}}
[kex] startHandshake ->                                            Chat.jsx:101
▶ {meId: '692ebaf3b0aec54271cb60d9', peerId: '692eb2099fa453847b5149aa'}
[kex:init] emit to 692eb2099fa453847b5149aa                        Chat.jsx:134
[kex:resp] received                                                Chat.jsx:137
▶ {fromUserId: '692eb2099fa453847b5149aa', data: {…}}
[kex:confirm] sending macA to 692eb2099fa453847b5149aa             Chat.jsx:205
[msg] deliver                                                      Chat.jsx:452
▶ {fromUserId: '692eb2099fa453847b5149aa', toUserId: '692ebaf3b0aec54271cb6
  0d9', seq: 1, ts: 1764755839018, iv: '1Jj6fGoT9QDZXuRN', …}
[msg] deliver                                                      Chat.jsx:452
▶ {fromUserId: '692eb2099fa453847b5149aa', toUserId: '692ebaf3b0aec54271cb6
  0d9', seq: 2, ts: 1764755847761, iv: 'lwWRFjP8IJNgbPZQ', …}
[socket] creating socket for user 692ebaf3b0aec54271cb60d9          Chat.jsx:22
[socket] connected, id= xRl1mpRfPg6g7ROuAAA3                        Chat.jsx:35
[socket] creating socket for user 692ebaf3b0aec54271cb60d9          Chat.jsx:22
[socket] connected, id= 2tQJuU6RmiMmKpURAAA7                       Chat.jsx:35
[file] encrypt+upload start                                  FileShare.jsx:13
▶ {name: 'bmw_car_sports_139454_1920x1080.jpg', size: 363747, type: 'image/
  jpeg'}
[file] encrypted chunk 0 / 6                                 FileShare.jsx:26
[file] upload complete, fileId 693009fd221f42217ef80fbb      FileShare.jsx:30
[file] notifying peer via socket                             FileShare.jsx:34
▶ {to: '692eb2099fa453847b5149aa', fileId: '693009fd221f42217ef80fbb'}
[socket] disconnected: transport close                             Chat.jsx:41
[socket] connected, id= 1Jc-mH8RK7ZBZjR9AAA9                       Chat.jsx:35
>
```

Attack



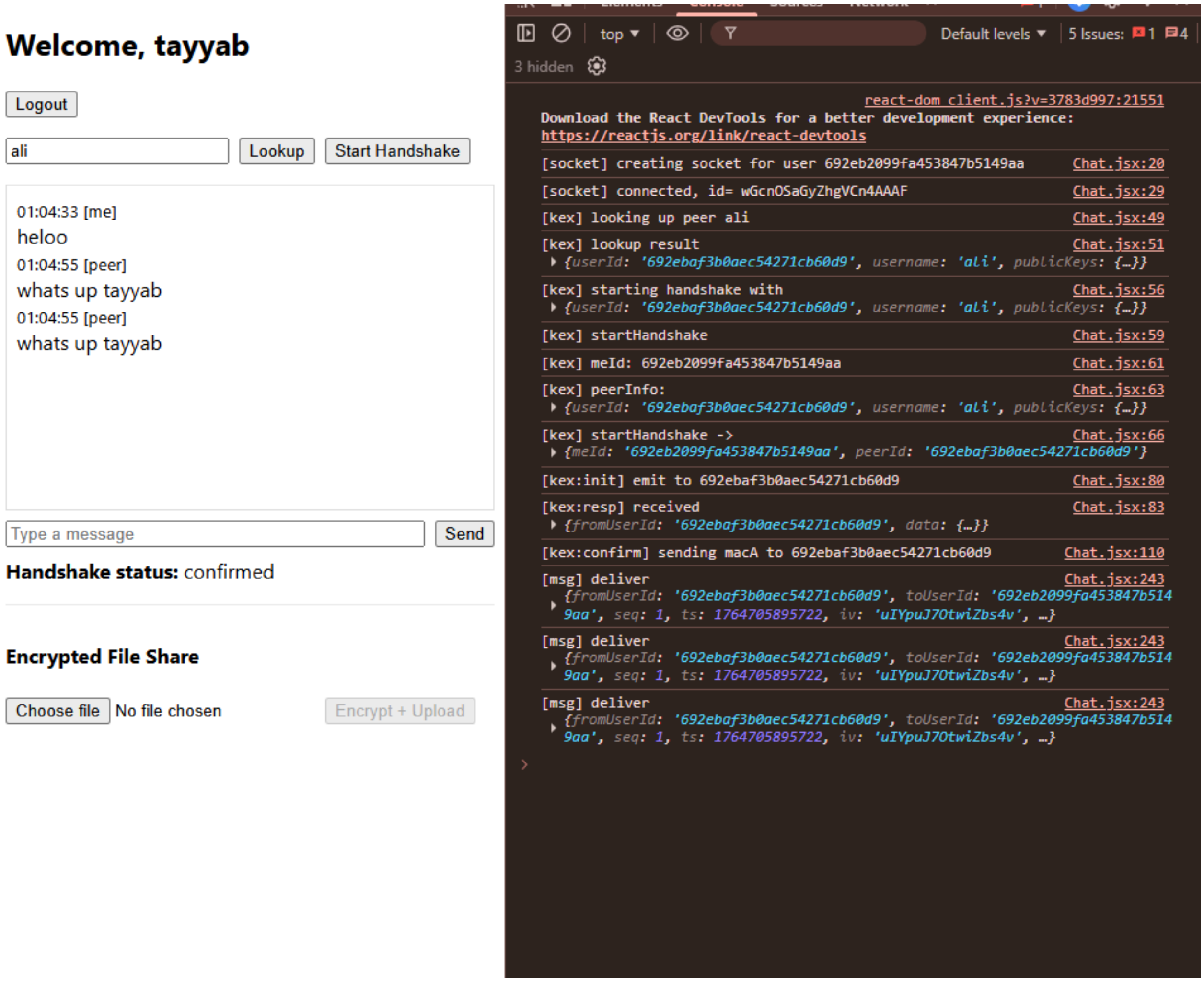
attack



Proxy

```
[mitm] client connected {
  userId: '692eb2099fa453847b5149aa',
  clientId: 'egVnzX09G8RjuIi2AAAC'
}
[mitm] client connected {
  userId: '692ebaf3b0aec54271cb60d9',
  clientId: '8qcv0B3rrOsf61ilAAAD'
}
[mitm] upstream connected { userId: '692eb2099fa453847b5149aa' }
[mitm] upstream connected { userId: '692ebaf3b0aec54271cb60d9' }
[mitm] upstream message:send relay
[mitm] downstream message:deliver
[mitm] upstream message:send relay
[mitm] downstream message:deliver
```

```
}
[mitm] upstream connected { userId: '692ebaf3b0aec54271cb60d9' }
[mitm] upstream connected { userId: '692eb2099fa453847b5149aa' }
[mitm] upstream disconnected { userId: '692eb2099fa453847b5149aa', r: 'io client disconnect' }
[mitm] client disconnected { userId: '692eb2099fa453847b5149aa' }
[mitm] upstream disconnected { userId: '692ebaf3b0aec54271cb60d9', r: 'io client disconnect' }
[mitm] client disconnected { userId: '692ebaf3b0aec54271cb60d9' }
[mitm] client connected {
  userId: '692eb2099fa453847b5149aa',
  clientId: 'wGcnOSaGyZhgVCn4AAAF'
}
[mitm] upstream connected { userId: '692eb2099fa453847b5149aa' }
[mitm] client connected {
  userId: '692ebaf3b0aec54271cb60d9',
  clientId: 'mQ-1kQ1j4O_Pr7t6AAAH'
}
[mitm] upstream connected { userId: '692ebaf3b0aec54271cb60d9' }
[mitm] upstream handshake:init emit { to: '692ebaf3b0aec54271cb60d9' }
