ABSTRACT

1. INTRODUCTION

**PrivChat** is an innovative chat program carefully designed to enable straightforward machine-to-machine (M2M) or peer-to-peer (P2P) communication. The fundamental design principle of PrivChat is to remove the necessity of intermediate servers for message transmission and storage of chat messages, and hence guarantee immaculate user privacy and data protection. The fundamental purpose of the application is to create a secure and straightforward pipeline for data exchange between the communicating parties, and hence it is the best solution for individuals and organizations that require exclusive control of their confidential conversations.

2 . LITERATURE SURVEY / REVIEW OF LITERATURE

Chat applications have evolved largely on the model of a centralized client-server architecture, with WhatsApp, Telegram, and Slack being excellent examples. Earlier versions used to transmit information without robust encryption, which raised serious privacy concerns. Subsequent releases introduced end-to-end encryption (E2EE) to most of these centralized systems, allegedly protecting information in transit. But even with E2EE, the metadata (who talks to whom, when, and with what frequency) is typically exposed to the server. More significantly, the server still performs sending and temporary storage of the encrypted messages, and users must have faith in the service provider not to compromise the encryption keys or the server infrastructure itself.

The advents of Web Real-Time Communication (WebRTC) have transformed the way we are able to communicate directly through the browser. WebRTC enables the user to establish direct audio, video, and data channels without the intervention of other servers to transfer the media or data. While a signaling server is still required initially to assist in getting connected (e.g., exchanging IP addresses, assisting with NAT traversal, and session data), once the direct connection is established, the data is transferred directly between peers without the server. This ability is the basis for real private communication channels. Most research balances the trade-off between the convenience of using centralized systems and the increased privacy of decentralized/P2P approaches, and WebRTC is a leading technology for making the latter become a reality on the web.

Fig 2.1 WEB RTC STRUCTURE

3 . SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Most modern chat applications, such as WhatsApp, Discord, Slack, and Facebook Messenger, rely on a centralized client-server architecture to facilitate communication between users. In this model, users interact through client applications installed on their devices—such as mobile phones or web browsers—while a central server, managed by the service provider, handles all message transmission, storage, and coordination.

When a user sends a message, the client application first encrypts the content (in apps that support end-to-end encryption) and then transmits it over the internet to the service's main server. This server receives the message, processes it, and typically stores it—at least temporarily. The storage duration depends on the app's design and features; some applications delete the message after delivery, while others retain it longer for backup, synchronization, or user accessibility across multiple devices.

After the server processes the message, it forwards it to the recipient’s client device. If the recipient is offline, the server will hold the message until the user comes online. Once the message is delivered, the recipient’s device decrypts the message and displays it in the app interface. While the message content remains private due to end-to-end encryption, the server still collects metadata—such as who sent the message, who received it, when it was sent, and the size or type of the message. This metadata, though not the content itself, can reveal significant patterns about user behavior and communication habits.

Centralized messaging systems are known for their reliability and convenience. They enable features like multi-device synchronization, group chats, and seamless notification services. They also make it easier for companies to scale up infrastructure, push updates, manage spam, and moderate content. However, this centralized control also introduces certain limitations. Since all communication flows through a single point—the central server—any server failure, compromise, or targeted surveillance can potentially disrupt services or expose user data. Moreover, while message content may be secure, the unavoidable collection of metadata poses privacy risks.

In summary, centralized chat applications offer an efficient and user-friendly communication experience, but their reliance on central servers makes them vulnerable to privacy concerns and system-wide failures. This has led to growing interest in alternative communication architectures, such as peer-to-peer technologies like WebRTC, which aim to reduce dependence on centralized systems and enhance user privacy.

Fig 3.1 Existing System

DISADVANTAGES

Despite their convenience and efficiency, centralized chat applications come with several significant disadvantages. One of the main concerns is **metadata exposure**. While end-to-end encryption protects the content of messages, the central server still collects metadata such as sender and receiver identities, timestamps, and message types. This information can reveal communication patterns and user behavior, posing serious privacy risks. Another key issue is the **single point of failure** inherent in centralized systems. If the main server experiences downtime, technical failure, or is targeted in a cyberattack, all communication within the platform may be disrupted. These systems are also vulnerable to **government surveillance and censorship**, as service providers may be required to comply with data requests or legal mandates that compromise user privacy. Furthermore, users have **limited control** over their data and communication flow, relying entirely on the policies and infrastructure of the platform. In some cases, messages and related data may be **stored long-term**, increasing the risk of data breaches or misuse for purposes such as analytics and targeted advertising. These disadvantages highlight the growing need for alternative communication architectures that prioritize privacy, decentralization, and user control.

