

**CSE7101- Capstone Project  
Review-3**

---

**Secure AI Chat System (End-to-End Encryption + AI  
Moderation)**

**Batch Number:** COM\_58

**Roll Number :**  
20221COM0136

**Name:**  
R. Charu Swathi Sree

**Under the Supervision of,**

**Dr.Ruhin Kouser R**  
**Assistant Professor – Senior Scale**  
**School of Computer Science and Engineering**  
**Presidency University**

**Name of the Program: B.tech. Computer Engineering (AI&ML)**

**Name of the HoD: Dr.Pallavi R**

**Name of the Program Project Coordinator: Dr.Debasmita Mishra**

**Name of the School Project Coordinators: Dr. Sampath A K , Dr. Geetha A**

# Content

---

- Abstract
- Literature Review
- System Design
- Architecture Diagram
- Algorithm Details
- Modules Description
- Source Code Details (50%)
- 50% Implementation with Live Demo
- 50% Report

# Abstract

---

Modern messaging platforms face two major challenges: ensuring user privacy and controlling toxic or abusive content. End-to-end encryption protects privacy but limits server-side moderation. This project presents a secure AI-enabled chat system that integrates end-to-end encryption with client-side AI-based content moderation. Messages are analyzed for toxicity before encryption, ensuring safety without compromising privacy. The system uses AES-256 for message encryption, RSA for secure key exchange, and NLP models for real-time moderation. The proposed approach demonstrates that secure communication and intelligent moderation can coexist in a single messaging platform.

# Literature Review

---

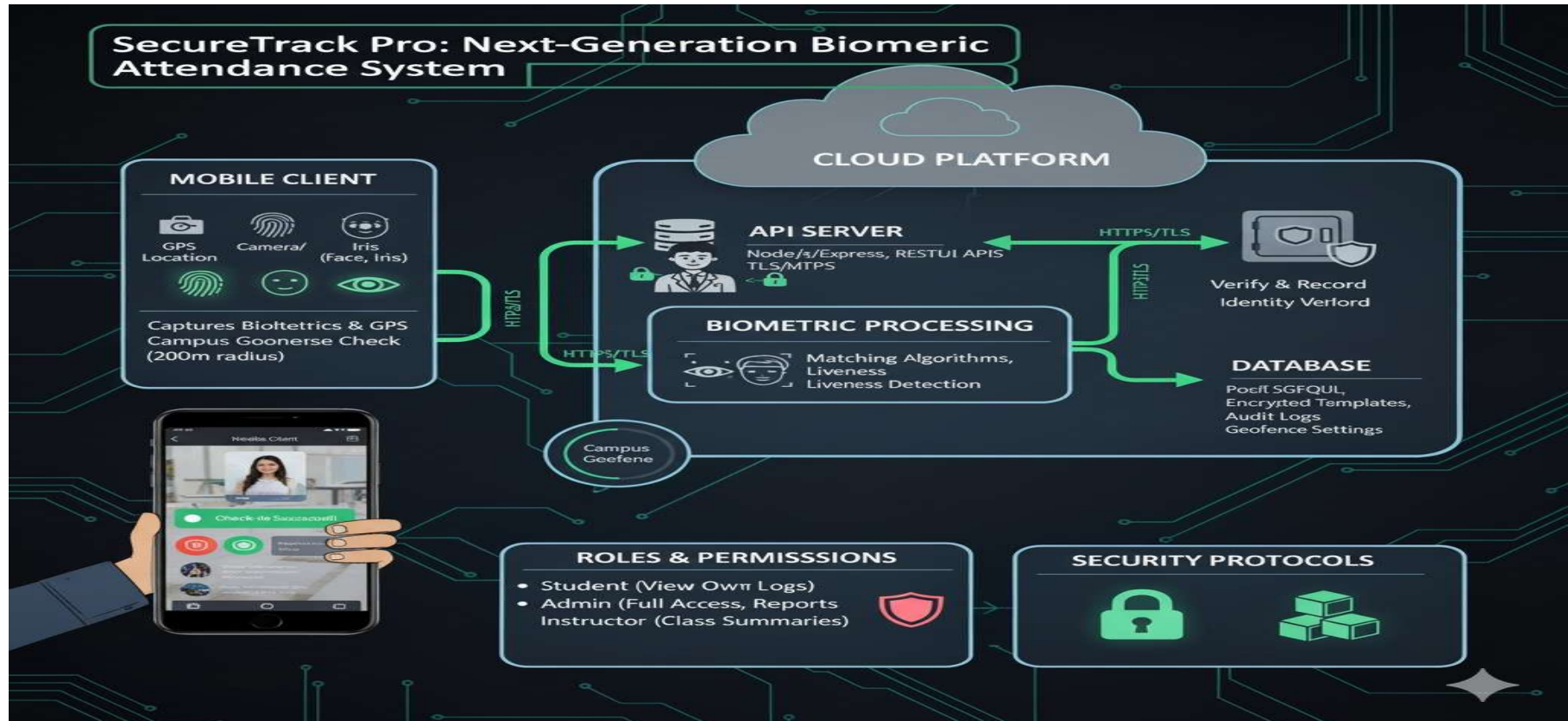
- End-to-end encryption is widely adopted in platforms like Signal and WhatsApp to protect message confidentiality.
- AI and NLP techniques are commonly used for detecting toxic and abusive content in online platforms.
- Existing systems often prioritize either privacy or moderation, but not both simultaneously.
- This project addresses the gap by combining client-side AI moderation with strong encryption.