[mitm] downstream handshake:init { from: '692eb2099fa453847b5149aa' }
[mitm] upstream handshake:resp emit { to: '692eb2099fa453847b5149aa' }
[mitm] downstream handshake:resp { from: '692ebaf3b0aec54271cb60d9' }
[mitm] precomputed keys for both legs (confirm+session)
[mitm] recomputed macB for initiator leg and cached keys
[mitm] upstream handshake:confirm emit { to: '692ebaf3b0aec54271cb60d9' }
[mitm] rewrote macA for responder leg
[mitm] downstream handshake:confirm { from: '692eb2099fa453847b5149aa' }
[mitm] upstream message:send relay
[mitm] message decrypted with first key (expected leg)
[mitm] message plaintext: {"text":"heloo","ts":1764705873002,"seq":1}
[mitm] downstream message:deliver
[mitm] upstream message:send relay
[mitm] message decrypted with first key (expected leg)
[mitm] message plaintext: {"text":"whats up tayyab","ts":1764705895722,"seq":1}
[mitm] downstream message:deliver
```


Welcome, tayyab
Logout
ali
Lookup
Start Handshake
01:04:33 [me]
heloo
01:04:55 [peer]
whats up tayyab
01:04:55 [peer]
whats up tayyab
Type a message
Send
Handshake status: confirmed
Encrypted File Share
Choose file
No file chosen
Encrypt + Upload
top
Default levels
5 Issues: 1 4
3 hidden
react-dom_client.js?v=3783d997:21551
Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools
[socket] creating socket for user 692eb2099fa453847b5149aa Chat.jsx:20
[socket] connected, id= wGcnOSaGyZhgVCn4AAAF Chat.jsx:29
[kex] looking up peer ali Chat.jsx:49
[kex] lookup result Chat.jsx:51
{userId: '692ebaf3b0aec54271cb60d9', username: 'ali', publicKeys: {…}}
[kex] starting handshake with Chat.jsx:56
{userId: '692ebaf3b0aec54271cb60d9', username: 'ali', publicKeys: {…}}
[kex] startHandshake Chat.jsx:59
[kex] meId: 692eb2099fa453847b5149aa Chat.jsx:61
[kex] peerInfo: Chat.jsx:63
{userId: '692ebaf3b0aec54271cb60d9', username: 'ali', publicKeys: {…}}
[kex] startHandshake -> Chat.jsx:66
{meId: '692eb2099fa453847b5149aa', peerId: '692ebaf3b0aec54271cb60d9'}
[kex:init] emit to 692ebaf3b0aec54271cb60d9 Chat.jsx:80
[kex:resp] received Chat.jsx:83
{fromUserId: '692ebaf3b0aec54271cb60d9', data: {…}}
[kex:confirm] sending macA to 692ebaf3b0aec54271cb60d9 Chat.jsx:110
[msg] deliver Chat.jsx:243
{fromUserId: '692ebaf3b0aec54271cb60d9', toUserId: '692eb2099fa453847b5149aa', seq: 1, ts: 1764705895722, iv: 'uIYpuJ7OtwiZbs4v', …}
[msg] deliver Chat.jsx:243
{fromUserId: '692ebaf3b0aec54271cb60d9', toUserId: '692eb2099fa453847b5149aa', seq: 1, ts: 1764705895722, iv: 'uIYpuJ7OtwiZbs4v', …}
[msg] deliver Chat.jsx:243
{fromUserId: '692ebaf3b0aec54271cb60d9', toUserId: '692eb2099fa453847b5149aa', seq: 1, ts: 1764705895722, iv: 'uIYpuJ7OtwiZbs4v', …}

Welcome, tayyab

Logout

ali Lookup Start Handshake

01:04:33 [me]
heloo
01:04:55 [peer]
whats up tayyab
01:04:55 [peer]
whats up tayyab

Type a message Send

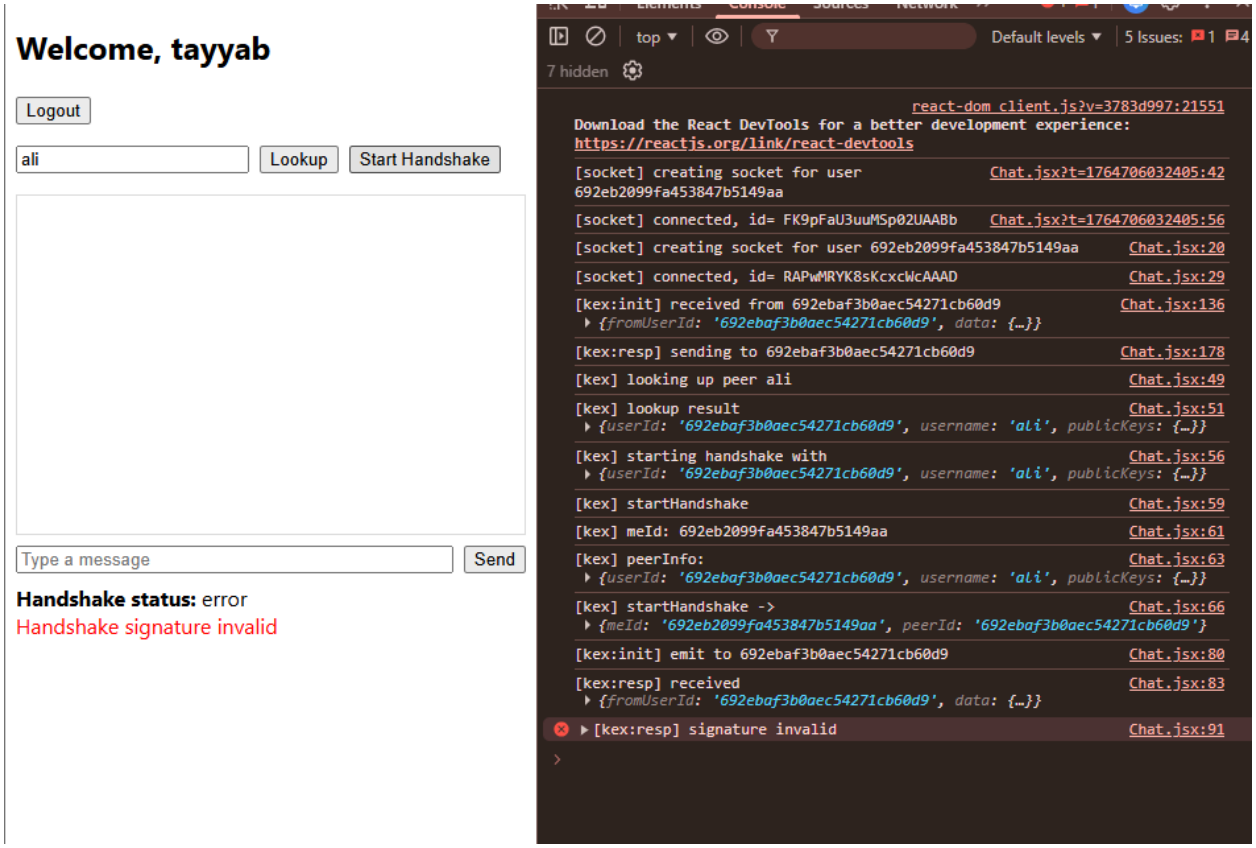
Handshake status: confirmed

Encrypted File Share

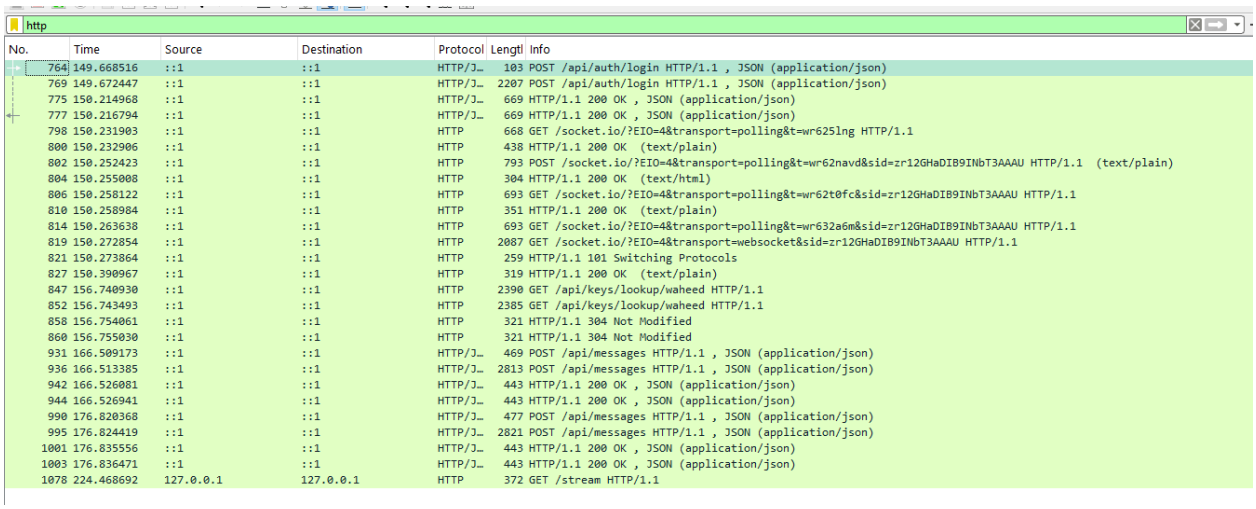
Choose file No file chosen Encrypt + Upload

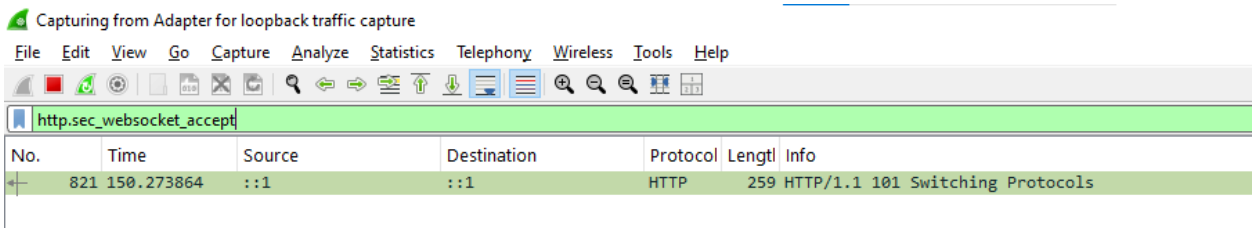
Proxy encrypt

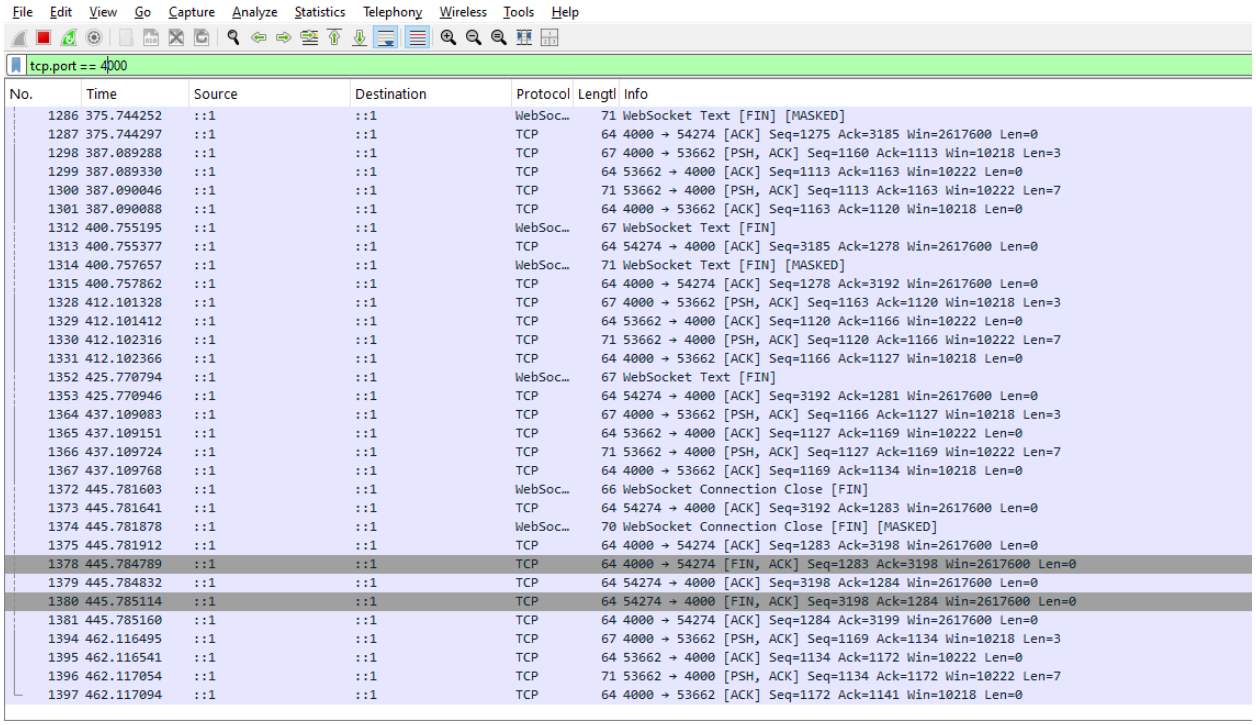
```
[mitm] client connected {
  userId: '692ebaf3b0aec54271cb60d9',
  clientId: 'OmuGrxL7DPFI5IRkAAAC'
}
[mitm] client connected {
  userId: '692eb2099fa453847b5149aa',