3.2 PROPOSED SYSTEM

The proposed system, **PrivChat**, introduces a hybrid communication architecture that balances the practical benefits of centralized infrastructure with the privacy advantages of peer-to-peer (P2P) networking. Unlike traditional centralized chat applications, PrivChat limits server involvement to the initial phases of connection setup and user management, ensuring that actual message data never passes through the server. The system employs a **Node.js-based authentication and signaling server**, whose responsibilities are strictly confined to managing user registration and login using secure practices such as **bcrypt-based password hashing**, and maintaining user sessions via **express-session**. This server also plays a critical role in the signaling phase of peer-to-peer connection setup. It facilitates the discovery and connection negotiation between clients using **WebSockets** to relay signaling messages—such as WebRTC offers, answers, and ICE candidates—that help browsers establish a direct link. Importantly, these signaling messages do not contain any actual chat data.

Once the signaling process is complete and the necessary connection details are exchanged, **a direct peer-to-peer data channel is established between the users’ browsers** using **WebRTC DataChannels**. From this point onward, all communication—including text messages, files, or media—is transmitted **directly between the peers**, completely bypassing the server. This approach greatly enhances

privacy and security, as no third-party infrastructure handles or stores the chat content. Additionally, since the server is not involved in ongoing communication, it eliminates the risk of message interception, surveillance, or centralized data breaches. PrivChat thus presents a secure, efficient, and privacy-preserving solution ideal for users who demand confidentiality without sacrificing usability.

ADVANTAGES

The **PrivChat** system offers several compelling advantages, most notably its ability to ensure **absolute data privacy**. Since chat messages are transmitted directly between users through peer-to-peer (P2P) channels, they never pass through or get stored on any central server. This architecture eliminates the primary point of vulnerability found in traditional chat applications, effectively safeguarding conversations from server-side data breaches, unauthorized third-party access, and surveillance by service providers. In addition to privacy, **enhanced security** is a core strength of the system. WebRTC-based P2P communication employs **Datagram Transport Layer Security (DTLS)** by default, which encrypts all data in transit, ensuring that messages remain secure from interception during transmission.

Another key advantage is the **reduced burden on the server infrastructure**. Because the server only handles authentication and the initial connection negotiation (signaling), it is not responsible for relaying or storing large volumes of user data. This lightweight role enables better scalability and system responsiveness, especially as the number of users grows. Furthermore, **trust requirements are minimized**, as users only need to rely on the server for login and peer discovery; the confidentiality and integrity of actual messages depend solely on the end-to-end connection between users. This builds user confidence in the platform, particularly for those concerned with digital privacy.

Performance-wise, PrivChat may also deliver **lower latency** compared to traditional systems. Since messages are exchanged directly between users without detouring through a distant server, data can flow more quickly, resulting in faster message delivery and a smoother user experience. Lastly, the system demonstrates a level of **resilience** uncommon in centralized architectures. Once the peer-to-peer connection is established, communication can continue even if the signaling server becomes temporarily unavailable. Although new connections cannot be initiated during server downtime, existing sessions remain unaffected, contributing to the system's robustness and reliability.

MODULES

The **PrivChat** application is architecturally divided into five core modules, each with a distinct set of responsibilities that together support secure, private, and real-time communication between users.

The first module is the **User Management Module**, which is responsible for all aspects of user authentication and account management. This includes secure user registration, where user details—such as full name, email, username, password, and mobile number—are collected and stored in a MySQL database. Passwords are hashed using **bcrypt** before storage to ensure security. During login, user credentials are verified using **bcrypt.compare()**, allowing only authenticated users to proceed. The module also handles **session management** using **express-session**, ensuring that users remain logged in across protected pages until they explicitly log out, at which point the session is terminated. This module is built using **Express.js**, **MySQL**, **bcrypt**, and **express-session**.

The second component is the **Web Interface Module**, which forms the front-end layer of the application and facilitates user interaction. It defines and handles routing for pages such as the home screen, login, registration, user dashboard, and the chat interface. Dynamic HTML content is rendered using **Handlebars.js (HBS)**, allowing seamless integration of back-end data into the user interface. Additionally, this module serves static assets including CSS and JavaScript files, with the styling enhanced using **Bootstrap**. Technologies involved in this module include **Express.js**, **HBS**, **HTML**, **CSS**, and **Bootstrap**.