# System Design

---

- The system follows a client-server architecture.  
Key design principles include:
  - Client-side AI moderation
  - End-to-end encryption
  - Secure communication channels
  - Server as a message router only
- This design ensures confidentiality, integrity, and user safety

# Architecture Diagram



# Algorithm Details

---

## **Algorithm: Secure Message Processing**

- Accept message input
- Perform AI-based toxicity check
- If toxic → block or warn user
- Generate AES session key
- Encrypt message using AES-GCM
- Encrypt AES key using RSA
- Transmit encrypted payload

## **Algorithm: Secure Message Processing**

- Accept message input
- Perform AI-based toxicity check
- If toxic → block or warn user
- Generate AES session key
- Encrypt message using AES-GCM
- Encrypt AES key using RSA
- Transmit encrypted payload

# Module Description

---

- User Authentication Module:** Handles login and sessions
- AI Moderation Module:** Detects toxic content
- Encryption Module:** Secures messages using AES and RSA
- Messaging Module:** Handles chat communication
- Backend Module:** Routes encrypted message

# Source Code: Main.py

---

```
from fastapi import FastAPI, WebSocket, WebSocketDisconnect
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
import json
```

```
app = FastAPI()
```

```
# ----- CORS -----
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)
```

```
# ----- MODERATION -----
BAD_WORDS = {
    "kill", "hate", "abuse", "stupid", "dumb", "idiot", "moron"
}
```

```
class Message(BaseModel):
    text: str
```

```
@app.post("/moderate")
def moderate(msg: Message):
    text = msg.text.lower()
    for word in BAD_WORDS:
        if word in text:
            return {"status": "BLOCKED"}
    return {"status": "SAFE"}
```

# Source Code

---

```
# ----- WEBSOCKET -----
connections = []

@app.websocket("/ws/chat")
async def chat_socket(ws: WebSocket):
    await ws.accept()
    connections.append(ws)
    print("🟢 Client connected")

    try:
        while True:
            data = await ws.receive_text()
            for c in connections:
                await c.send_text(data)
    except WebSocketDisconnect:
        connections.remove(ws)
        print("🔴 Client disconnected")

# ----- ROOT -----
@app.get("/")
def root():
    return {"status": "Backend running"}
```

# Source Code: html & css

---

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Secure AI Chat</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

<style>
body {
  margin: 0;
  font-family: Arial, sans-serif;
  background: #efeae2;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