  clientId: 'RAPwMRYK8sKcxcWcAAAD'
}
[mitm] upstream connected { userId: '692ebaf3b0aec54271cb60d9' }
[mitm] upstream connected { userId: '692eb2099fa453847b5149aa' }
[mitm] upstream handshake:init emit { to: '692eb2099fa453847b5149aa' }
[mitm] downstream handshake:init { from: '692ebaf3b0aec54271cb60d9' }
[mitm] upstream handshake:resp emit { to: '692ebaf3b0aec54271cb60d9' }
[mitm] downstream handshake:resp { from: '692eb2099fa453847b5149aa' }
[mitm] precomputed keys for both legs (confirm+session)
[mitm] recomputed macB for initiator leg and cached keys
[mitm] upstream handshake:init emit { to: '692ebaf3b0aec54271cb60d9' }
[mitm] downstream handshake:init { from: '692eb2099fa453847b5149aa' }
[mitm] upstream handshake:resp emit { to: '692eb2099fa453847b5149aa' }
[mitm] downstream handshake:resp { from: '692ebaf3b0aec54271cb60d9' }
[mitm] precomputed keys for both legs (confirm+session)
[mitm] recomputed macB for initiator leg and cached keys
```

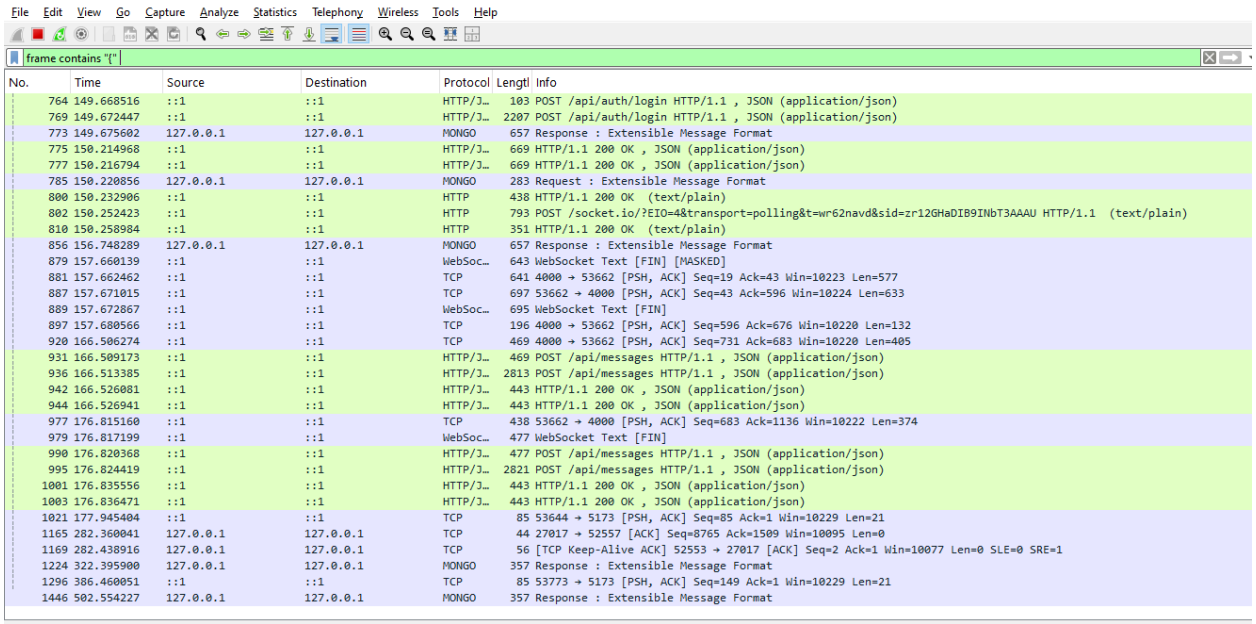


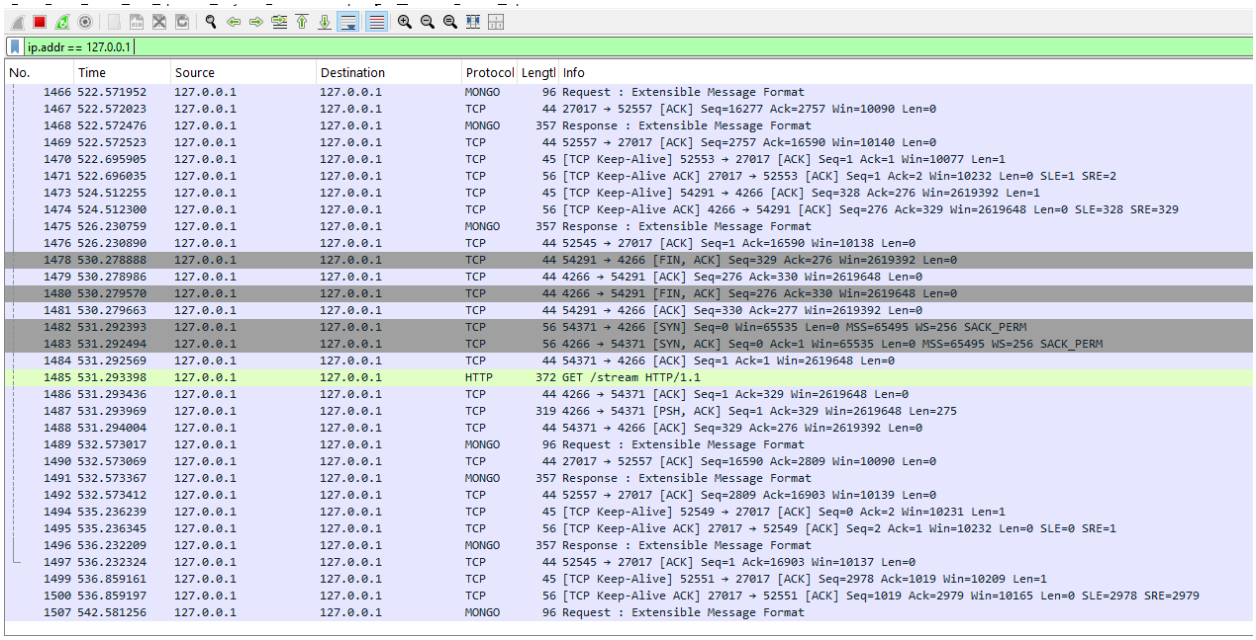
Analysis using wireshark:




Capturing from Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http.sec_websocket_accept
No. Time Source Destination Protocol Lengtl Info
821 150.273864 ::1 ::1 HTTP 259 HTTP/1.1 101 Switching Protocols

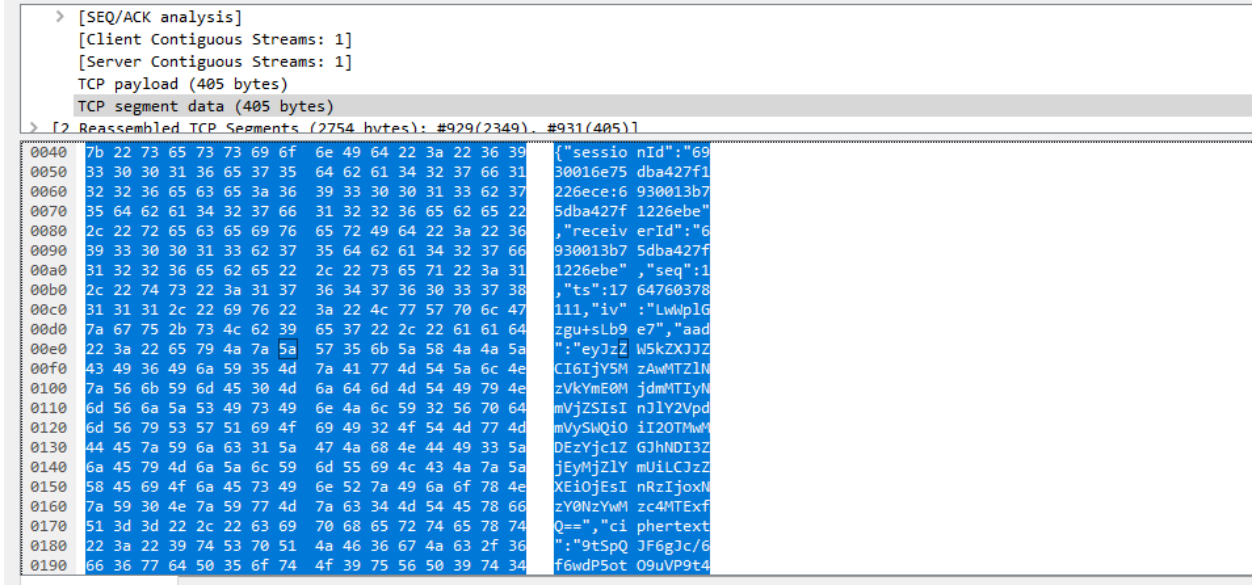

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.port == 4000
No. Time Source Destination Protocol Lengtl Info
1286 375.744252 ::1 ::1 WebSoc… 71 WebSocket Text [FIN] [MASKED]
1287 375.744297 ::1 ::1 TCP 64 4000 → 54274 [ACK] Seq=1275 Ack=3185 Win=2617600 Len=0
1298 387.089288 ::1 ::1 TCP 67 4000 → 53662 [PSH, ACK] Seq=1160 Ack=1113 Win=10218 Len=3
1299 387.089330 ::1 ::1 TCP 64 53662 → 4000 [ACK] Seq=1113 Ack=1163 Win=10222 Len=0
1300 387.090046 ::1 ::1 TCP 71 53662 → 4000 [PSH, ACK] Seq=1113 Ack=1163 Win=10222 Len=7
1301 387.090088 ::1 ::1 TCP 64 4000 → 53662 [ACK] Seq=1163 Ack=1120 Win=10218 Len=0
1312 400.755195 ::1 ::1 WebSoc… 67 WebSocket Text [FIN]
1313 400.755377 ::1 ::1 TCP 64 54274 → 4000 [ACK] Seq=3185 Ack=1278 Win=2617600 Len=0
1314 400.757657 ::1 ::1 WebSoc… 71 WebSocket Text [FIN] [MASKED]
1315 400.757862 ::1 ::1 TCP 64 4000 → 54274 [ACK] Seq=1278 Ack=3192 Win=2617600 Len=0
1328 412.101328 ::1 ::1 TCP 67 4000 → 53662 [PSH, ACK] Seq=1163 Ack=1120 Win=10218 Len=3
1329 412.101412 ::1 ::1 TCP 64 53662 → 4000 [ACK] Seq=1120 Ack=1166 Win=10222 Len=0
1330 412.102316 ::1 ::1 TCP 71 53662 → 4000 [PSH, ACK] Seq=1120 Ack=1166 Win=10222 Len=7
1331 412.102366 ::1 ::1 TCP 64 4000 → 53662 [ACK] Seq=1166 Ack=1127 Win=10218 Len=0
1352 425.770794 ::1 ::1 WebSoc… 67 WebSocket Text [FIN]
1353 425.770946 ::1 ::1 TCP 64 54274 → 4000 [ACK] Seq=3192 Ack=1281 Win=2617600 Len=0
1364 437.109083 ::1 ::1 TCP 67 4000 → 53662 [PSH, ACK] Seq=1166 Ack=1127 Win=10218 Len=3
1365 437.109151 ::1 ::1 TCP 64 53662 → 4000 [ACK] Seq=1127 Ack=1169 Win=10222 Len=0
1366 437.109724 ::1 ::1 TCP 71 53662 → 4000 [PSH, ACK] Seq=1127 Ack=1169 Win=10222 Len=7
1367 437.109768 ::1 ::1 TCP 64 4000 → 53662 [ACK] Seq=1169 Ack=1134 Win=10218 Len=0
1372 445.781603 ::1 ::1 WebSoc… 66 WebSocket Connection Close [FIN]
1373 445.781641 ::1 ::1 TCP 64 54274 → 4000 [ACK] Seq=3192 Ack=1283 Win=2617600 Len=0