Next is the **Room Management Module**, which orchestrates the creation, joining, and lifecycle of private chat rooms. Users can generate new rooms identified by a unique PIN code, and the room creator is automatically assigned as the host. Others may join a room by entering its PIN; however, an **approval system** ensures that only users authorized by the host are allowed to enter. The module tracks active rooms, connected clients, and associated usernames to maintain the overall state. This server-side logic is implemented using **Node.js**, with **WebSockets** (via the ws library) facilitating real-time signaling and status updates.

The **Signaling Module** plays a critical role in setting up peer-to-peer connections by acting as the temporary communication bridge for signaling data. It includes a **WebSocket server** that handles all incoming client connections and parses signaling messages such as room creation, join requests, WebRTC offers, answers, and ICE candidates. These signaling messages do not contain chat data; instead, they carry connection metadata required for WebRTC peers to negotiate a direct connection. The server forwards these messages between clients in a specific room and maintains a mapping of sockets to users and rooms to ensure accurate communication. This module is implemented using **JavaScript** and the **ws** WebSocket library.

Finally, the **Peer-to-Peer Data Channel Module** operates entirely on the **client side** and is responsible for the actual real-time communication between users. Using the **WebRTC API**, the module initializes and manages an **RTCPeerConnection** object and establishes a **DataChannel** for sending and receiving chat messages. Once connected, all messages are transmitted directly between browsers without any server intervention. The module also updates the chat interface in real-time, rendering both incoming and outgoing messages with clear visual distinctions, including sender identification. This setup ensures privacy, speed, and a fluid chat experience, leveraging native **JavaScript** and **WebRTC** technologies.

Together, these modules provide a scalable, secure, and privacy-respecting communication system that blends server-side control with the decentralized power of peer-to-peer networking.

4.FEASIBILITY STUDY

The development and deployment of the **PrivChat** hybrid peer-to-peer communication system is both technically and economically feasible, offering significant advantages in privacy and scalability over traditional centralized chat applications.

**4.1 Technical Feasibility**

From a technical standpoint, PrivChat leverages mature and widely supported technologies such as **Node.js**, **Express.js**, **MySQL**, and the **WebRTC API**. The use of **WebSockets** for signaling and **WebRTC DataChannels** for peer-to-peer communication aligns with modern web standards and enjoys broad compatibility across major web browsers and platforms. The implementation of secure user authentication through **bcrypt** and session management with **express-session** provides a strong foundation for protecting user credentials and managing access control.

The system’s architecture is modular, which simplifies development, testing, and future scalability. By limiting the server’s responsibilities to authentication and signaling, PrivChat avoids common pitfalls of centralized systems such as server overload and single points of failure in message transmission. Additionally, the direct peer-to-peer data channels enable low-latency communication while maintaining end-to-end encryption enforced by WebRTC’s built-in DTLS protocol. This ensures that even in cases of high user load, the server remains responsive and communication privacy is maintained.

**4.2 Economic Feasibility**

Economically, PrivChat reduces server infrastructure costs significantly by offloading the bulk of message transmission to direct client connections. This not only lowers bandwidth and storage expenses but also simplifies server maintenance and scaling efforts. The signaling server requires only minimal resources since it handles metadata and connection setup rather than message content. Furthermore, the use of open-source technologies and frameworks contributes to minimizing licensing and development costs.

The system also meets the growing market demand for privacy-focused communication platforms, potentially attracting users who prioritize confidentiality and control over their data. This trend could open up opportunities for monetization through premium features, enterprise solutions, or privacy-centric service tiers without compromising the core principles of user privacy.

**4.3 Operational Feasibility**

PrivChat’s design emphasizes user privacy and ease of use, which enhances operational feasibility. The hybrid approach balances the need for central authentication with peer-to-peer direct communication, simplifying user onboarding while safeguarding message content. Users benefit from intuitive web interfaces and responsive real-time chat experiences, supported by robust backend services. Moreover, the system’s resilience—allowing ongoing communication even if the signaling server temporarily fails—improves reliability and user satisfaction.

