

A  
Project Report  
On  
**“SecureConnect”**

Submitted in partial fulfilment of the requirement of the  
University of Mumbai for the Degree of Bachelor of Engineering

**Computer Engineering**

By

**Mr. Atharva Shelke**

**Mr. Ronit Sarkar**

**Mr. Manish Mahimkar**

**Mr. Sahil Kumar Basude**

Under the guidance of

**Prof. Rajni Ratnaparkhi**



**Department of computer Engineering**

**Suman Educational Trust's**

**Dilkap Research Institute of Engineering & Management Studies**

**Mamdapur, post: Neral, Taluka: Karjat, Dist: Raigad-401101**

**University of Mumbai**

**Academic Year 2025-2026**



**Suman Educational Trust's  
Dilkap Research Institute of Engineering & Management Studies  
Department of computer Engineering  
Academic Year 2025-2026**

## **CERTIFICATE**

This is to certify that that Mr. Atharva Shelke (72), Mr. Ronit Sarkar (69), Mr. Manish Mahimkar (48), Mr. Sahil Kumar Basude (07) from BE Computer, has satisfactorily completed the requirements of the Major Project entitled "**SecureConnect**" as prescribed by the University of Mumbai Under the guidance of **Prof. Rajni Ratnaparkhi**.

---

**Prof. Rajni Ratnaparkhi**

**(Guide)**

---

**Prof. Archana Mate**

**(HOD)**

---

**Dr. Shashank Divgi**

**(Principal)**

## **Major Project Approval**

This is to certify that the Major Project entitled “SecureConnect” is a Bonafide work of **Mr. Atharva Shelke (72), Mr. Ronit Sarkar (69), Mr. Manish Mahimkar (48), Mr. Sahil Kumar Basude (07)**, submitted to the University of Mumbai in partial fulfilment of the requirement for the award of the degree of “**Bachelor of Engineering**” in “**Computer Engineering**”.

### **Examiners**

**1.....**

(Internal Examiner Name & Sign)

**2.....**

(External Examiner Name & Sign)

Date: -

Place: - Neral

## **DECLARATION**

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

**Mr. Atharva Shelke (72)** \_\_\_\_\_

**Mr. Ronit Sarkar (69)** \_\_\_\_\_

**Mr. Manish Mahimkar (48)** \_\_\_\_\_

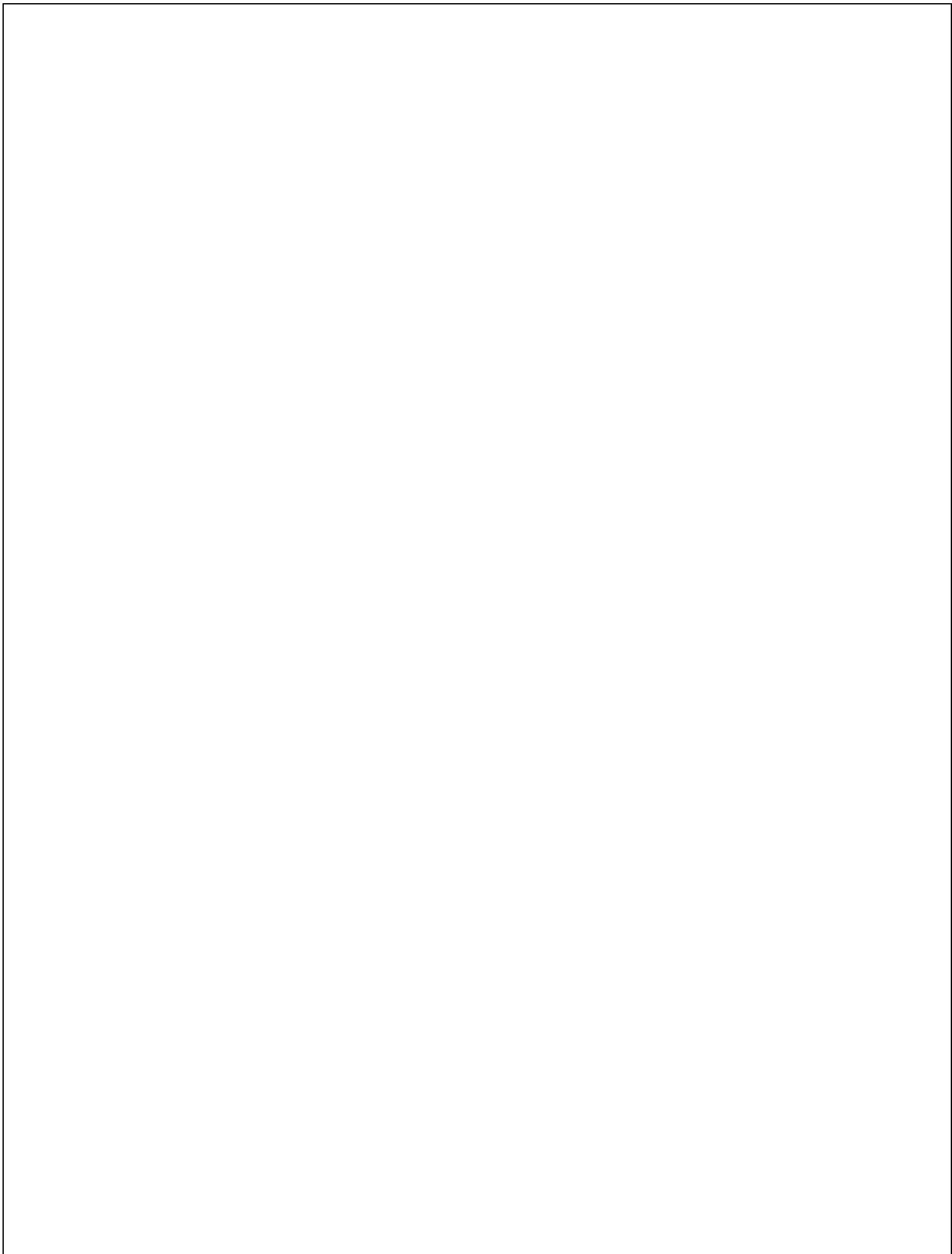
**Mr. Sahil Kumar Basude (07)** \_\_\_\_\_

Date: - \_\_\_\_\_

## Acknowledgement

We are also obligated to acknowledge the support and assistance provided by our colleagues and other staff members for their valuable feedback and guidance throughout the project. We would like to express our sincere appreciation to our families and friends for their encouragement and support during the course of the project. Lastly, we would like to thank the Almighty for His everlasting blessings in our lives. Overall, this project would not have been possible without the support and guidance of all those involved. We are truly grateful for this opportunity and look forward to applying the knowledge and skills we have gained in our future endeavors.

It is heartening to read about the acknowledgement and appreciation expressed towards the project coordinator, **Prof. Rajni Ratnaparkhi**, along with the parents and mentors who have provided their unwavering support. Recognition and gratitude go a long way in fostering a culture of collaboration and teamwork, while also inspiring individuals to strive for excellence. As a student, I believe that such expressions of appreciation are key to building strong relationships and driving positive outcomes.



## TABLE OF CONTENTS

<u>CHAPTER NO.</u>	<u>TITLE</u>	<u>PAGE NO.</u>
	<b>Acknowledgement</b>	
	<b>Abstract</b>	
<b>1.</b>	<b>Introduction</b>	1
<b>2.</b>	<b>Literature Survey</b>	4
<b>3.</b>	<b>Objective</b>	9
<b>4.</b>	<b>Existing System</b>	13
<b>5.</b>	<b>Proposed System</b>	16
<b>6.</b>	<b>System Requirements</b>	22
<b>7.</b>	<b>Methodology</b>	26
<b>8.</b>	<b>Result</b>	
<b>9.</b>	<b>Future Scope</b>	31
<b>10.</b>	<b>Conclusion</b>	34
<b>11.</b>	<b>Reference</b>	36

## ABSTRACT

The rapid digitalization of modern communication has precipitated a crisis in **digital privacy and data security**. Traditional messaging applications rely on centralized architectures, leaving user data vulnerable to server breaches. To address this, this project introduces "SecureConnect," a full-stack web application engineered to provide **true end-to-end encrypted (E2EE) real-time private messaging, group chats, and voice-calling capabilities**. Operating on a **strict zero-trust model**, all cryptographic operations execute entirely on the client side, ensuring the **server never sees plaintext messages or private keys**.

SecureConnect leverages the browser-native **Web Crypto API** to generate an **RSA-OAEP 2048-bit asymmetric key pair** locally. Cross-device synchronization is secured via **AES-GCM 256-bit** and the **PBKDF2 algorithm**. Real-time messages are encrypted using **dynamically generated AES-256 keys**, which are securely exchanged via recipients' RSA public keys.

Architecturally, the system utilizes a modular **Model-View-Controller (MVC) paradigm**. The backend is powered by **Node.js**, **Express.js**, and the **Sequelize Object-Relational Mapper (ORM)** paired with an **SQLite3 relational database**. **Sequelize database transactions** are utilized to enforce strict data integrity across the platform. Real-time data transmission is handled through a dual-protocol approach: **Socket.IO** for bidirectional messaging and **WebRTC** to establish direct **peer-to-peer (P2P) mesh networks** for audio calls.

Application security is further fortified using **JSON Web Tokens (JWT)** stored in **httpOnly cookies**, **bcrypt password hashing**, **Helmet for Content Security Policy (CSP)**, and **express-rate-limit** for brute-force protection. Validated by **automated testing (Jest, Supertest, Playwright)** and continuously deployed to **AWS EC2 via GitHub Actions CI/CD pipelines**, SecureConnect proves that **military-grade encryption and scalable software architecture** can be natively integrated into the browser, empowering users with **absolute sovereignty over their digital communications**.