1374 445.781878 ::1 ::1 WebSoc… 70 WebSocket Connection Close [FIN] [MASKED]
1375 445.781912 ::1 ::1 TCP 64 4000 → 54274 [ACK] Seq=1283 Ack=3198 Win=2617600 Len=0
1378 445.784789 ::1 ::1 TCP 64 4000 → 54274 [FIN, ACK] Seq=1283 Ack=3198 Win=2617600 Len=0
1379 445.784832 ::1 ::1 TCP 64 54274 → 4000 [ACK] Seq=3198 Ack=1284 Win=2617600 Len=0
1380 445.785114 ::1 ::1 TCP 64 54274 → 4000 [FIN, ACK] Seq=3198 Ack=1284 Win=2617600 Len=0
1381 445.785160 ::1 ::1 TCP 64 4000 → 54274 [ACK] Seq=1284 Ack=3199 Win=2617600 Len=0
1394 462.116495 ::1 ::1 TCP 67 4000 → 53662 [PSH, ACK] Seq=1169 Ack=1134 Win=10218 Len=3
1395 462.116541 ::1 ::1 TCP 64 53662 → 4000 [ACK] Seq=1134 Ack=1172 Win=10222 Len=0
1396 462.117054 ::1 ::1 TCP 71 53662 → 4000 [PSH, ACK] Seq=1134 Ack=1172 Win=10222 Len=7
1397 462.117094 ::1 ::1 TCP 64 4000 → 53662 [ACK] Seq=1172 Ack=1141 Win=10218 Len=0


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
frame contains "{"
No. Time Source Destination Protocol Lengtl Info
764 149.668516 ::1 ::1 HTTP/J… 103 POST /api/auth/login HTTP/1.1 , JSON (application/json)
769 149.672447 ::1 ::1 HTTP/J… 2207 POST /api/auth/login HTTP/1.1 , JSON (application/json)
773 149.675602 127.0.0.1 127.0.0.1 MONGO 657 Response : Extensible Message Format
775 150.214968 ::1 ::1 HTTP/J… 669 HTTP/1.1 200 OK , JSON (application/json)
777 150.216794 ::1 ::1 HTTP/J… 669 HTTP/1.1 200 OK , JSON (application/json)
785 150.220856 127.0.0.1 127.0.0.1 MONGO 283 Request : Extensible Message Format
800 150.232906 ::1 ::1 HTTP 438 HTTP/1.1 200 OK (text/plain)
802 150.252423 ::1 ::1 HTTP 793 POST /socket.io/?EIO=4&transport=polling&t=wr62navd&sid=zr12GHaDI89INbT3AAAU HTTP/1.1 (text/plain)
810 150.258984 ::1 ::1 HTTP 351 HTTP/1.1 200 OK (text/plain)
856 156.748289 127.0.0.1 127.0.0.1 MONGO 657 Response : Extensible Message Format
879 157.660139 ::1 ::1 WebSoc… 643 WebSocket Text [FIN] [MASKED]
881 157.662462 ::1 ::1 TCP 641 4000 → 53662 [PSH, ACK] Seq=19 Ack=43 Win=10223 Len=577
887 157.671015 ::1 ::1 TCP 697 53662 → 4000 [PSH, ACK] Seq=43 Ack=596 Win=10224 Len=633
889 157.672867 ::1 ::1 WebSoc… 695 WebSocket Text [FIN]
897 157.680566 ::1 ::1 TCP 196 4000 → 53662 [PSH, ACK] Seq=596 Ack=676 Win=10220 Len=132
920 166.506274 ::1 ::1 TCP 469 4000 → 53662 [PSH, ACK] Seq=731 Ack=683 Win=10220 Len=405
931 166.509173 ::1 ::1 HTTP/J… 469 POST /api/messages HTTP/1.1 , JSON (application/json)
936 166.513385 ::1 ::1 HTTP/J… 2813 POST /api/messages HTTP/1.1 , JSON (application/json)
942 166.526081 ::1 ::1 HTTP/J… 443 HTTP/1.1 200 OK , JSON (application/json)
944 166.526941 ::1 ::1 HTTP/J… 443 HTTP/1.1 200 OK , JSON (application/json)
977 176.815160 ::1 ::1 TCP 438 53662 → 4000 [PSH, ACK] Seq=683 Ack=1136 Win=10222 Len=374
979 176.817199 ::1 ::1 WebSoc… 477 WebSocket Text [FIN]
990 176.820368 ::1 ::1 HTTP/J… 477 POST /api/messages HTTP/1.1 , JSON (application/json)
995 176.824419 ::1 ::1 HTTP/J… 2821 POST /api/messages HTTP/1.1 , JSON (application/json)
1001 176.835556 ::1 ::1 HTTP/J… 443 HTTP/1.1 200 OK , JSON (application/json)
1003 176.836471 ::1 ::1 HTTP/J… 443 HTTP/1.1 200 OK , JSON (application/json)
1021 177.945404 ::1 ::1 TCP 85 53644 → 5173 [PSH, ACK] Seq=85 Ack=1 Win=10229 Len=21
1165 282.360041 127.0.0.1 127.0.0.1 TCP 44 27017 → 52557 [ACK] Seq=8765 Ack=1509 Win=10095 Len=0