**Conclusion**

Overall, the PrivChat system is a technically viable, economically sustainable, and operationally practical solution for secure and private real-time communication. Its architecture not only addresses current challenges in centralized messaging but also anticipates future needs for scalability, privacy, and user empowerment. This forward-thinking design positions PrivChat as a promising project in the evolving landscape of secure communication technologies.

5 . SOFTWARE REQUIREMENT SPECTIFICATION

5.1 HARDWARE REQUIREMENT

The **PrivChat** system has been designed to run efficiently with modest hardware, making it accessible for both developers and end users. The hardware requirements are divided into two primary categories: the **server-side infrastructure** and the **client-side user devices**.

On the server side, the system requires a reliable and capable machine to handle signaling, user authentication, session management, and room coordination. A **quad-core CPU or higher** is recommended to manage concurrent WebSocket connections and handle multiple user requests without performance degradation. At least **8GB of RAM** is necessary to ensure smooth operation, especially under moderate to heavy user load. The server should be equipped with a minimum of **250GB of SSD storage**, which allows sufficient space for the operating system, Node.js environment, MySQL database, log files, and other essential dependencies. Since the server is also responsible for relaying signaling messages during WebRTC connection setup, a **stable and high-bandwidth internet connection** is essential to maintain low latency and high availability.

On the client side, users can connect to the PrivChat platform using standard computing devices with minimal resource requirements. A **dual-core processor** and at least **4GB of RAM** are sufficient for a smooth experience, as most of the chat functionalities are handled directly in the browser using JavaScript and WebRTC. The system is lightweight in terms of storage, as it does not store any chat messages locally—communications are transient and directly exchanged over peer-to-peer channels. Users also need a **stable internet connection** to establish and maintain real-time data channels between peers. Although not required for the current text-based chat features, optional **webcam and microphone** support would be necessary in future versions that include audio or video calling capabilities.

Overall, the hardware requirements are deliberately kept lightweight to ensure ease of deployment and broad compatibility, aligning well with the privacy-focused, decentralized nature of the PrivChat application.

5.2 SOFTWARE REQUIRMENT

The development and deployment of the **PrivChat** application requires a well-defined software environment that supports both server-side and client-side operations. The system is designed to be **platform-independent**, allowing development and deployment across **Windows**, **Linux**, and **macOS** operating systems. This cross-platform compatibility ensures flexibility for both developers and users during setup and usage.

For the **backend environment**, **Node.js** serves as the core runtime engine. It is recommended to use a long-term support (LTS) version such as **v18.x or v20.x** to ensure stability and ongoing updates. The **Node Package Manager (npm)** is used extensively to manage project dependencies. Key libraries include **Express.js**, which handles HTTP routing and middleware logic, and **ws**, a lightweight and efficient WebSocket library that enables real-time signaling communication between users during peer connection setup. Additional packages include **bcrypt**, which is essential for securely hashing user passwords, and **express-session**, which facilitates secure and persistent user sessions across the application lifecycle. Database interactions are managed using the **mysql** Node.js driver, while dynamic HTML content rendering is handled by **Handlebars.js (hbs)** as a server-side templating engine.

The backend is powered by a **MySQL database server**, with version **8.0 or higher** recommended for optimal performance and compatibility. Developers can use tools like **MySQL Workbench** or other MySQL clients to design the database schema, execute queries, and manage data administration tasks.

On the **frontend**, the application is built using standard web technologies such as **HTML5**, **CSS3**, and **JavaScript**. For enhanced responsiveness and visual design, **Bootstrap 5.3** is used, offering a rich set of pre-designed UI components and a mobile-first layout system. The core technology enabling real-time peer-to-peer communication is the **WebRTC API**, which is natively supported by all modern browsers including **Google Chrome**, **Mozilla Firefox**, **Microsoft Edge**, and **Apple Safari**. WebRTC is vital for establishing secure and efficient data channels between connected peers without routing message data through a server.

In terms of **development tools**, a robust code editor such as **Visual Studio Code**, **Sublime Text**, or any IDE with JavaScript and Node.js support is recommended. A modern web browser is also essential for testing and debugging the frontend application, especially to ensure compatibility and performance of the WebRTC-based peer-to-peer features.

Overall, the software stack used in PrivChat is composed of **open-source, modern, and widely supported technologies**, making the system not only cost-effective but also scalable and easy to maintain. This environment ensures that the development process is streamlined, and the application remains robust and secure throughout its lifecycle.