.chat-container {
  width: 420px;
  height: 90vh;
  background: #f0f0f0;
  display: flex;
  flex-direction: column;
  border-radius: 10px;
  box-shadow: 0 10px 30px rgba(0,0,0,0.2);
  overflow: hidden;
}
```

# Source Code

---

```
.header {
  background: #075e54;
  color: white;
  padding: 12px;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.messages {
  flex: 1;
  padding: 15px;
  overflow-y: auto;
  background: #efeae2;
}

.message {
  max-width: 75%;
  padding: 10px;
  margin-bottom: 10px;
  border-radius: 10px;
  font-size: 14px;
  line-height: 1.4;
}

.me {
  background: #dcf8c6;
  margin-left: auto;
}

.other {
  background: white;
  margin-right: auto;
}
```

# • Source Code

---

```
.system {
  background: #ffe7a3;
  text-align: center;
  margin: 10px auto;
  border-radius: 8px;
  font-size: 13px;
}

.input-area {
  display: flex;
  padding: 10px;
  background: #f0f0f0;
  border-top: 1px solid #ccc;
}

.input-area input {
  flex: 1;
  padding: 10px;
  border-radius: 20px;
  border: 1px solid #ccc;
  outline: none;
}

.input-area button {
  margin-left: 10px;
  padding: 10px 16px;
  background: #075e54;
  color: white;
  border: none;
  border-radius: 20px;
  cursor: pointer;
}

.payload {
  background: black;
  color: #00f0f0;
  font-size: 12px;
  padding: 10px;
  height: 120px;
  overflow-y: auto;
}
```

# Source Code

---

```
</style>
</head>

<body>

<div class="chat-container">
  <div class="header">
    <strong>Secure AI Chat</strong>
    <select id="userSelect" onchange="switchUser()">
      <option value="alice">Alice</option>
      <option value="bob">Bob</option>
    </select>
  </div>

  <div id="messages" class="messages"></div>

  <div class="input-area">
    <input id="messageInput" placeholder="Type a message..." />
    <button onclick="sendMessage()">Send</button>
  </div>

  <div class="payload" id="payloadBox">
    Encrypted payload will appear here...
  </div>
</div>

<script>
let currentUser = "alice";
const ws = new WebSocket("ws://127.0.0.1:8000/ws/chat");

ws.onmessage = (event) => {
  const data = JSON.parse(event.data);
  addMessage(data.sender, data.text);
};

function switchUser() {
  currentUser = document.getElementById("userSelect").value;
}
```



**PRESIDENCY  
UNIVERSITY**  
Private University Estd. in Karnataka State by Act No. 41 of 2013



# Source Code

---

```
async function sendMessage() {
  const input = document.getElementById("messageInput");
  const text = input.value.trim();
  if (!text) return;

  // Moderation check
  const response = await fetch("http://127.0.0.1:8000/moderate", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ text })
  });

  const result = await response.json();

  if (result.status === "UNSAFE") {
    addSystemMessage("Message blocked by safety system");
    input.value = "";
    return;
  }

  // Encryption demo payload
  const encryptedPayload = {
    sender: currentUser,
    text: btoa(text),
    aes_key: "AES-256-KEY-DEMO",
    timestamp: new Date().toISOString()
  };

  document.getElementById("payloadBox").innerText =
    JSON.stringify(encryptedPayload, null, 2);

  ws.send(JSON.stringify({
    sender: currentUser,
    text: text
  }));

  input.value = "";
}
```

# Source Code

---

```
function addMessage(sender, text) {  
  const msgBox = document.getElementById("messages");  
  const div = document.createElement("div");  
  div.className = "message " + (sender === currentUser ? "me" : "other");  
  div.innerText = sender + ": " + text;  
  msgBox.appendChild(div);  
  msgBox.scrollTop = msgBox.scrollHeight;  
}
```

```
function addSystemMessage(text) {  
  const msgBox = document.getElementById("messages");  
  const div = document.createElement("div");  
  div.className = "message system";  
  div.innerText = text;  
  msgBox.appendChild(div);  
}
```

```
</script>
```

```
</body>
```

```
</html>
```

# THANK YOU