1169 282.438916 127.0.0.1 127.0.0.1 TCP 56 [TCP Keep-Alive ACK] 52553 → 27017 [ACK] Seq=2 Ack=1 Win=10077 Len=0 SLE=0 SRE=1
1224 322.395900 127.0.0.1 127.0.0.1 MONGO 357 Response : Extensible Message Format
1296 386.460051 ::1 ::1 TCP 85 53773 → 5173 [PSH, ACK] Seq=149 Ack=1 Win=10229 Len=21
1446 502.554227 127.0.0.1 127.0.0.1 MONGO 357 Response : Extensible Message Format



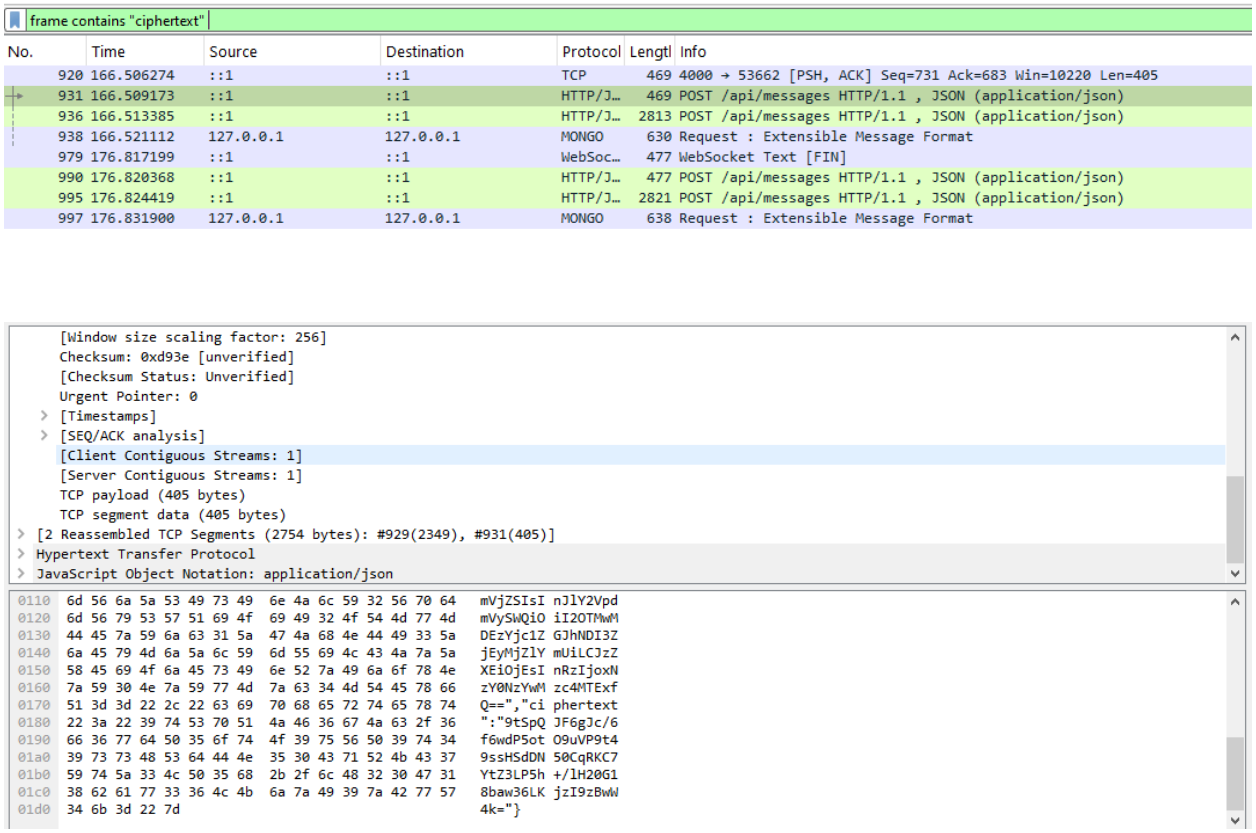
ip.addr == 127.0.0.1

No.	Time	Source	Destination	Protocol	Length	Info
1466	522.571952	127.0.0.1	127.0.0.1	MONGO	96	Request : Extensible Message Format
1467	522.572023	127.0.0.1	127.0.0.1	TCP	44	27017 → 52557 [ACK] Seq=16277 Ack=2757 Win=10090 Len=0
1468	522.572476	127.0.0.1	127.0.0.1	MONGO	357	Response : Extensible Message Format
1469	522.572523	127.0.0.1	127.0.0.1	TCP	44	52557 → 27017 [ACK] Seq=2757 Ack=16590 Win=10140 Len=0
1470	522.695905	127.0.0.1	127.0.0.1	TCP	45	[TCP Keep-Alive] 52553 → 27017 [ACK] Seq=1 Ack=1 Win=10077 Len=1
1471	522.696035	127.0.0.1	127.0.0.1	TCP	56	[TCP Keep-Alive ACK] 27017 → 52553 [ACK] Seq=1 Ack=2 Win=10232 Len=0 SLE=1 SRE=2
1473	524.512255	127.0.0.1	127.0.0.1	TCP	45	[TCP Keep-Alive] 54291 → 4266 [ACK] Seq=328 Ack=276 Win=2619392 Len=1
1474	524.512300	127.0.0.1	127.0.0.1	TCP	56	[TCP Keep-Alive ACK] 4266 → 54291 [ACK] Seq=276 Ack=329 Win=2619648 Len=0 SLE=328 SRE=329
1475	526.230759	127.0.0.1	127.0.0.1	MONGO	357	Response : Extensible Message Format
1476	526.230890	127.0.0.1	127.0.0.1	TCP	44	52545 → 27017 [ACK] Seq=1 Ack=16590 Win=10138 Len=0
1478	530.278888	127.0.0.1	127.0.0.1	TCP	44	54291 → 4266 [FIN, ACK] Seq=329 Ack=276 Win=2619392 Len=0
1479	530.278986	127.0.0.1	127.0.0.1	TCP	44	4266 → 54291 [ACK] Seq=276 Ack=330 Win=2619648 Len=0
1480	530.279570	127.0.0.1	127.0.0.1	TCP	44	4266 → 54291 [FIN, ACK] Seq=276 Ack=330 Win=2619648 Len=0
1481	530.279663	127.0.0.1	127.0.0.1	TCP	44	54291 → 4266 [ACK] Seq=330 Ack=277 Win=2619392 Len=0
1482	531.292393	127.0.0.1	127.0.0.1	TCP	56	54371 → 4266 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1483	531.292494	127.0.0.1	127.0.0.1	TCP	56	4266 → 54371 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1484	531.292569	127.0.0.1	127.0.0.1	TCP	44	54371 → 4266 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
1485	531.293398	127.0.0.1	127.0.0.1	HTTP	372	GET /stream HTTP/1.1
1486	531.293436	127.0.0.1	127.0.0.1	TCP	44	4266 → 54371 [ACK] Seq=1 Ack=329 Win=2619648 Len=0
1487	531.293969	127.0.0.1	127.0.0.1	TCP	319	4266 → 54371 [PSH, ACK] Seq=1 Ack=329 Win=2619648 Len=275
1488	531.294004	127.0.0.1	127.0.0.1	TCP	44	54371 → 4266 [ACK] Seq=329 Ack=276 Win=2619392 Len=0
1489	532.573017	127.0.0.1	127.0.0.1	MONGO	96	Request : Extensible Message Format
1490	532.573069	127.0.0.1	127.0.0.1	TCP	44	27017 → 52557 [ACK] Seq=16590 Ack=2809 Win=10090 Len=0
1491	532.573367	127.0.0.1	127.0.0.1	MONGO	357	Response : Extensible Message Format
1492	532.573412	127.0.0.1	127.0.0.1	TCP	44	52557 → 27017 [ACK] Seq=2809 Ack=16903 Win=10139 Len=0
1494	535.236239	127.0.0.1	127.0.0.1	TCP	45	[TCP Keep-Alive] 52549 → 27017 [ACK] Seq=0 Ack=2 Win=10231 Len=1
1495	535.236345	127.0.0.1	127.0.0.1	TCP	56	[TCP Keep-Alive ACK] 27017 → 52549 [ACK] Seq=2 Ack=1 Win=10232 Len=0 SLE=0 SRE=1
1496	536.232209	127.0.0.1	127.0.0.1	MONGO	357	Response : Extensible Message Format
