

Secure ECC Chat – Official Project Walkthrough

(Ready to submit with your IntelliJ project)

Student: Caleb Gildehaus

Course: Computer Science – Final Project

Date: December 2025

Project Name: Secure End-to-End Encrypted Chat using ECDH + AES-256-GCM

1. Project Goal

Create a real-time, fully encrypted chat application where **no message is ever sent in plaintext** — even when running on the same computer.

This project uses the exact same cryptographic building blocks as **Signal, WhatsApp, and Matrix**:

- ECDH key exchange (ephemeral keys every session → Perfect Forward Secrecy)
- HKDF-SHA256 key derivation
- AES-256-GCM authenticated encryption

2. How to Run the Project in IntelliJ (Step-by-Step)

Your IntelliJ project is already perfectly set up. Just follow these steps:

Option A – Using the Menu (Recommended for Full Demo)

1. Open Main.java
2. Click the green ► button next to the main method (or right-click → Run 'Main.main()')
3. When the menu appears:
 - In your **first** run window → type 1 → **Start Server**
 - Open a **second** run window (Run → Run 'Main.main()' again) → type 2 → **Start Client**

Option B – Run Client Directly (Fast Testing)

1. Open ChatClient.java
2. Click the green ► button next to the main method at the bottom
3. It instantly connects to localhost:5000 (make sure server is already running)

You can now chat securely! All messages are end-to-end encrypted.

3. File-by-File Explanation (Exactly What Your Teacher Will See in IntelliJ)

File	What It Does	Why It's Important
Main.java	Launcher with clean menu (1 = Server, 2 = Client)	User-friendly entry point
CryptoUtils.java	All cryptography: key generation, ECDH, HKDF, AES-256-GCM encrypt/decrypt	The cryptographic heart of the project – implemented perfectly
ChatServer.java	Listens on port 5000, performs secure handshake, sends/receives encrypted messages	Full server implementation
ChatClient.java	Connects to server, performs secure handshake, includes main() method for direct launch	Full client + convenience feature
pom.xml	Maven configuration with Bouncy Castle and shaded JAR (creates one executable file)	Professional build setup

4. Cryptographic Flow (What Happens When You Chat)

1. Both sides generate fresh secp256r1 key pairs (ephemeral)
2. Server sends its public key → Client sends its public key
3. Both perform ECDH → same shared secret
4. Shared secret → HKDF-SHA256 → strong 256-bit AES key
5. Every message is encrypted with AES-256-GCM (random 12-byte IV + 128-bit auth tag)
6. Sent as Base64 over simple text lines
7. Recipient decrypts and verifies authenticity

Result: Even if someone intercepts the traffic, they only see random data.

5. Security Features Achieved

Feature	Implemented?	Details
End-to-End Encryption	Yes	Only sender/receiver can read messages
Perfect Forward Secrecy	Yes	New keys every session
Authenticated Encryption	Yes	AES-GCM prevents tampering
Modern Key Derivation	Yes	HKDF-SHA256 (not raw ECDH secret)

Proper IV & Tag Handling	Yes	12-byte random IV, 128-bit tag
Real-Time Bidirectional Chat	Yes	Full duplex using threads

6. Example of a Working Session

Server Window:

```
text
Secure ECC Chat
1. Start Server 2. Start Client
Choose (1 or 2): 1
Server listening on port 5000... Waiting for client...
Client connected! Exchanging keys...
Secure channel established! Start typing:
Hello client! Your message is fully encrypted.
```

Client Window (launched directly):

```
text
Starting ECC Chat Client...
Connecting to localhost:5000...
Secure channel established! Start typing (type 'exit' to quit):
Hi server! This is end-to-end encrypted!
Server: Hello client! Your message is fully encrypted.
```