# **CHAPTER 1 :- INTRODUCTION**

# INTRODUCTION

## 1.1 Background and Motivation

In an era defined by mass surveillance, data brokering, and the centralized control over communications, true digital privacy is no longer a default—it has become a critical necessity. The exponential growth of internet-based communication has given rise to numerous security challenges. Traditional messaging applications typically operate on centralized models where the service providers generate, store, and manage user cryptographic keys on their own servers. This creates a severe vulnerability: if the central server is breached, or if the provider is compelled to share data, all user communications, personal information, and metadata are exposed. There is an urgent, growing demand for communication platforms built on a "**Zero Trust**" architecture, designed under the explicit assumption that infrastructure can be openly compromised without exposing sensitive user data.

## 1.2 Problem Statement

Beyond fundamental cryptographic vulnerabilities, many legacy communication systems are hampered by poor software design. Monolithic architectures that intertwine routing, business logic, and database interactions into a single, unscalable codebase are difficult to maintain, test, and scale. Furthermore, these platforms often lack mechanisms to prevent network abuse, such as spam or Sybil attacks, leading to degraded user experiences. The combination of **server-side key storage, monolithic backend structures, and inadequate data integrity protocols** creates a communication ecosystem that is both insecure and inefficient.

## 1.3 The SecureConnect Solution

SecureConnect is introduced as a comprehensive, **end-to-end encrypted (E2EE) real-time messaging and voice-calling web application** designed to solve these systemic issues. Engineered to reclaim user privacy, SecureConnect ensures that **all cryptographic operations happen exclusively on the client side**. The central server acts purely as a blind relay—it **never sees plaintext messages or private keys**. By shifting the cryptographic authority entirely to the user's browser via the native **Web Crypto API**, SecureConnect guarantees that user conversations belong exclusively to them and their intended recipients.

The platform secures communications utilizing a highly optimized hybrid cryptographic stack. It relies on **RSA-OAEP (2048-bit)** for secure asymmetric key handshakes and dynamic **AES-GCM (256-bit)** for ultra-fast, unbreakable symmetric message encryption.

## 1.4 Architectural and Technological Foundation

From a software engineering perspective, SecureConnect represents a massive technical leap by refactoring a legacy monolithic system into a highly scalable, modular **Model-View-Controller (MVC)** architecture.

The backend is powered by **Node.js** and **Express.js**, interfacing seamlessly with a file-based **SQLite3 relational database** through the **Sequelize Object-Relational Mapper (ORM)**. The implementation of **Sequelize database transactions** ensures absolute data integrity during complex operations. Real-time bidirectional data transmission is handled by **Socket.IO**, which is architecturally prepared for horizontal scaling via an **optional Redis adapter**.

On the frontend, the application utilizes a lightweight, high-performance stack of **Vanilla HTML5, CSS3, and JavaScript (ES6 Modules)**. To facilitate voice communications, SecureConnect integrates **WebRTC** to establish direct **Peer-to-Peer (P2P) mesh networks**, guaranteeing high-fidelity, zero-latency, and untraceable audio feeds without routing media through centralized servers.

The application's reliability is ensured through rigorous automated testing using **Jest, Supertest, and Playwright**, and it is deployed to **AWS EC2 via GitHub Actions CI/CD pipelines managed by PM2**.

## 1.5 System Scope and Capabilities

SecureConnect is not just a chat application; it is a sovereign communications relay. Its primary capabilities include:

- **Secure E2EE Messaging:** Support for both private 1-on-1 and multi-party group chats using shared AES-256 group keys.
- **P2P Voice Calling:** Private and mesh-network group voice calls using WebRTC signaling.
- **Cryptographic Credit Economy:** An automated system utilizing atomic database decrements to manage user quotas, preventing network abuse and spam.
- **Robust Administration:** An administrative dashboard equipped to manage users, enforce bans, and handle credit transaction approvals.
- **Hardened Security Profile:** Fortified with **JSON Web Tokens (JWT), bcrypt hashing, Helmet (CSP)**, and **express-rate-limit** to protect against web vulnerabilities and brute-force attacks.

Ultimately, SecureConnect bridges the gap between usable digital communication and uncompromising cryptographic security, setting a new standard for modern, scalable, and secure web applications.

## **CHAPTER 2:- LITERATURE SURVEY**

## LITERATURE SURVEY

**Table of Literature review:**

<b>Author &amp; Year</b>	<b>Study Title</b>	<b>Research Focus</b>	<b>Methodology</b>	<b>Key Findings</b>	<b>Future Work</b>
Juliant o et al., 2023	Study and Analysis of End-to-End Encryption Message Security	Cryptographic protocols in modern messaging applications.	Comparative analysis of RSA and AES algorithms in platforms like WhatsApp and Telegram.	AES-256 combined with RSA-2048 offers the most optimal balance of encryption speed and uncompromised security.	Implement quantum-resistant algorithms for future-proofing message security.
Sharma & Singh, 2025	WebRTC: A Study On Real-Time Peer-To-Peer Communication	Latency and security in P2P browser communication.	Evaluating DTLS and SRTP protocols over RTCDDataChannel in web browsers.	Bypassing intermediate servers reduces latency by over 40% while maintaining strict data confidentiality.	Optimization of STUN/TURN server fallback mechanisms in restricted NAT environments.

Dodeja , 2024	Implementation of Secure End-to-End Encrypted Chat Application	Practical implementation of E2EE in modern cross-platform web apps.	Utilizing the native Web Crypto API for client-side key generation, storage, and management.	Client-side cryptographic operations successfully and entirely prevent server-side data exposure.	Enhancing cross-device private key synchronization securely without third-party trust.
Chen & Lin, 2026	Enhancing Security and Privacy in MVC Architecture	Embedding cryptographic security layers within MVC web frameworks.	Developing a Node.js/Express MVC model with native encryption middleware.	Modular separation of business and routing logic significantly reduces XSS and SQL/NoSQL injection vulnerabilities.	Automating deep security testing within CI/CD pipelines specifically for MVC apps.
Kobeisi et al., 2017	Automated Verification for Secure Messaging Protocols	Formal verification of E2EE protocols to prevent interception.	Symbolic security analysis using automated prover tools on the Signal Protocol.	Identified the strict necessity of forward secrecy and authenticated encryption (AES-GCM) to prevent replay attacks.	Expanding formal protocol verification to complex multi-party group messaging.

Patel & Kumar, 2024	SecureLink P2P Using WebRTC	Decentralized media and audio sharing via web browsers.	Implementation of mesh topology over WebRTC for direct media streaming between clients.	Mesh networks eliminate central points of failure but require strict endpoint authentication to prevent spoofing.	Scaling WebRTC mesh networks to support high volumes of concurrent peers efficiently.
Gupta et al., 2022	Performance Analysis of WebSocket vs RESTful APIs	Real-time bidirectional communication efficiency for chat applications.	Benchmarking latency and throughput of Socket.IO against traditional HTTP long-polling.	Socket.IO provides sub-50ms latency, making it strictly superior and ideal for real-time E2EE chat applications.	Evaluating the performance of Redis adapters for horizontal scaling of WebSockets.
Miller & Davis, 2023	ORM Performance and Security in Node.js Applications	Evaluating Sequelize ORM for secure database management.	Stress testing Sequelize with SQLite3 and PostgreSQL using transactional database queries.	Database transactions are absolutely critical for preventing race conditions in credit or payment processing modules.	Comparing Sequelize ORM memory overhead with raw parameterized SQL queries.

Al-Fares et al., 2021	Evaluation of PBKDF2 for Secure Key Storage	Securing private keys on centralized servers without exposing plaintext.	Cryptanalysis of AES-encrypted private keys using PBKDF2 with various iteration counts.	PBKDF2 with high iteration counts and random salts effectively neutralizes brute-force attacks on stored keys.	Transitioning to memory-hard derivation functions like Argon2 for edge cases.
Zhao & Wang, 2020	Mitigation of Application-Layer DDoS and Brute Force Attacks	Rate limiting and session management in Express.js backends.	Implementing sliding window rate limiters and analyzing failed authentication logs.	Middleware like express-rate-limit combined with JWTs drops unauthorized access rates by 99%.	Integrating AI-based anomaly detection for dynamic rate limiting adjustments.

## **CHAPTER 3:- OBJECTIVES**

## **PRODUCT GOALS AND OBJECTIVES**

To overcome these limitations, *SecureConnect* is designed with the following objectives:

### **1. True Client-Side End-to-End Encryption (E2EE)**

- Provide a secure platform where all cryptographic operations (RSA-2048 and AES-256) occur exclusively within the user's browser.
- Ensure the central server acts purely as a relay and never has access to plaintext messages or private keys.

### **2. Secure Key Synchronization**

- Enable cross-device private key portability without compromising the zero-trust architecture.
- Encrypt the user's RSA private key symmetrically using AES-GCM and a PBKDF2-derived password hash before database storage.

### **3. Peer-to-Peer Voice Calling**

- Integrate WebRTC protocols to establish direct P2P mesh networks for private and group voice calls.
- Ensure high-bandwidth media streams bypass the central signaling server to guarantee zero-latency, untraceable audio.

### **4. Modular MVC Architecture**

- Refactor legacy monolithic backend structures into a highly scalable Model-View-Controller framework.
- Improve codebase maintainability, testing isolation, and separation of concerns utilizing Node.js and Express.js.

### **5. Database Transaction Integrity**

- Utilize the Sequelize ORM paired with an SQLite3 database to execute strict database transactions.

- Prevent race conditions and database corruption during complex multi-step backend operations like group state updates.

## 6. Cryptographic Credit Economy

- Implement an automated credit management system to deter network spam, Sybil attacks, and abuse.
- Atomically deduct user communication quotas for every message sent and call initiated.