1497	536.232324	127.0.0.1	127.0.0.1	TCP	44	52545 → 27017 [ACK] Seq=1 Ack=16903 Win=10137 Len=0
1499	536.859161	127.0.0.1	127.0.0.1	TCP	45	[TCP Keep-Alive] 52551 → 27017 [ACK] Seq=2978 Ack=1019 Win=10209 Len=1
1500	536.859197	127.0.0.1	127.0.0.1	TCP	56	[TCP Keep-Alive ACK] 27017 → 52551 [ACK] Seq=1019 Ack=2979 Win=10165 Len=0 SLE=2978 SRE=2979
1507	542.581256	127.0.0.1	127.0.0.1	MONGO	96	Request : Extensible Message Format



```
> [SEQ/ACK analysis]
  [Client Contiguous Streams: 1]
  [Server Contiguous Streams: 1]
  TCP payload (405 bytes)
  TCP segment data (405 bytes)
> [2 Reassembled TCP Segments (2754 bytes): #929(2349), #931(405)]
```

```
0040  7b 22 73 65 73 73 69 6f  6e 49 64 22 3a 22 36 39   {"sessio nId":"69
0050  33 30 30 31 36 65 37 35  64 62 61 34 32 37 66 31   30016e75 dba427f1
0060  32 32 36 65 63 65 3a 36  39 33 30 30 31 33 62 37   226ece:6 930013b7
0070  35 64 62 61 34 32 37 66  31 32 32 36 65 62 65 22   5dba427f 1226ebe"
0080  2c 22 72 65 63 65 69 76  65 72 49 64 22 3a 22 36   ,"receiv erId":"6
0090  39 33 30 30 31 33 62 37  35 64 62 61 34 32 37 66   930013b7 5dba427f
00a0  31 32 32 36 65 62 65 22  2c 22 73 65 71 22 3a 31   1226ebe" ,"seq":1
00b0  2c 22 74 73 22 3a 31 37  36 34 37 36 30 33 37 38   ,"ts":17 64760378
00c0  31 31 31 2c 22 69 76 22  3a 22 4c 77 57 70 6c 47   111,"iv" :"LwWplG
00d0  7a 67 75 2b 73 4c 62 39  65 37 22 2c 22 61 61 64   zgu+sLb9 e7","aad
00e0  22 3a 22 65 79 4a 7a 5a  57 35 6b 5a 58 4a 4a 5a   ":"eyJzZ W5kZXJJZ
00f0  43 49 36 49 6a 59 35 4d  7a 41 77 4d 54 5a 6c 4e   CI6IjY5M zAwMTZlN
0100  7a 56 6b 59 6d 45 30 4d  6a 64 6d 4d 54 49 79 4e   zVkYmE0M jdmMTIyN
0110  6d 56 6a 5a 53 49 73 49  6e 4a 6c 59 32 56 70 64   mVjZSIsI nJlY2Vpd
0120  6d 56 79 53 57 51 69 4f  69 49 32 4f 54 4d 77 4d   mVySWQiO iI2OTMwM
0130  44 45 7a 59 6a 63 31 5a  47 4a 68 4e 44 49 33 5a   DEzYjc1Z GJhNDI3Z
0140  6a 45 79 4d 6a 5a 6c 59  6d 55 69 4c 43 4a 7a 5a   jEyMjZlY mUiLCJzZ
0150  58 45 69 4f 6a 45 73 49  6e 52 7a 49 6a 6f 78 4e   XEiOjEsI nRzIjoxN
0160  7a 59 30 4e 7a 59 77 4d  7a 63 34 4d 54 45 78 66   zY0NzYwM zc4MTExf
0170  51 3d 3d 22 2c 22 63 69  70 68 65 72 74 65 78 74   Q==","ci phertext
0180  22 3a 22 39 74 53 70 51  4a 46 36 67 4a 63 2f 36   ":"9tSpQ JF6gJc/6
0190  66 36 77 64 50 35 6f 74  4f 39 75 56 50 39 74 34   f6wdP5ot O9uVP9t4
```



frame contains "ciphertext"

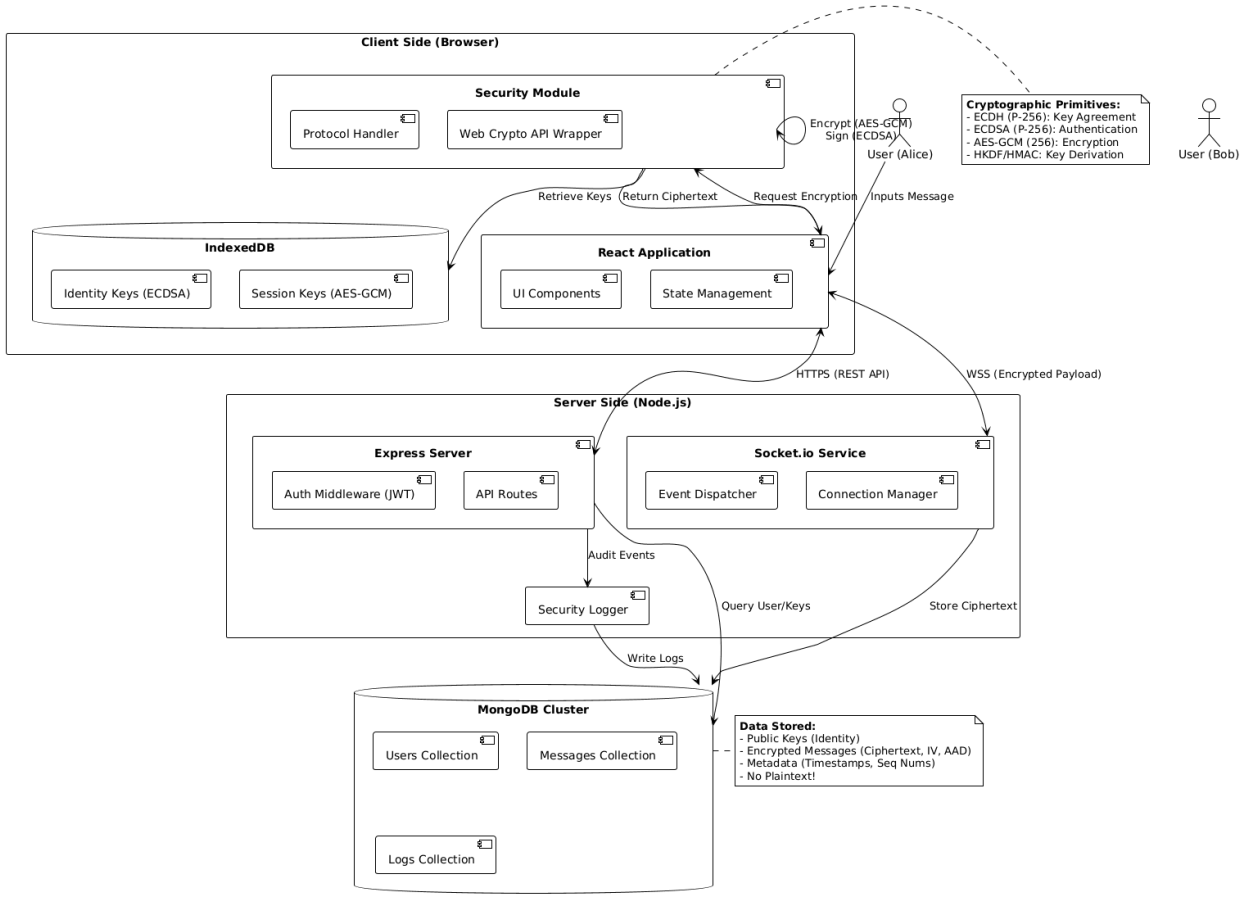
No.	Time	Source	Destination	Protocol	Lengtl	Info
920	166.506274	::1	::1	TCP	469	4000 → 53662 [PSH, ACK] Seq=731 Ack=683 Win=10220 Len=405
931	166.509173	::1	::1	HTTP/J…	469	POST /api/messages HTTP/1.1 , JSON (application/json)
936	166.513385	::1	::1	HTTP/J…	2813	POST /api/messages HTTP/1.1 , JSON (application/json)
938	166.521112	127.0.0.1	127.0.0.1	MONGO	630	Request : Extensible Message Format
979	176.817199	::1	::1	WebSoc…	477	WebSocket Text [FIN]
990	176.820368	::1	::1	HTTP/J…	477	POST /api/messages HTTP/1.1 , JSON (application/json)
995	176.824419	::1	::1	HTTP/J…	2821	POST /api/messages HTTP/1.1 , JSON (application/json)
997	176.831900	127.0.0.1	127.0.0.1	MONGO	638	Request : Extensible Message Format


```
    [Window size scaling factor: 256]
    Checksum: 0xd93e [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]
    [Client Contiguous Streams: 1]