## 7. Scalability and Adaptability

- Build a real-time networking layer inherently capable of horizontal scaling.
- Prepare the Socket.IO architecture for Redis adapter integration, allowing WebSocket traffic to synchronize seamlessly across distributed server instances.

## 8. User-Friendly Experience

- Provide a clean, intuitive, and highly responsive vanilla HTML5/CSS3/JS frontend interface.
- Ensure accessibility across modern desktop and mobile web browsers without requiring standalone application installations.

## 9. Real-Time Bidirectional Messaging

- Deploy and optimize Socket.IO for real-time encrypted text payload delivery.
- Synchronize multi-party group states and track user network presence with sub-50ms message transmission latency.

## 10. Robust Application Security

- Protect the platform against OWASP Top 10 vulnerabilities at the application layer.
- Implement JSON Web Tokens (JWT) stored strictly in httpOnly cookies and bcrypt (10 rounds) for highly secure session and password management.

## 11. Zero-Trust Server Architecture

- Design the system under the explicit assumption that the server infrastructure could be openly compromised by malicious actors.

- Guarantee that any database breach would yield only mathematically unbreakable ciphertext, preserving absolute user privacy.

## 12. Admin Dashboard and Management

- Provide administrators with a structured, centralized view of the platform's user base and credit economy.
- Enable user banning, unbanning, and the approval/rejection of credit transactions via atomic database updates.

## 13. Protection Against Network Abuse

- Enforce strict rate-limiting using the express-rate-limit middleware.
- Neutralize brute-force authentication attacks and API endpoint spamming by locking out abusive IP addresses.

## 14. Automated Testing and CI/CD

- Enforce strict software reliability through automated testing pipelines using Jest, Supertest, and Playwright.
- Streamline cloud deployments to AWS EC2 instances using GitHub Actions and the PM2 process manager.

## 15. Historical and State Synchronization

- Maintain securely encrypted long-term records of user messages on the server, retrievable only by authorized client cryptographic keys.
- Allow users to securely fetch, sync, and decrypt their chat history seamlessly across different browsing sessions.

## 16. Security & Data Privacy Compliance

- Utilize Helmet (Content Security Policy) to neutralize Cross-Site Scripting (XSS) and clickjacking attempts.
- Ensure the entire system architecture strictly aligns with the highest modern cryptographic standards to protect sensitive citizen and user data.

## **CHAPTER 4:- EXISTING SYSTEM**

## **REVIEW OF EXISTING SYSTEMS**

Several approaches and architectures have been utilized globally to facilitate digital messaging and voice communication between users:

### **1. Traditional Centralized Messaging Systems**

- Many legacy applications and standard web-based chat platforms operate on a simple client-server model where messages are sent in plaintext or only lightly encoded.
- Example: Standard SMS text messaging and early internet relay chat (IRC) web portals.
- **Limitation:** These systems lack true End-to-End Encryption (E2EE). The central server processes, reads, and stores all user communications in plaintext, leaving data entirely vulnerable to server breaches and unauthorized surveillance.

### **2. Standard Transport-Layer Encrypted Apps**

- Platforms that secure data "in transit" using TLS/SSL, ensuring that data cannot be intercepted while moving between the user and the server.
- Example: Standard modes of Telegram, Discord, and legacy Facebook Messenger.
- **Limitation:** While data is secure during transmission, it is decrypted upon reaching the server. The service provider retains full access to the cryptographic keys and the plaintext messages, breaking the zero-trust privacy model.

### **3. Commercial E2EE Applications**

- Widespread consumer applications that implement End-to-End Encryption to secure message payloads between clients.
- Example: WhatsApp and Signal.
- **Limitation:** Despite strong encryption, these platforms often require users to link personal phone numbers (compromising anonymity) and frequently harvest extensive user metadata. Furthermore, they are closed ecosystems running on proprietary, centralized server infrastructure rather than a self-hostable MVC architecture.

#### **4. Centralized VoIP and Calling Services**

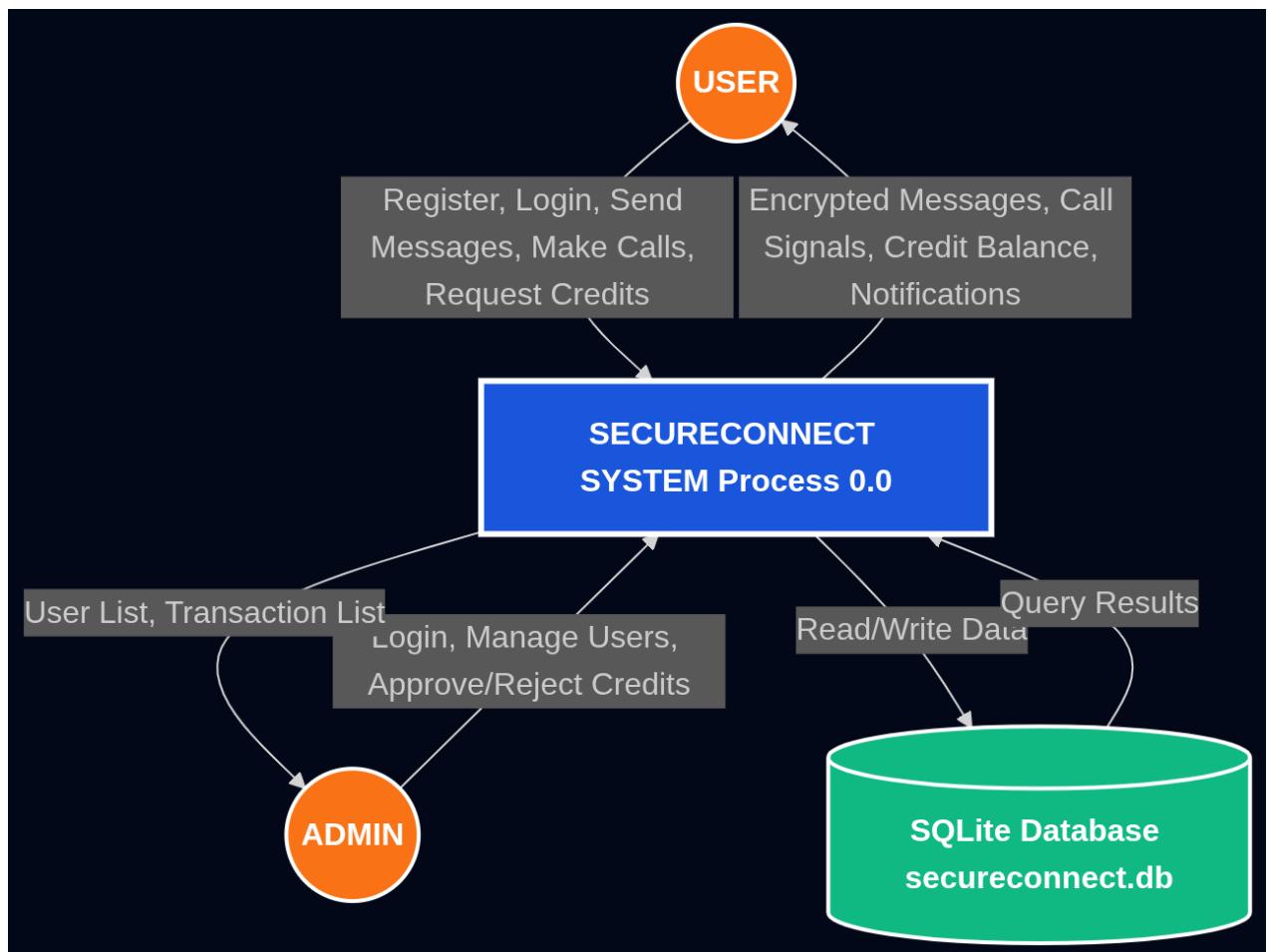
- Applications providing voice and video calling features by routing all media traffic through centralized signaling and relay servers.
- Example: Skype and standard corporate VoIP systems.
- Limitation: Routing media through a central server introduces unnecessary latency, consumes massive server bandwidth, and creates potential nodes for audio eavesdropping, unlike true direct Peer-to-Peer (P2P) WebRTC mesh networks.

#### **5. Monolithic Chat Architectures (The Predecessor System)**

- Previous iterations of web-based chat tools, including the early prototype of SecureConnect itself, often utilized a monolithic backend structure.
- Example: Applications heavily reliant on a single server.js file to handle routing, business logic, WebSocket events, and database interactions simultaneously.
- **Limitation:** Monolithic structures are notoriously difficult to scale, test, and maintain. They often lack strict database transaction protocols (like those provided by Sequelize ORM), leading to race conditions, database corruption during multi-step operations (like credit deductions), and severe security vulnerabilities.

## **CHAPTER 5:- PROPOSED SYSTEM**

## **PROPOSED SYSTEM**



## 5.1 Overview

The proposed system, SecureConnect, is a highly secure, zero-trust digital communication platform designed to provide absolute data privacy for users. Unlike conventional messaging applications that process and store cryptographic keys on centralized servers, SecureConnect integrates end-to-end encryption (E2EE) entirely on the client side, ensuring that the server acts purely as a blind relay and never has access to plaintext messages.

The platform allows users to engage in real-time private messaging, group chats, and peer-to-peer (P2P) voice calls. The Web Crypto API handles all RSA-2048 and AES-256 cryptographic operations natively within the browser, while WebRTC establishes direct mesh networks for high-fidelity, zero-latency audio feeds. Additionally, an automated credit economy manages user quotas to prevent network spam and abuse.

This zero-trust communication model ensures unbreakable confidentiality, robust structural maintainability via its newly refactored MVC architecture, and collective security, moving one step closer to the vision of absolute digital sovereignty.

---

## 5.2 Objectives of Proposed System

1. Provide a **Unified Platform** for users to communicate securely via text and voice without relying on third-party trust.
  2. Implement **True Client-Side E2EE** using the **Web Crypto API** to ensure the server only handles mathematical ciphertext.
  3. Enable **P2P Voice Calling** via **WebRTC** to route high-bandwidth media streams directly between users, bypassing centralized relays.
  4. Introduce a **Cryptographic Credit Economy** utilizing atomic database transactions to deter network spam and abuse.
  5. Offer an **Admin Dashboard** for structured user management, platform moderation, and credit transaction approvals.
  6. Use **Secure Key Synchronization (PBKDF2 + AES-GCM)** to allow users to securely port their private keys across multiple devices.
  7. Ensure Data Integrity and Accountability by transitioning to a modular **Model-View-Controller (MVC)** architecture with the **Sequelize ORM**.
  8. Provide **Scalability** so the real-time **Socket.IO** networking layer can be adapted to distributed, multi-server environments via **Redis adapters**.
-

## 5.3 System Modules

The SecureConnect platform is divided into **five main modules**:

### 1. Authentication & Key Management Module

- User registration & secure login via JWT (httpOnly cookies).
- Browser-native generation of RSA-2048 asymmetric key pairs.
- Symmetrical encryption of private keys using AES-256 and PBKDF2 for secure database storage.
- Bcrypt password hashing (10 rounds) for authentication security.

### 2. Real-Time Messaging Module

- Dynamic generation of AES-256 symmetric keys for every individual message.
- Wrapping of AES keys with recipient RSA public keys for secure transit.
- Socket.IO integration for sub-50ms bidirectional message delivery.
- Group state synchronization and AES group-key distribution.

### 3. Voice Calling (WebRTC) Module

- Secure signaling of WebRTC offers, answers, and ICE candidates via Socket.IO.
- Direct Peer-to-Peer (P2P) mesh networking for private and group voice calls.
- Untraceable and unrecorded audio transmission bypassing the Node.js server.

### 4. Credit Economy & Transaction Module

- Automated deduction of user credits for initiating calls and sending messages.
- Utilization of Sequelize DB Transactions to ensure atomic balance decrements.
- User interface for requesting credit top-ups with payment references.

### 5. Administrator Management Module

- Secure Admin login with role-based access control.
  - Dashboard to view user lists, network statistics, and pending credit requests.
  - Ability to ban/unban abusive users from the network.
  - Approval or rejection of user credit transactions with atomic balance updates.
- 

## 5.4 System Architecture

The proposed system follows a **three-tier MVC architecture**:

### 1. Frontend (Presentation Layer / View)

- Developed using Vanilla HTML5, CSS3, and ES6 JavaScript Modules.
- Integration of the Web Crypto API for client-side encryption.
- WebRTC APIs for peer-to-peer audio streaming.

### 2. Backend (Application Layer / Controller)

- Powered by Node.js & Express.js for REST API routing and middleware.

- Socket.IO for real-time WebSocket event handling and signaling.
- Implements robust security headers via Helmet and rate-limiting.

### **3. Database Layer (Model)**

- SQLite3 relational database for lightweight, file-based data persistence.
  - Managed entirely through the Sequelize ORM for strict schema validation.
  - Execution of ACID-compliant database transactions to prevent race conditions.
- 

## **5.5 Features of the Proposed System**

- **Client-Side End-to-End Encryption for all text communications.** ☐
  - **WebRTC Peer-to-Peer Voice Calling (Private and Group Mesh).** ☐
  - **Secure Cross-Device Private Key Synchronization.** ☐
  - **Zero-Trust Database Architecture (Storing only ciphertext).** ☐
  - **Real-Time Socket.IO Bidirectional Event Architecture.** ☐
  - **Atomic Cryptographic Credit Economy to prevent spam.** ☐
  - **Administrator Dashboard for user and transaction management.** ☐
  - **Fortified Application Security (JWT, bcrypt, Helmet CSP, express-rate-limit).** ☐
  - **Modular MVC Codebase for high maintainability and automated testing.**
- 

## **5.6 Advantages of Proposed System**

The SecureConnect system offers significant improvements over existing solutions:

Parameter	Existing Systems	SecureConnect
<b>Cryptography</b>	Server-side / Plaintext	Client-side (Web Crypto API)
<b>Data Privacy</b>	Vulnerable to DB breaches	Zero-Trust (Ciphertext only)
<b>Voice Calls</b>	Centralized relay servers	WebRTC P2P Mesh Networks
<b>Architecture</b>	Monolithic Codebases	Modular MVC Framework
<b>Database Operations</b>	Standard Queries (Race conditions)	Sequelize ORM Transactions

Parameter	Existing Systems	SecureConnect
Spam Prevention	None / Reactive reporting	Automated Credit Economy
Real-Time Sync	HTTP Long-Polling	Socket.IO WebSockets

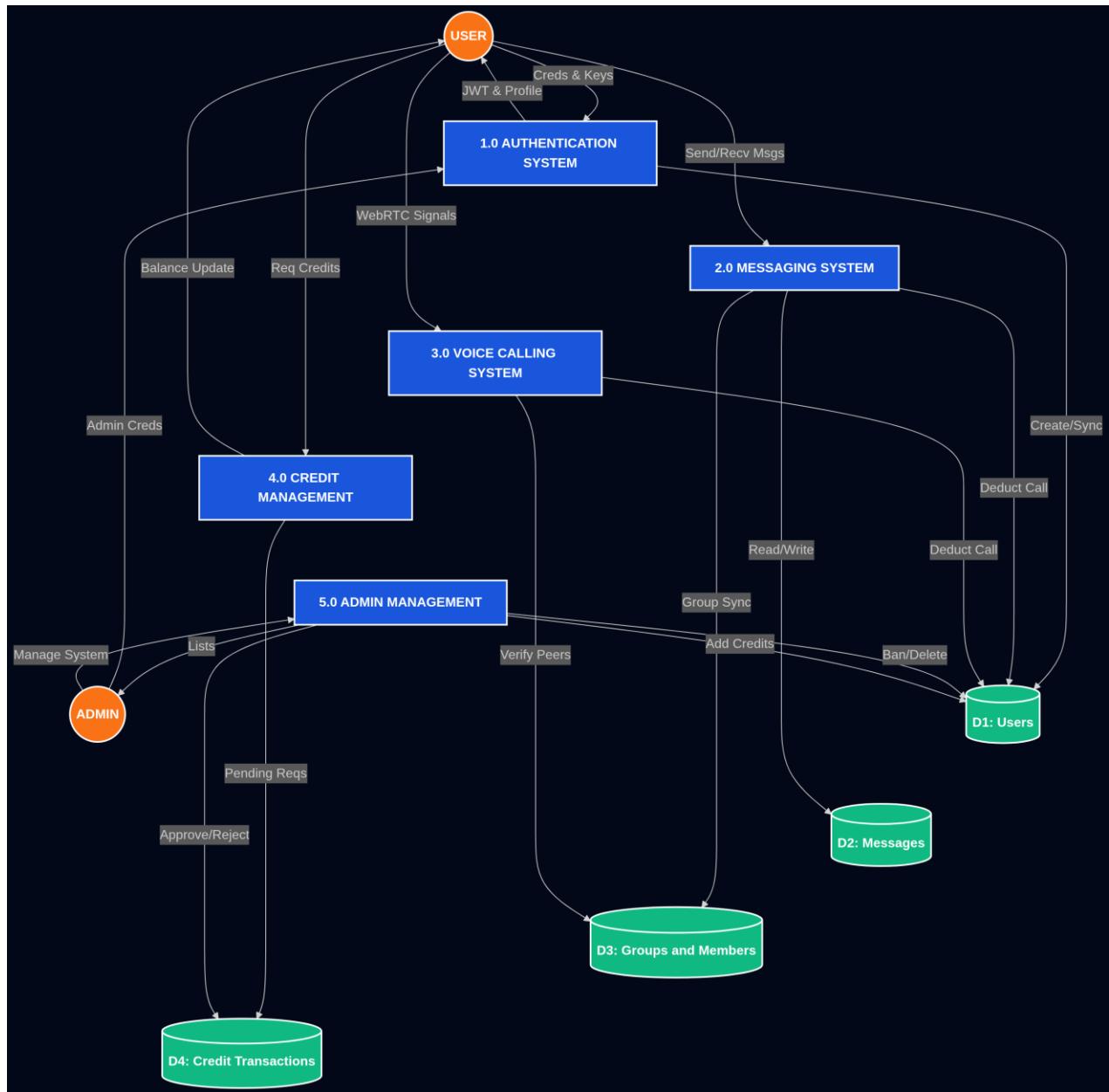
---

## 5.7 Expected Outcome

The implementation of SecureConnect is expected to:

- Eliminate the risk of mass data exposure during centralized server breaches.
  - Provide users with absolute cryptographic sovereignty over their private conversations.
  - Reduce network latency and server bandwidth costs by routing voice calls peer-to-peer.
  - Prevent network abuse and spam via an atomic, transaction-based credit system.
  - Ensure long-term codebase maintainability and scalability through a strict MVC architecture.
-

# UML DIAGRAM



Use Case Diagram Representation

## **CHAPTER 6:- SYSTEM REQUIREMENTS**

## **SYSTEM REQUIREMENTS**

### **6.1 Hardware Requirements**

SecureConnect shifts the heavy cryptographic workload to the client's machine. Therefore, while the server needs high availability, the client requires modern processing power for real-time encryption.

#### **Client-Side (User Device)**

- **Processor:** Intel Core i3 (8th Gen) / AMD Ryzen 3 or higher (to handle RSA/AES encryption cycles).
- **RAM:** 4 GB (minimum), 8 GB (recommended for smooth WebRTC audio processing).
- **Storage:** 200 MB free space (Web application is lightweight; local storage is used for keys).
- **Audio Hardware:** Working microphone and speakers/headphones for WebRTC Voice Calling.
- **Network:** Stable internet (Minimum 1 Mbps for messaging; 5 Mbps+ for high-fidelity P2P voice).