    [Server Contiguous Streams: 1]
    TCP payload (405 bytes)
    TCP segment data (405 bytes)
> [2 Reassembled TCP Segments (2754 bytes): #929(2349), #931(405)]
> Hypertext Transfer Protocol
> JavaScript Object Notation: application/json
```

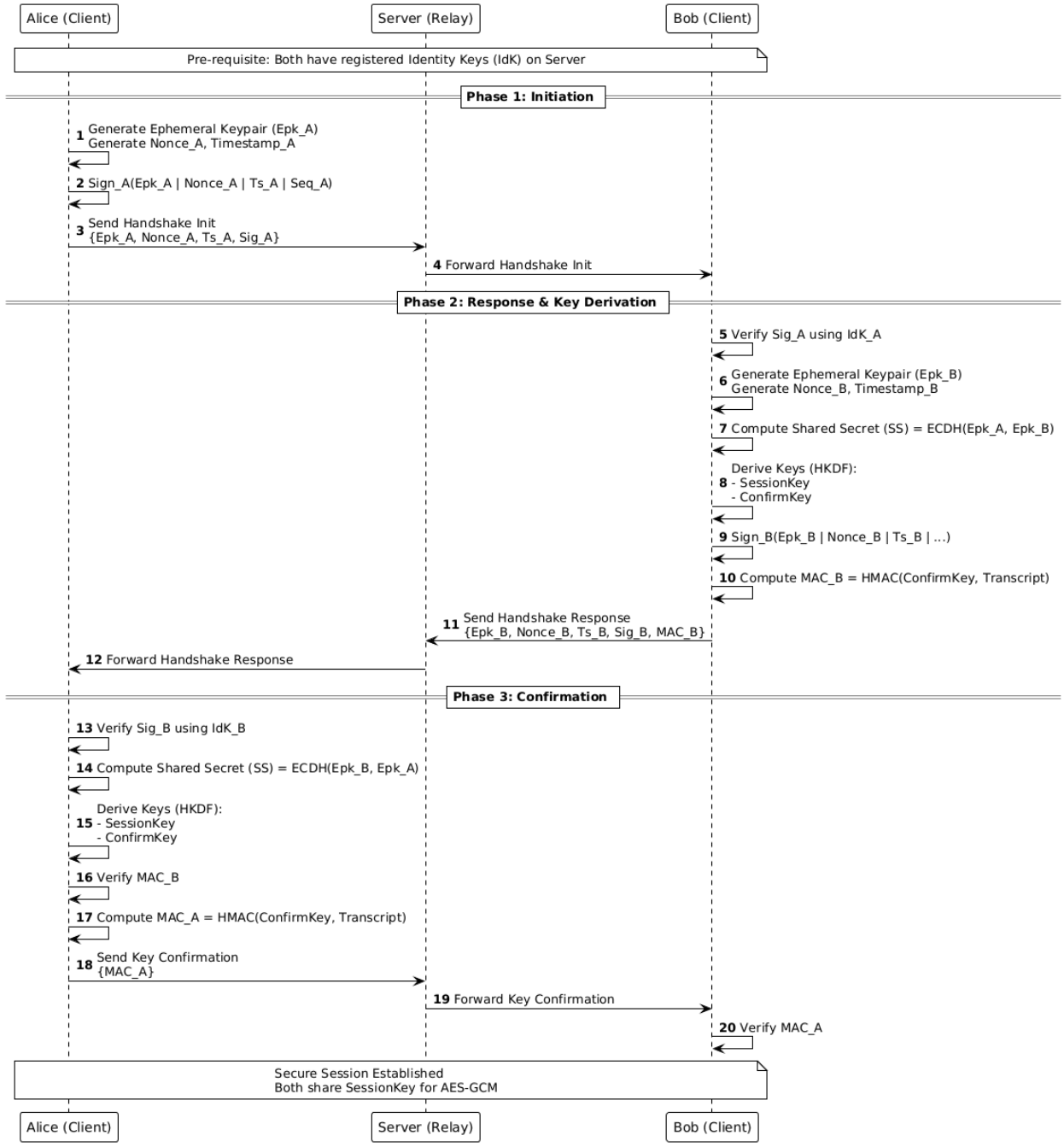


```
0110  6d 56 6a 5a 53 49 73 49  6e 4a 6c 59 32 56 70 64   mVjZSIsI nJlY2Vpd
0120  6d 56 79 53 57 51 69 4f  69 49 32 4f 54 4d 77 4d   mVySWQiO iI2OTMwM
0130  44 45 7a 59 6a 63 31 5a  47 4a 68 4e 44 49 33 5a   DEzYjc1Z GJhNDI3Z
0140  6a 45 79 4d 6a 5a 6c 59  6d 55 69 4c 43 4a 7a 5a   jEyMjZlY mUiLCJzZ
0150  58 45 69 4f 6a 45 73 49  6e 52 7a 49 6a 6f 78 4e   XEiOjEsI nRzIjoxN
0160  7a 59 30 4e 7a 59 77 4d  7a 63 34 4d 54 45 78 66   zY0NzYwM zc4MTExf
0170  51 3d 3d 22 2c 22 63 69  70 68 65 72 74 65 78 74   Q==","ci phertext
0180  22 3a 22 39 74 53 70 51  4a 46 36 67 4a 63 2f 36   ":"9tSpQ JF6gJc/6
0190  66 36 77 64 50 35 6f 74  4f 39 75 56 50 39 74 34   f6wdP5ot O9uVP9t4
01a0  39 73 73 48 53 64 44 4e  35 30 43 71 52 4b 43 37   9ssHSdDN 50CqRKC7
01b0  59 74 5a 33 4c 50 35 68  2b 2f 6c 48 32 30 47 31   YtZ3LP5h +/lH20G1
01c0  38 62 61 77 33 36 4c 4b  6a 7a 49 39 7a 42 77 57   8baw36LK jzI9zBwW
01d0  34 6b 3d 22 7d                                     4k="}
```

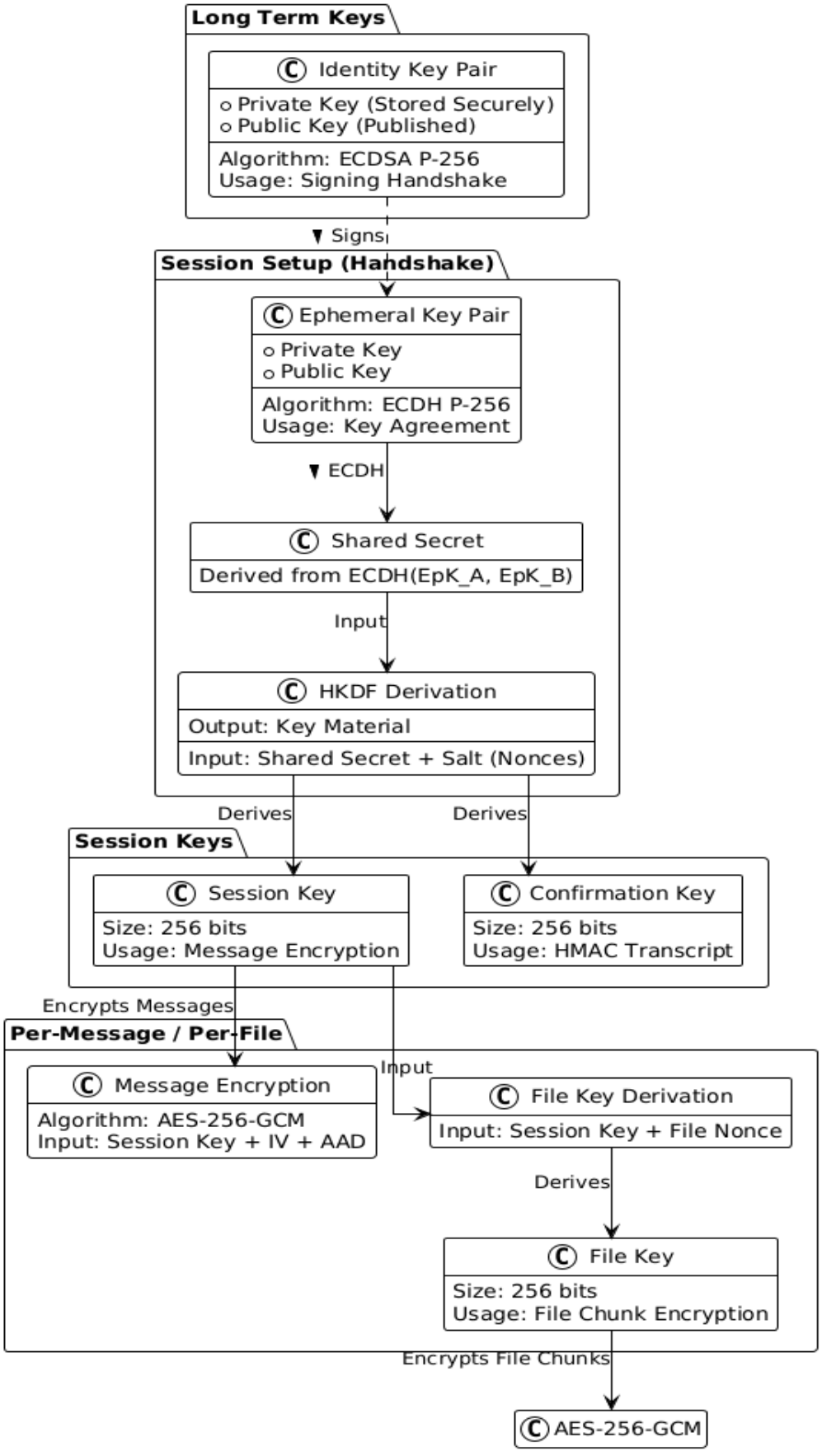
Architecture Diagram



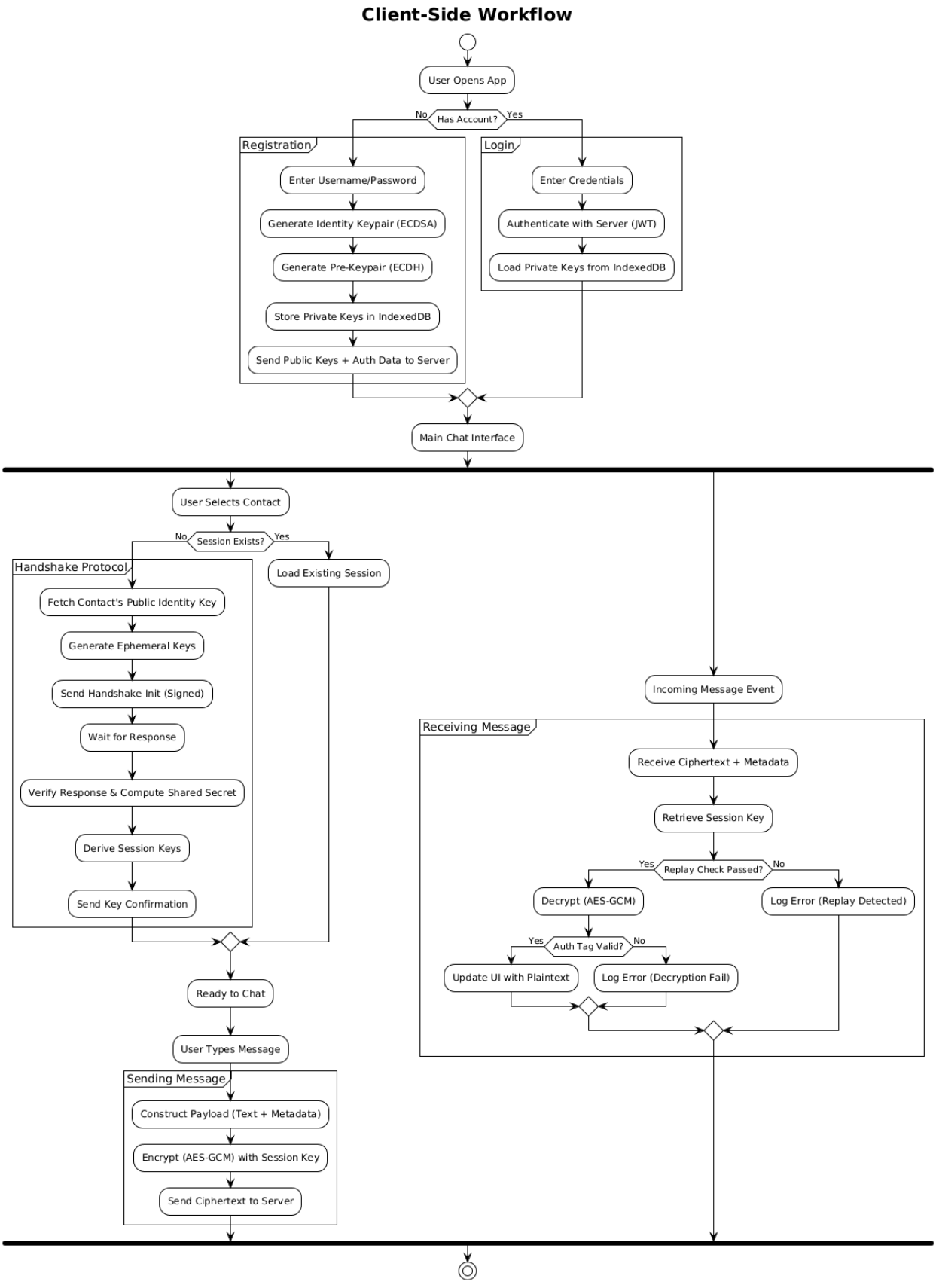
Key exchange protocol diagram



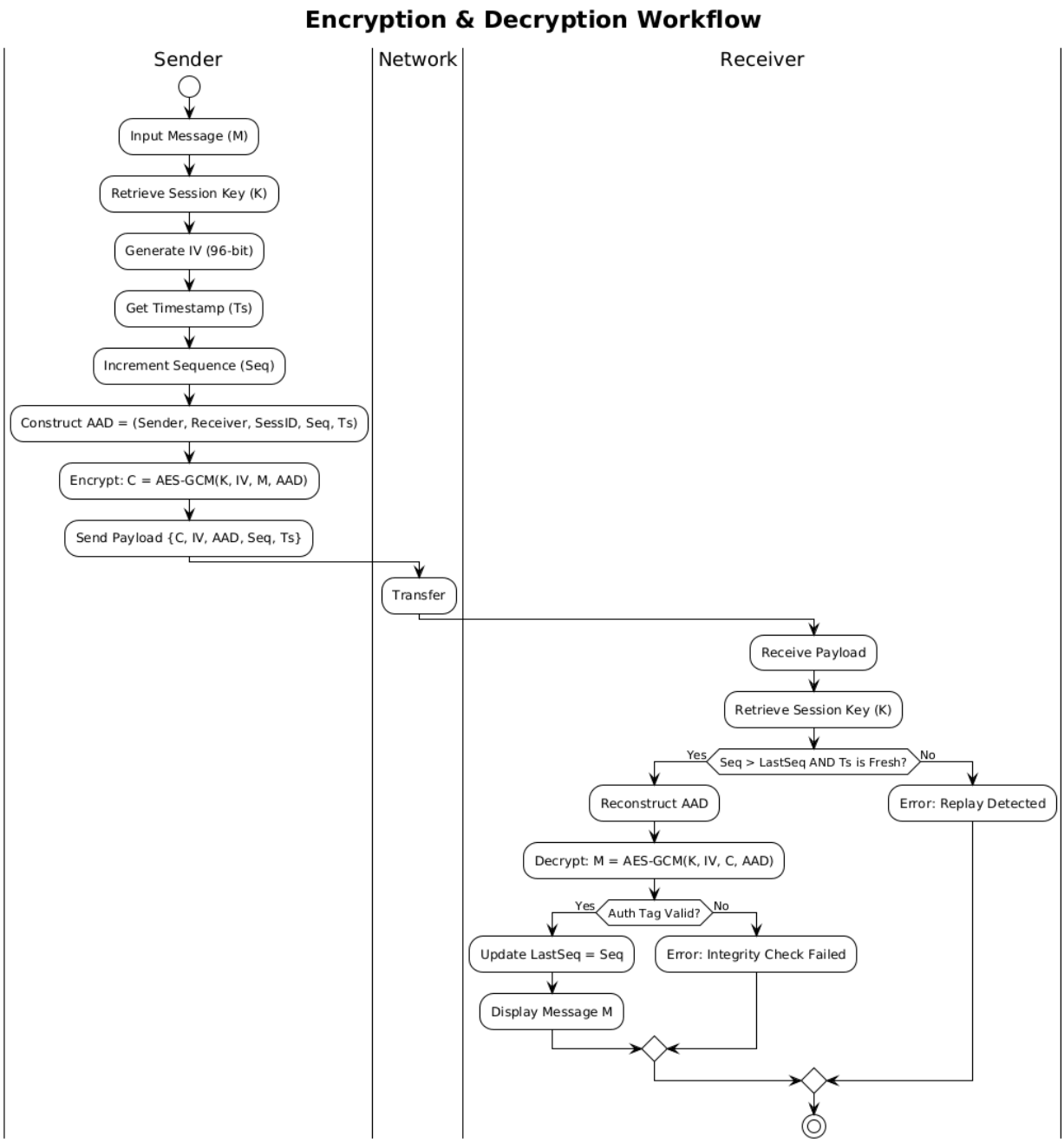
Cryptographic design


Long Term Keys
Identity Key Pair
Private Key (Stored Securely)
Public Key (Published)
Algorithm: ECDSA P-256
Usage: Signing Handshake
Signs
Session Setup (Handshake)
Ephemeral Key Pair
Private Key
Public Key
Algorithm: ECDH P-256
Usage: Key Agreement
ECDH
Shared Secret
Derived from ECDH(EpK_A, EpK_B)
Input
HKDF Derivation
Output: Key Material
Input: Shared Secret + Salt (Nonces)
Derives
Derives
Session Keys
Session Key
Size: 256 bits
Usage: Message Encryption
Confirmation Key
Size: 256 bits
Usage: HMAC Transcript
Encrypts Messages
Per-Message / Per-File
Input
Message Encryption
Algorithm: AES-256-GCM
Input: Session Key + IV + AAD
File Key Derivation
Input: Session Key + File Nonce
Derives
File Key
Size: 256 bits
Usage: File Chunk Encryption
Encrypts File Chunks
AES-256-GCM

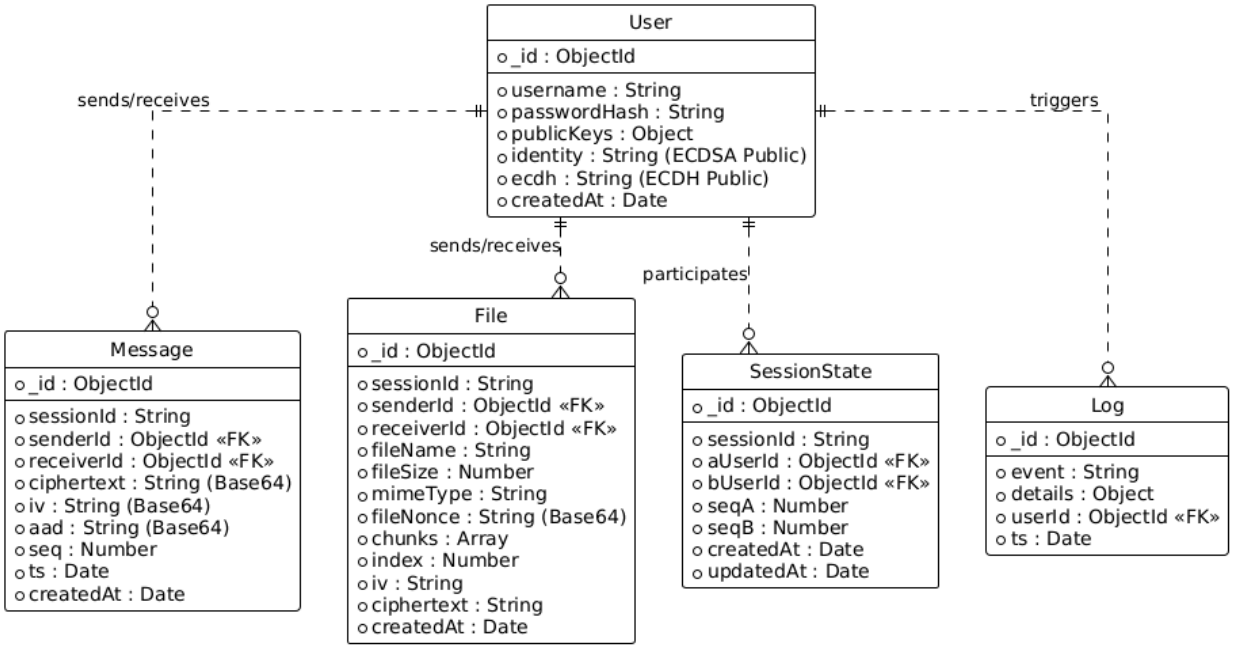
Client-side flow diagram


Client-Side Workflow
User Opens App
Has Account?
No
Yes
Registration
Enter Username/Password
Generate Identity Keypair (ECDSA)
Generate Pre-Keypair (ECDH)
Store Private Keys in IndexedDB
Send Public Keys + Auth Data to Server
Login
Enter Credentials
Authenticate with Server (JWT)
Load Private Keys from IndexedDB
Main Chat Interface
User Selects Contact
Session Exists?
No
Yes
Handshake Protocol
Fetch Contact's Public Identity Key
Generate Ephemeral Keys
Send Handshake Init (Signed)
Wait for Response
Verify Response & Compute Shared Secret
Derive Session Keys
Send Key Confirmation
Load Existing Session
Ready to Chat
User Types Message
Sending Message
Construct Payload (Text + Metadata)
Encrypt (AES-GCM) with Session Key
Send Ciphertext to Server
Incoming Message Event
Receiving Message
Receive Ciphertext + Metadata
Retrieve Session Key
Replay Check Passed?
Yes
No
Decrypt (AES-GCM)
Log Error (Replay Detected)
Auth Tag Valid?