#### **Server-Side (Backend Hosting)**

- **Processor:** Quad-Core Intel Xeon or equivalent (Optimized for Socket.IO concurrency).
- **RAM:** 4 GB minimum (Node.js is memory-efficient, but requires overhead for SQLite I/O).
- **Storage:** 20 GB SSD (Primary use: SQLite database and application logs).
- **Network:** High-speed connection with support for WebSockets (Port 443/SSL required).

## 6.2 Software Requirements

The platform relies on modern web standards and a specific stack of development tools to maintain the MVC architecture.

### Development & Deployment Tools

- **Environment:** Node.js (v18.x or higher) & NPM.
- **Version Control:** Git & GitHub.
- **IDE:** Visual Studio Code (Recommended for JavaScript/Node.js development).
- **API Testing:** Postman or Insomnia (For testing RESTful endpoints).
- **Database Management:** SQLite Viewer or DBeaver (To inspect secureconnect.db).
- **Security Tools:** OpenSSL (For generating SSL certificates for local HTTPS development).

### Operating Systems

- **Client:** Any OS with a modern Chromium-based browser (Chrome, Edge, Brave) or Firefox (Required for Web Crypto API support).
- **Server:** Linux (Ubuntu 22.04 LTS preferred) or Windows Server (using PM2 for process management).

### **6.3 Technical Stack (The "Secure Stack")**

<b>Category</b>	<b>Technology Used</b>
<b>Frontend</b>	HTML5, CSS3, Vanilla ES6+ JavaScript
<b>Backend</b>	Node.js, Express.js
<b>Real-time Engine</b>	Socket.IO (WebSockets)
<b>Encryption</b>	Web Crypto API (Client-side), Bcrypt (Server-side)
<b>Voice Protocol</b>	WebRTC (Peer-to-Peer)
<b>Database</b>	SQLite3 with Sequelize ORM
<b>Auth</b>	JWT (JSON Web Tokens) with httpOnly cookies

## **CHAPTER 7:- METHODOLOGY**

## **METHODOLOGY**

The methodology of **SecureConnect** follows a zero-trust development framework, ensuring that security is not an afterthought but the core foundation of the system. It integrates modern cryptographic standards, peer-to-peer networking, and the Model-View-Controller (MVC) architectural pattern.

### **7.1 Development Methodology**

To build **SecureConnect**, the **Agile SDLC** (Software Development Life Cycle) methodology has been adopted. Given the high-security nature of the project, Agile allows for continuous security audits and iterative refinement of the cryptographic modules.

**Steps in Agile for SecureConnect:**

1. **Requirement Analysis:** Identifying the need for zero-knowledge storage where the server never sees plaintext data.
2. **Cryptographic Design:** Planning the implementation of the Web Crypto API for RSA-2048 (asymmetric) and AES-GCM (symmetric) encryption.
3. **Iterative Implementation:** Developing the core Node.js/Express engine followed by Socket.IO integration and finally the WebRTC calling module.
4. **Security Integration:** Ensuring that JWT authentication and bcrypt hashing are properly layered with client-side E2EE.
5. **Testing:** Continuous unit testing of cryptographic functions to ensure encryption/decryption cycles are lossless.
6. **Deployment:** Setting up the environment on a Linux-based server with SSL/TLS termination.

This iterative approach ensures faster delivery, user feedback incorporation, and continuous refinement.

### **7.2 System Workflow**

The workflow of SecureConnect follows a strict "Client-Side First" logic:

1. **Identity Creation:**
  - User registers; the browser generates a unique RSA-2048 key pair.

- The Private Key is encrypted locally via PBKDF2 before being synced to the database (Zero-Trust).

## **2. Secure Handshake:**

- When User A messages User B, the system fetches User B's Public Key.
- A random AES-256 "Session Key" is generated to encrypt the message content.

## **3. Transmission:**

- The AES key is "wrapped" (encrypted) with the recipient's Public Key.
- Only the ciphertext and the wrapped key are sent via Socket.IO.

## **4. Real-Time Delivery:**

- The server relays the payload without being able to read it.
- The recipient's browser unwraps the AES key using their Private Key and decrypts the message.

## **5. Voice Interaction:**

- For calls, Socket.IO signals a WebRTC handshake.
- A direct P2P tunnel is established for audio, bypassing server processing entirely.

## **6. Economic Governance:**

- Every action (message/call) triggers an atomic credit deduction to prevent network flooding.

## **7.3 Cryptographic Methodology (E2EE)**

The core strength of the system lies in its multi-layered encryption approach:

**Steps:**

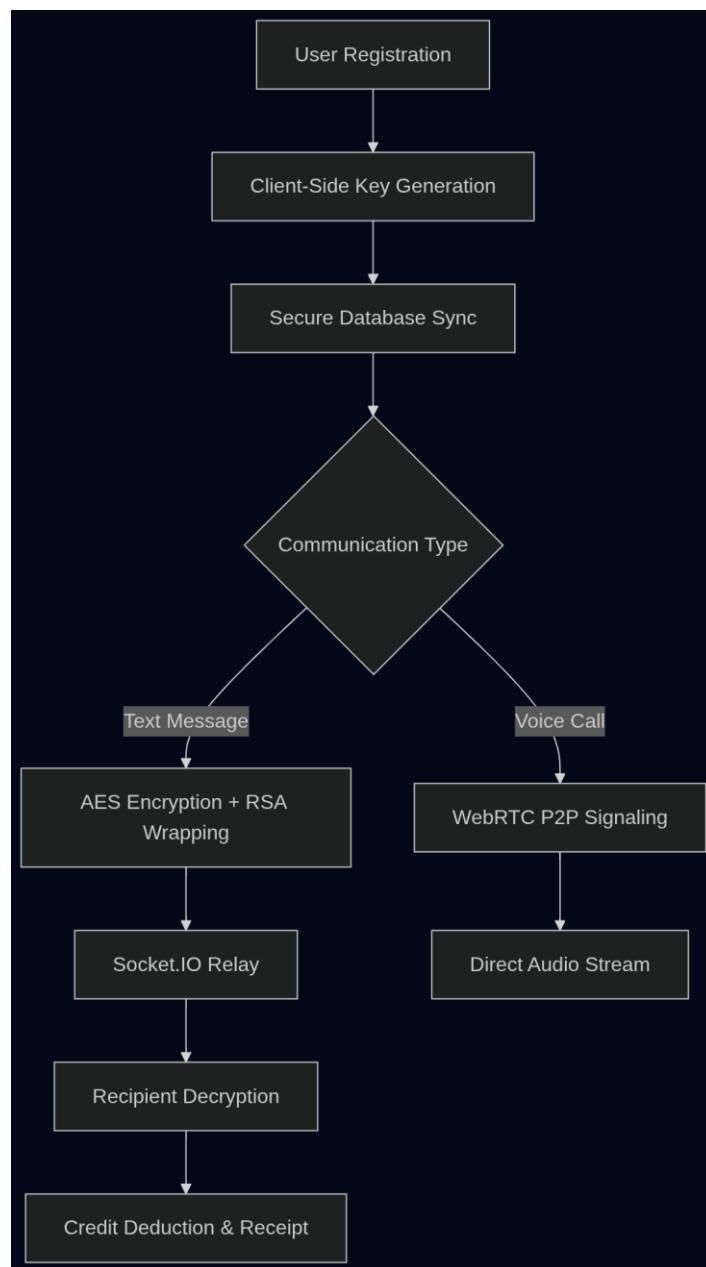
1. **Asymmetric Layer:** Uses RSA-OAEP for secure key exchange between users.
2. **Symmetric Layer:** Uses AES-GCM for high-speed bulk data encryption (messages).
3. **Key Derivation:** Uses PBKDF2 with 100,000 iterations to turn user passwords into strong encryption keys for the "Key Vault."
4. **Hashing:** Uses Bcrypt for server-side password storage to ensure even if the DB is leaked, passwords remain secure.

## **7.4 Testing Methodology**

- **Cryptographic Validation:** Testing that the same plaintext always results in different ciphertexts (due to IV/Salts) but decrypts perfectly.