Yes
No
Update UI with Plaintext
Log Error (Decryption Fail)

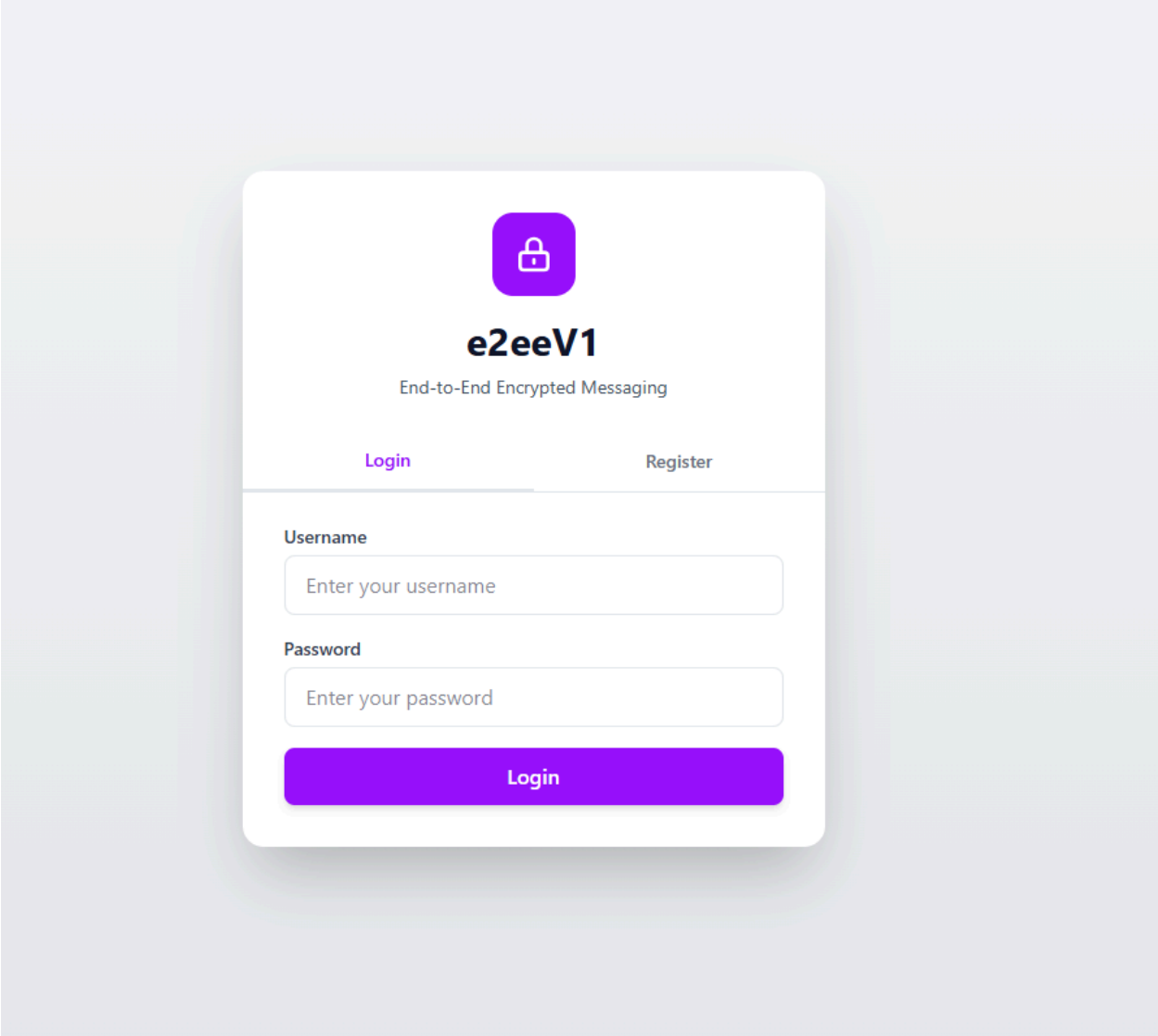
Encryption/decryption workflows

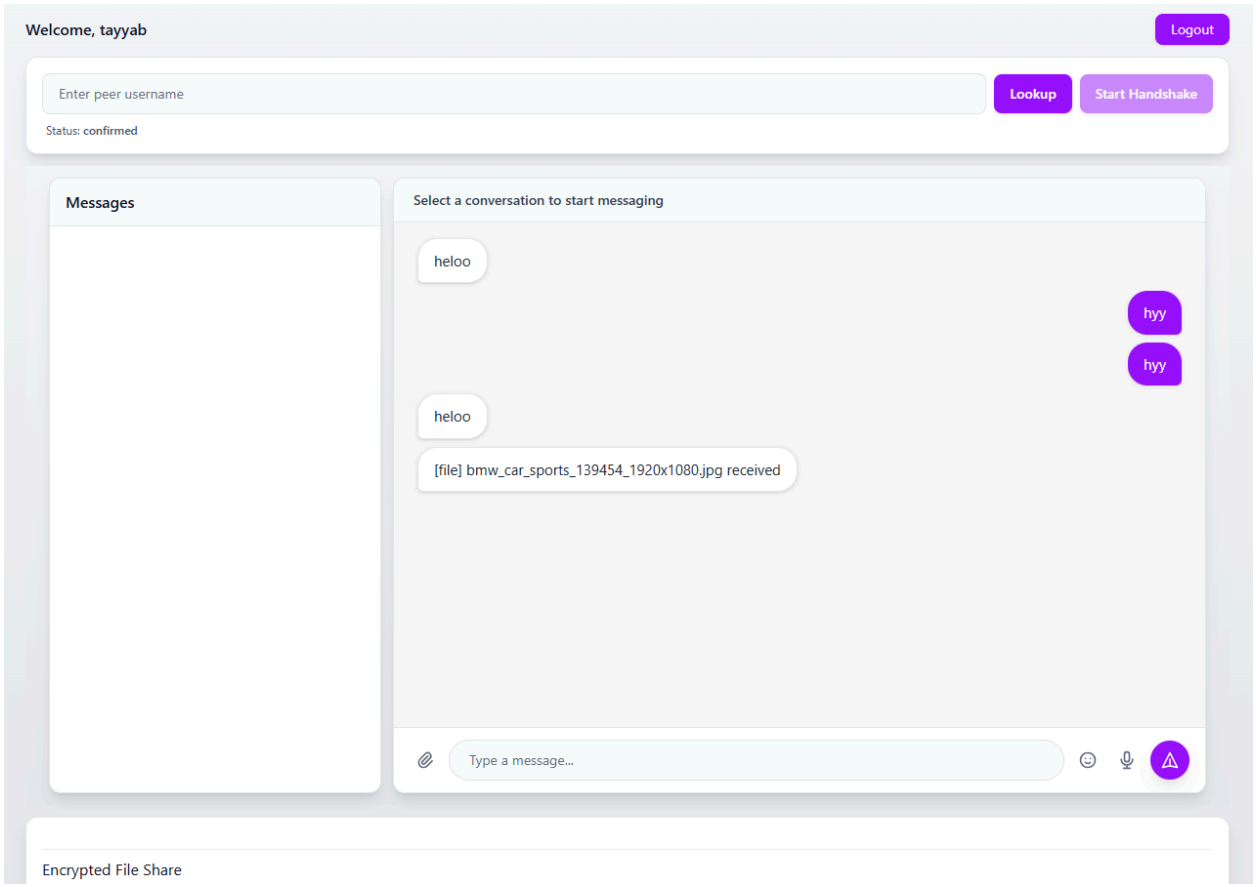


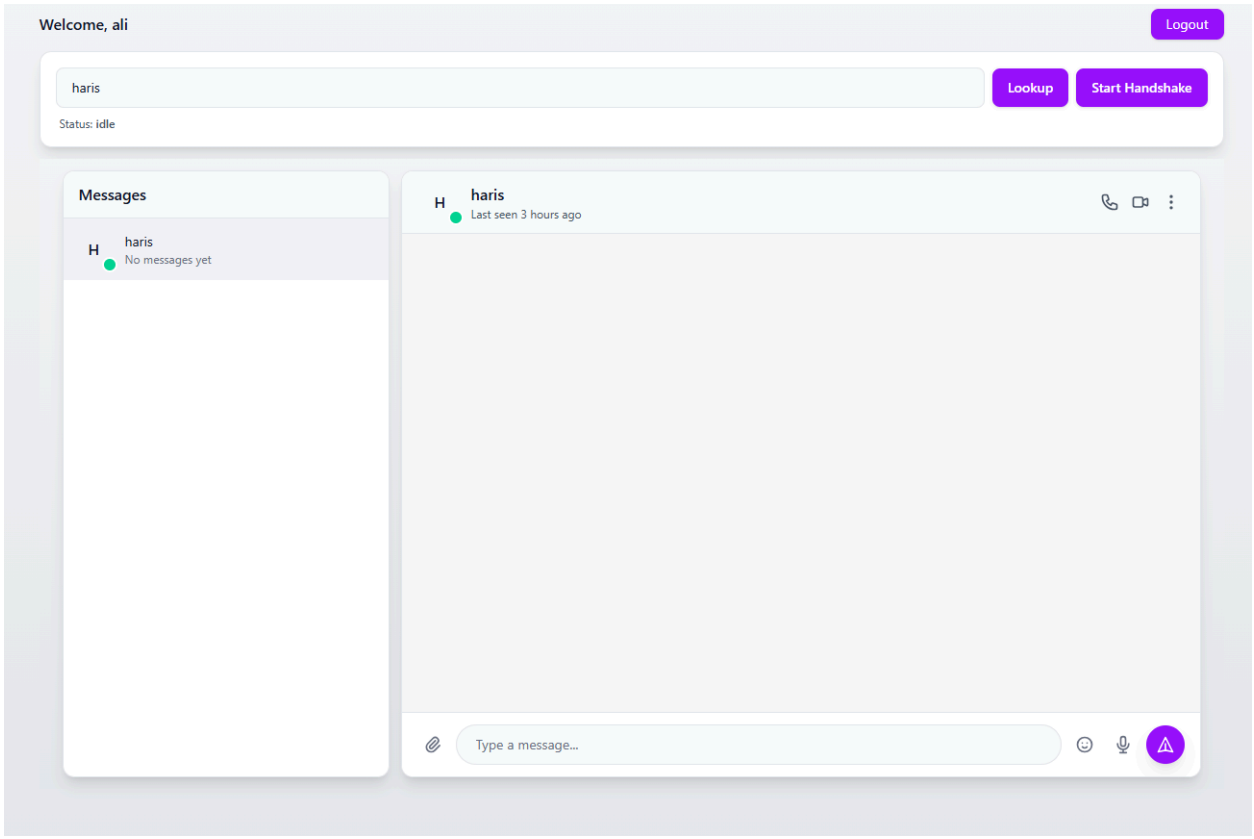
Schema design



User interface




Welcome, tayyab
Logout
Enter peer username
Lookup
Start Handshake
Status: confirmed
Messages
Select a conversation to start messaging
heloo
hyy
hyy
heloo
[file] bmw_car_sports_139454_1920x1080.jpg received
Type a message...
Encrypted File Share


Welcome, ali
Logout
haris
Lookup
Start Handshake
Status: idle
Messages
H
haris
No messages yet
H
haris
Last seen 3 hours ago
Type a message...