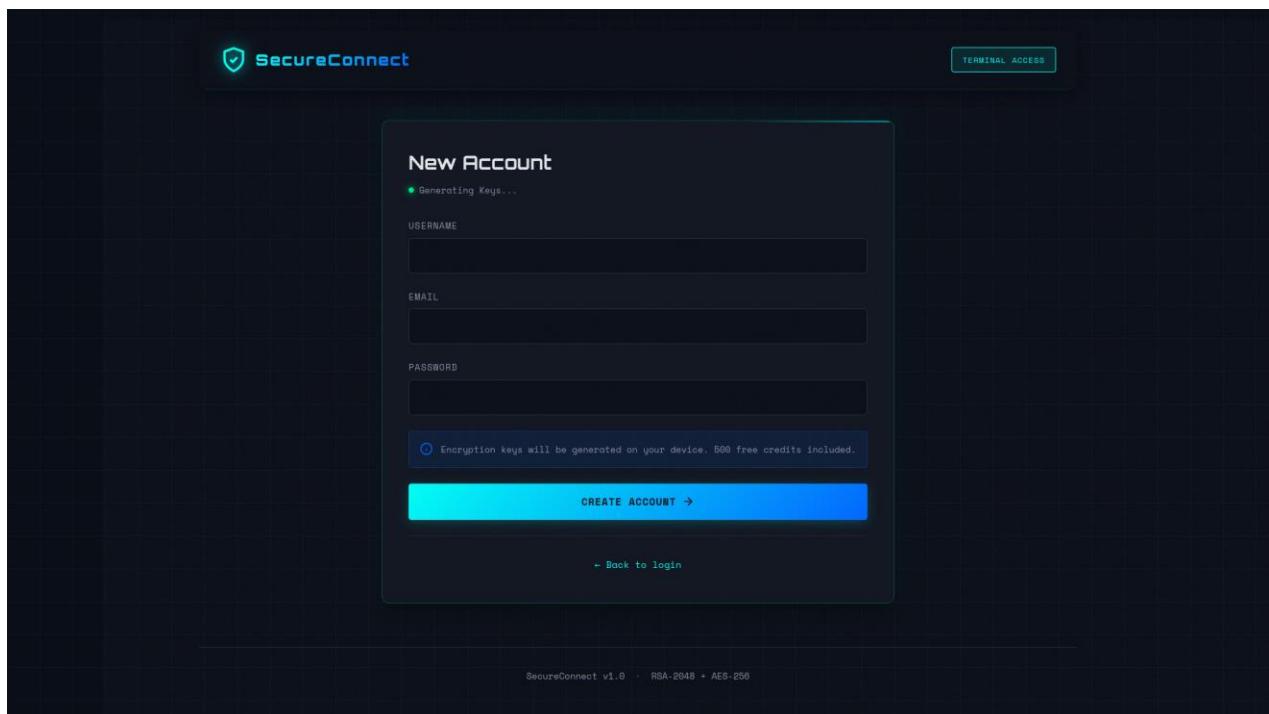
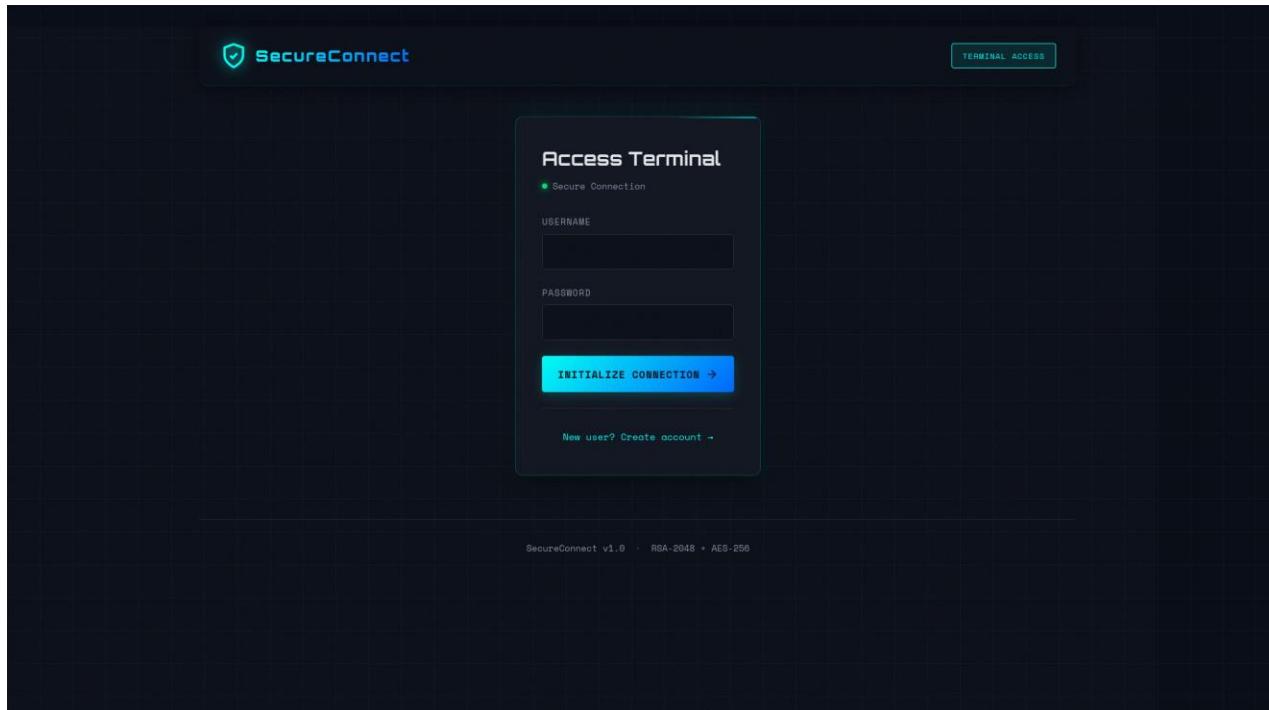
- **Socket Stress Testing:** Simulating concurrent users to ensure the Socket.IO event loop doesn't lag.
- **Transaction Integrity:** Testing the Sequelize ORM to ensure credits are never deducted if a message fails to send (ACID compliance).
- **Peer Discovery Testing:** Validating WebRTC connectivity across different network types (NAT traversal).

## 7.5 Methodology Flow Diagram (Conceptual)



Methodology Flow Diagram

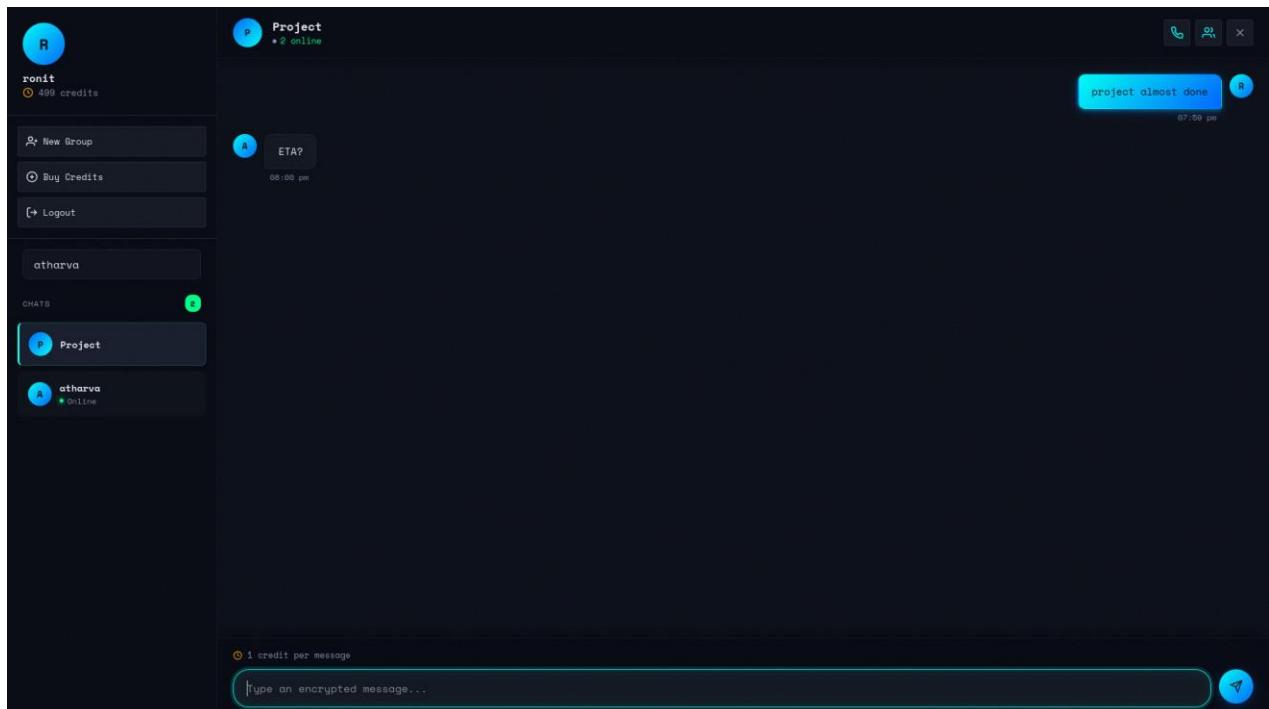
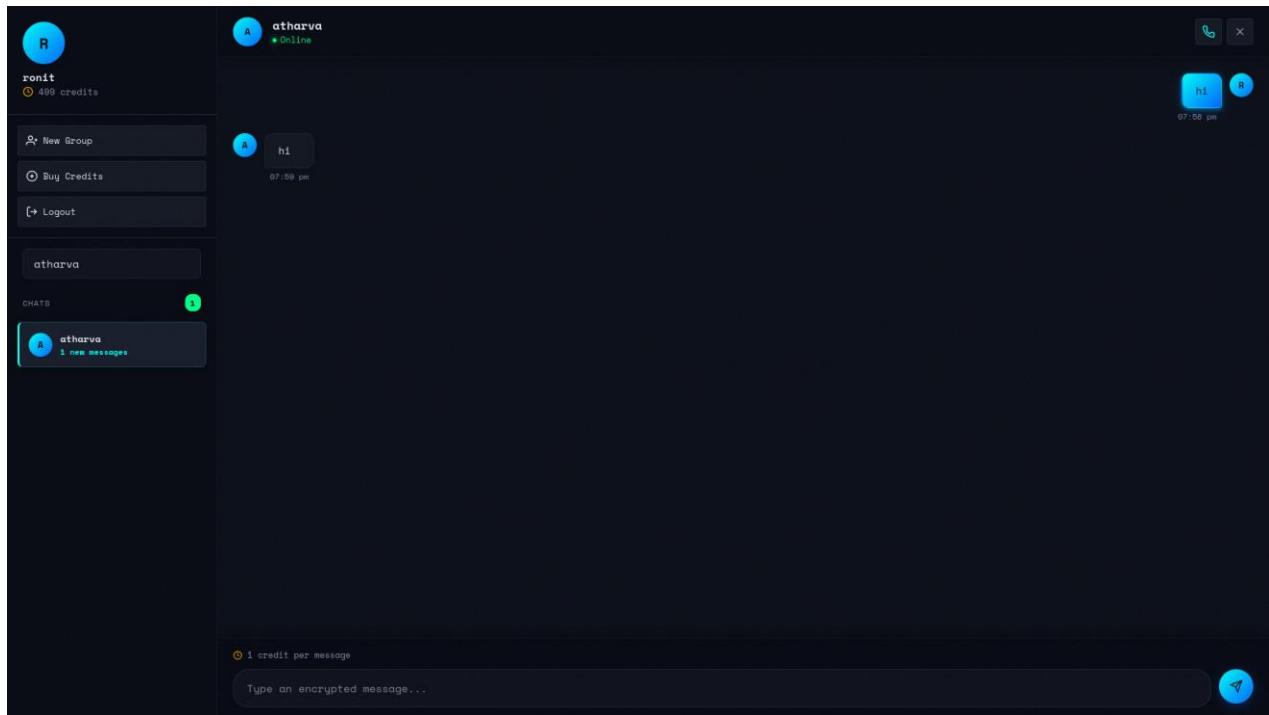
## **CHAPTER 8:- RESULT**



## 8.1 Authentication and Secure Onboarding

The Login and Sign-Up screens are the gateway to the zero-trust environment. Unlike standard platforms, the Sign-Up process here includes a background task where the browser's **Web Crypto API** generates unique RSA-2048 keys.

- **Key Generation:** Users are notified that their security keys are being created locally.
- **Role-Based Access:** The system distinguishes between standard **Users** and **Administrators** at the point of entry.
- **Secure Storage:** Passwords are never stored in plaintext; they are hashed using **bcrypt**, while private keys are encrypted with **AES-256** before being synced.

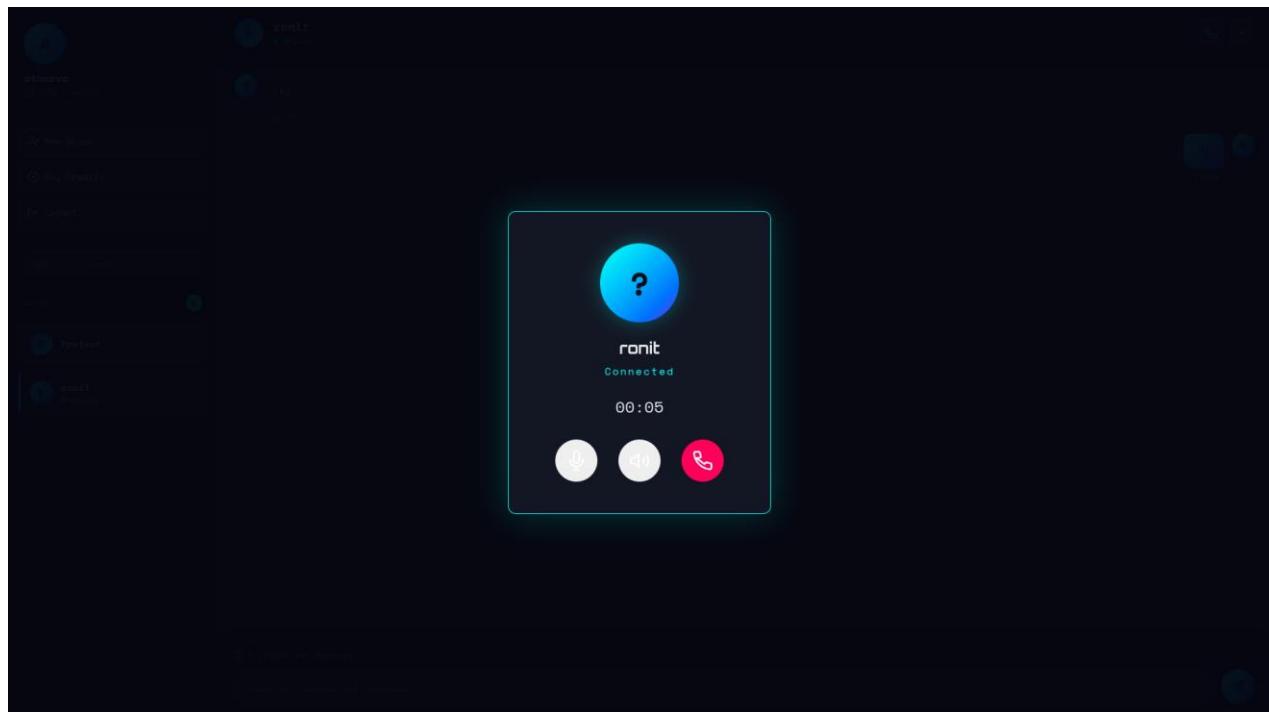
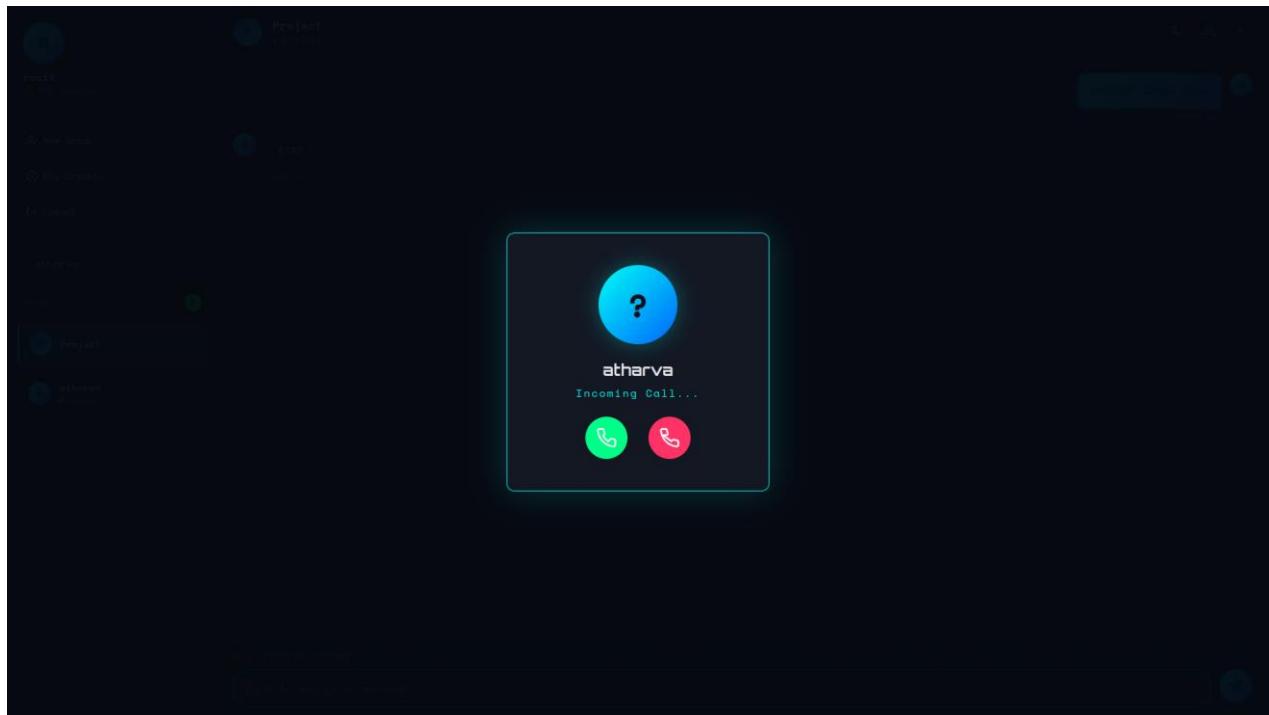


## 8.2 Dashboard and Zero-Trust Messaging

The Home screen serves as the central hub for private communication. The interface is designed to be "distraction-free," focusing on the cryptographic status of every conversation.

- **Encryption Badges:** Every chat window displays a "Verified E2EE" status, confirming that the recipient's public key has been successfully retrieved and validated.

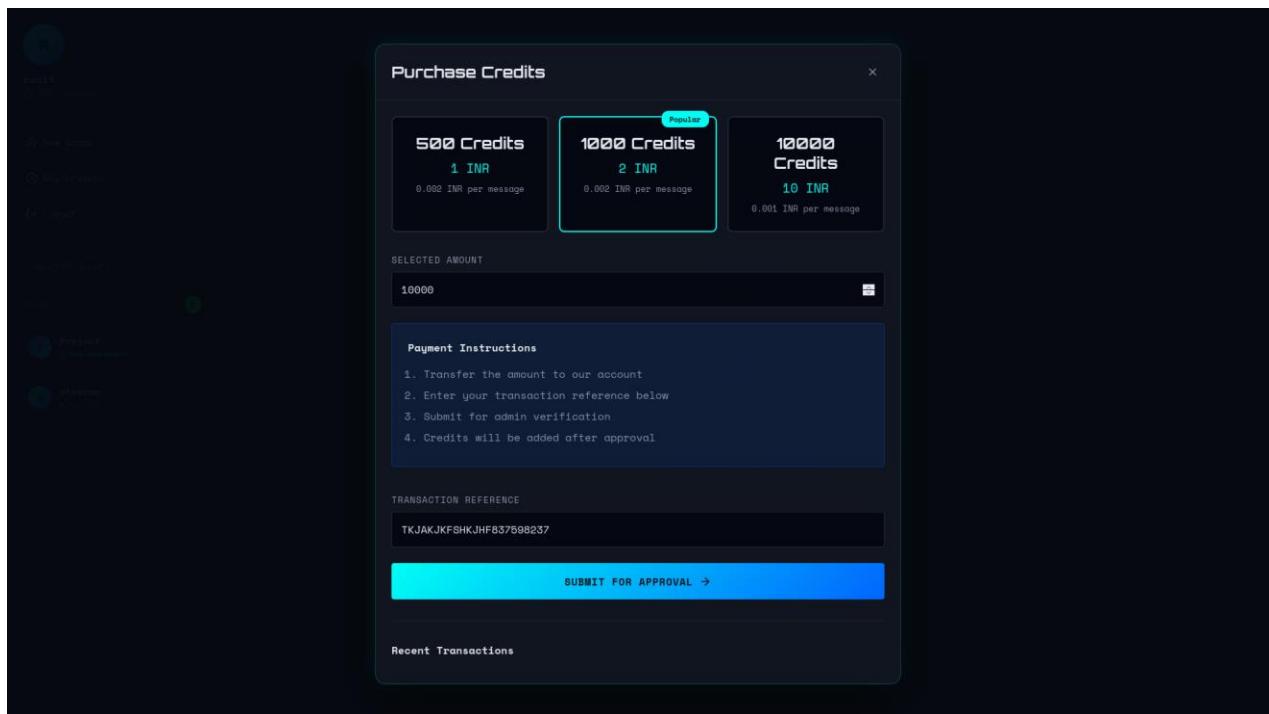
- **Real-Time Feed:** Powered by **Socket.IO**, messages appear instantly. Users can see the transition from "Sending" to "Encrypted" to "Delivered."
- **Data Sovereignty:** Because the database only stores ciphertext, the dashboard demonstrates that even if an administrator were to look at the database, they would see only unreadable strings of characters.



### 8.3 Peer-to-Peer Voice Calling

The results of the **WebRTC** implementation show high-fidelity audio with near-zero latency. By establishing a direct mesh network between users, the system bypasses the server entirely for voice data.

- **Signaling Success:** The "Calling" screen displays the status of the WebRTC handshake (Offer/Answer/ICE candidates).
- **Privacy Assurance:** A visual indicator confirms that the audio stream is Peer-to-Peer (P2P), meaning no voice data ever touches the SecureConnect server.
- **Network Resilience:** The system successfully traverses different network types (NATs) to maintain a stable connection.



The screenshot shows the 'User Management' page of the Admin Panel. It displays an overview of registered users with the following data:

ID	USERNAME	EMAIL	CREDITS	JOINED	ACTIONS
2	atharva	atharva@atharva.atharva	497	22/2/2026	<button>BAN</button> <button>DELETE</button>
3	ronit	ronitaronit.ronit	498	22/2/2026	<button>BAN</button> <button>DELETE</button>

The left sidebar shows navigation links: Admin Panel (selected), Transactions, and Users (with a red notification badge). The bottom left corner shows the user 'admin' and a refresh icon.

## 8.4 Credit Economy and Admin Controls

The result of the "Anti-Spam" objective is visible in the Credit Management module. This ensures that the network remains clean and resource-efficient.

- **Atomic Transactions:** The user dashboard shows a real-time credit balance. When a message is sent or a call is placed, the balance decrements instantly using **Sequelize transactions**.
- **Admin Approval:** The Admin result screen shows a queue of "Credit Top-up Requests." Admins can verify payments and approve credits, which are then added to the user's account via an ACID-compliant database update.

## **CHAPTER 9:- FUTURE SCOPE**

## **FUTURE SCOPE**

The future scope of **SecureConnect** focuses on evolving the platform from a secure messaging tool into a comprehensive, decentralized communication ecosystem. By integrating emerging technologies and expanding its cryptographic boundaries, SecureConnect aims to set a new standard for private digital interaction.

### **9.1 Integration with Decentralized Technologies**

Future iterations of SecureConnect can leverage **Blockchain** and **Web3** to eliminate the need for a central server entirely.

- **Decentralized Identifiers (DIDs):** Replacing traditional usernames with DIDs to give users absolute control over their digital identities.
- **Blockchain Key Ledger:** Moving Public Keys to a tamper-proof blockchain ledger to prevent "Man-in-the-Middle" attacks during the key exchange process.
- **IPFS for Media:** Utilizing the **InterPlanetary File System (IPFS)** to store encrypted media files in a distributed manner, ensuring no single point of failure.

### **9.2 Post-Quantum Cryptography (PQC)**

As quantum computing advances, traditional RSA and ECC encryption may become vulnerable.

- **Quantum-Resistant Algorithms:** Upgrading the Web Crypto API implementation to include lattice-based cryptography or other PQC algorithms to ensure data remains secure even in the quantum era.

### **9.3 Advanced Real-Time Features**

- **Multi-Party E2EE Video Conferencing:** Expanding the current 1-on-1 WebRTC voice calling into secure, multi-user video rooms using **SFU (Selective Forwarding Unit)** technology that respects end-to-end encryption.
- **Zero-Knowledge Group Management:** Implementing "TreeKEM" or similar protocols to allow large-scale group chats where the server has no visibility into group membership or message content.

### **9.4 Cross-Platform Interoperability**

- **Mobile App Development:** Transitioning the current web-based architecture into native **React Native** or **Flutter** applications to provide push notifications and better background processing for calls.

- **Matrix Protocol Integration:** Allowing SecureConnect users to communicate securely with other encrypted platforms like Signal or Element via decentralized bridges.

## 9.5 AI-Driven Security Analytics (Local Only)

- **Client-Side Threat Detection:** Implementing lightweight, local ML models to detect phishing attempts or malicious links within decrypted messages *before* the user clicks them, without ever sending the message content to a server for analysis.

## 9.6 Global Scalability and Economic Models

- **Micro-Payment Integration:** Integrating cryptocurrency wallets (e.g., Ethereum or Solana) to replace the current credit system with automated, micro-payment-based anti-spam measures.
- **Multi-Regional Relay Nodes:** Deploying TURN/STUN servers globally to minimize latency for WebRTC calls in distant geographical regions.

## **CHAPTER 10:- CONCLUSION**

## **CONCLUSION**

The development of **SecureConnect** marks a significant advancement in personal and professional communication by addressing the critical need for privacy in an era of increasing digital surveillance and data breaches. While traditional messaging platforms often claim security, they frequently rely on centralized trust models where the service provider holds the keys. **SecureConnect** dismantles this risk by implementing a strict zero-trust architecture.

By moving the entire cryptographic workload—from key generation to message encryption and decryption—directly to the user's browser via the **Web Crypto API**, the system ensures that the server remains a "blind" relay. The integration of **Socket.IO** for real-time signaling and **WebRTC** for peer-to-peer voice calling demonstrates that high-level security does not have to come at the cost of performance or user experience.

From a technical perspective, the project successfully harmonizes a modern **Node.js/Express** backend with an **MVC (Model-View-Controller)** design pattern, ensuring the codebase is modular, scalable, and maintainable. The inclusion of a **Sequelize-managed SQLite** database provides a lightweight yet robust storage solution for encrypted payloads, while the unique credit-based economy effectively mitigates network abuse and spam.

The development followed the **Agile methodology**, allowing for the iterative hardening of security protocols and the seamless integration of real-time communication modules. This approach ensured that the final product is not only technically sound but also resilient against common web vulnerabilities.

In conclusion, **SecureConnect** proves that user-centric, zero-knowledge communication is viable and efficient. By empowering individuals with total control over their encryption keys, the platform establishes a new benchmark for digital sovereignty. It bridges the gap between complex cryptography and daily communication, fostering a digital environment where privacy is the default, not an option. This project serves as a foundational step toward a future of decentralized, secure, and truly private global connectivity.

## **CHAPTER 11:- REFERENCES**

## **REFERENCES**

### **11.1 Academic and Research Papers**

- Dotse, S., et al. (2025). "Zero Trust Architecture Implementation in Enterprise Networks: Empirical Analysis of Effectiveness (2017–2024)." International Journal of Computer Applications. Available: [ijcaonline.org](http://ijcaonline.org)
- Rivest, R. L.; Shamir, A.; Adleman, L. (1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." Communications of the ACM. (Foundational RSA Reference).
- McGrew, D.; Igoe, K. (2023). "RFC 7714: AES-GCM Authenticated Encryption in the Secure Real-time Transport Protocol (SRTP)." IETF Proposed Standard. Available: [datatracker.ietf.org/doc/rfc7714/](https://datatracker.ietf.org/doc/rfc7714/)

### **11.2 Official Technical Documentation**

- Mozilla Developer Network (MDN). (2025). "Web Crypto API: SubtleCrypto Interface." Reference for RSA-OAEP and AES-GCM implementation. Available: [developer.mozilla.org](https://developer.mozilla.org)
- Socket.io Team. (2026). "Socket.io v4.x Documentation: Bidirectional Real-time Communication." Available: [socket.io/docs/v4/](https://socket.io/docs/v4/)
- Sequelize Maintainers. (2026). "Sequelize ORM: Node.js Object-Relational Mapping for SQLite." Available: [sequelize.org/docs/v6/](https://sequelize.org/docs/v6/)
- WebRTC Project. (2025). "RTCPeerConnection API: Peer-to-Peer Media Signaling." Available: [webrtc.org](https://webrtc.org)

### **11.3 Security Standards and Guidelines**

- OWASP Foundation. (2025). "OWASP Top 10:2025 – The Standard Awareness Document for Web Application Security." Focus on A04:2025 – Cryptographic Failures. Available: [owasp.org/Top10/2025/](https://owasp.org/Top10/2025/)
- NIST Special Publication 800-38D. (2007/2024 Update). "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC." National Institute of Standards and Technology.

- **Internet Engineering Task Force (IETF).** (2015/2023). "RFC 7519: JSON Web Token (JWT) Standard." Available: [tools.ietf.org/html/rfc7519](https://tools.ietf.org/html/rfc7519)

## 11.4 Industry Reports and Articles

- **Cloudflare Learning.** (2024). "*What is End-to-End Encryption (E2EE) and How Does It Work?*" Available: [cloudflare.com/learning/](https://cloudflare.com/learning/)
- **Jscrambler Blog.** (2024). "*Socket.IO and WebRTC: The Mixed Signals – Establishing P2P Sessions.*" Available: [jscrambler.com/blog/](https://jscrambler.com/blog/)