Chapter 1

INTRODUCTION

In the digital era, communication systems have evolved drastically, enabling people to connect instantly across the world through chat applications, messaging systems, and social platforms. However, this advancement has also given rise to a new set of challenges related to security, privacy, integrity, and protection of sensitive information. Traditional messaging platforms often rely on server-side data storage and minimal encryption, leaving conversations exposed to potential risks such as unauthorized access, data leakage, man-in-the-middle attacks, eavesdropping, and message manipulation. As users increasingly rely on digital communication for personal, financial, and professional matters, ensuring the confidentiality and authenticity of chat messages has become more important than ever.

To address these concerns, cryptographic techniques are widely adopted to provide secure communication channels. Cryptography plays a vital role in protecting messages, securing key exchanges, ensuring confidentiality, verifying identities, and preventing unauthorized access. However, even with encryption, traditional chat systems still struggle with issues like message tampering, loss of integrity, and server-side manipulation. This leads to situations where chat messages can be altered, deleted, or injected without the user's knowledge. The absence of message-level integrity verification presents a major vulnerability in modern communication systems.

To overcome these limitations, the concept of **Chain Chat with Cryptography** has been introduced. This approach combines modern cryptographic algorithms with a blockchain-inspired message-chaining mechanism that ensures every message is securely encrypted and linked to the previous one. This creates a tamper-proof chain of messages where even a slight modification in one message breaks the entire chain, instantly revealing any manipulation attempt. Chain Chat uses encryption algorithms like **AES for secure message encoding**, **RSA or ECC for key exchange**, and **SHA-256 for hashing and message chaining** to provide maximum confidentiality and integrity.

The fundamental idea behind Chain Chat is to ensure that each message is not just encrypted but also recorded as a secure link in a communication chain, similar to how blocks are linked in a blockchain. Every message contains:

1. The encrypted message body
2. A cryptographic hash of the message
3. The hash of the previous message

This structure ensures that no attacker or unauthorized entity can alter, insert, or delete any message without immediately breaking the chain. Furthermore, end-to-end encryption prevents unauthorized reading of the

message content, making the system highly resilient against modern cyber threats.

Another crucial aspect of the system is secure key management. Key exchange mechanisms like Diffie-Hellman, RSA, or Elliptic Curve Cryptography ensure that messages are encrypted only with the intended recipient's key. As a result, even if an attacker intercepts the conversation, they cannot derive the message content without possessing the correct decryption key.

The motivation behind developing a system like Chain Chat stems from the increasing cybercrimes and data breaches targeting messaging systems. Attackers exploit weak encryption and vulnerabilities in chat applications to access sensitive information such as banking data, personal conversations, authentication codes, or corporate information. Chain Chat aims to build a robust communication platform where data privacy is not just a feature but a default guarantee.

In industries like healthcare, finance, defense, education, and corporate sectors, secure communication is essential for maintaining confidentiality and protecting sensitive information. A chat system that ensures both encryption and tamper-proof message storage can significantly enhance trust and reliability among users. Chain Chat with Cryptography is designed to meet this need by offering a system where every message is protected using advanced cryptographic techniques, ensuring that no one—including the service provider—can access or modify the data.

In addition to real-time communication, this system can be extended to secure logs, authentication requests, transaction messages, IoT communications, and more. Its flexible design makes it suitable for various modern security applications. By implementing Chain Chat, users can communicate without the fear of data breaches, manipulation, or surveillance.

Overall, this project focuses on designing a secure, transparent, and tamper-proof messaging system that leverages cryptography and hash-based chaining. It not only protects messages from unauthorized access but also ensures that every message remains verifiable and unchanged throughout the communication process. Thus, Chain Chat provides a comprehensive security model for modern communication needs.

1.1 Objectives

1. To develop a secure chat system using strong cryptographic algorithms to protect message confidentiality.
2. To implement end-to-end encryption ensuring that only the intended users can read the messages.
3. To apply hash chaining mechanisms that prevent any modification, deletion, or insertion of chat messages.

4. To ensure secure key exchange using RSA/ECC for reliable cryptographic communication.
5. To verify message integrity through SHA-256-based hashing for every message in the chat chain.

1.2 Motivations

1. Rapid increase in cyberattacks targeting unprotected chat systems and user communications.
2. Need for a tamper-proof messaging system where message integrity is guaranteed through cryptographic chaining.
3. Lack of complete end-to-end encryption in many widely used messaging platforms.
4. Growing concerns about privacy, data leakage, and unauthorized message interception.
5. Motivation to utilize cryptographic algorithms in a real-world practical system to secure communication.
6. To ensure confidentiality, integrity, and authenticity in chat communication across all users.

1.3 Summary

In today's digital world, chat applications are widely used for personal, professional, and financial communication. However, traditional messaging platforms face major security problems such as data leakage, message tampering, eavesdropping, unauthorized access, and server-side manipulation. Many chat systems rely on centralized servers and basic encryption, which makes them vulnerable to cyber-attacks.

To solve these issues, Chain Chat with Cryptography is introduced. This system combines strong encryption with a blockchain-inspired message chaining technique to ensure fully secure communication.

Chapter 2

LITERATURE SURVEY

Secure communication has become a critical requirement in the digital age, where chat applications and online messaging have become the primary mode of communication for millions of users. With the rise of cyber threats such as eavesdropping, unauthorized message access, tampering, and man-in-the-middle attacks, ensuring message privacy and integrity has become a major challenge. Traditional chat systems often rely on basic encryption or server-based security models, which are insufficient to protect communication against modern, sophisticated attacks.

Cybercriminals increasingly target messaging platforms to intercept or manipulate confidential information, including personal conversations, financial details, authentication codes, and business-related data. The lack of message-level integrity verification, weak encryption algorithms, and centralized storage make these systems vulnerable to data breaches and unauthorized modifications.

To overcome these challenges, researchers have explored the use of cryptographic algorithms and blockchain-inspired chaining mechanisms to enhance the security of communication systems. Cryptography ensures confidentiality through encryption techniques such as AES, RSA, and ECC, while hashing algorithms like SHA-256 ensure message integrity. Blockchain-inspired message chaining further strengthens security by linking every message to its predecessor through cryptographic hashes, making the entire chat tamper-proof.

Existing studies demonstrate that combining encryption with hash-chaining significantly improves communication security by:

- Preventing unauthorized message modification
- Detecting message insertion or deletion
- Ensuring tamper-resistant communication history
- Eliminating single points of failure
- Securing key exchange and authentication

Due to these advantages, Chain Chat with Cryptography has emerged as an effective approach for secure, transparent, and tamper-proof messaging systems.

2.1 Research Behind Defining the Problem

To define the problem accurately, extensive literature surveys and analysis of existing communication systems were conducted. Key research findings include:

- Cyberattacks on chat applications are increasing rapidly, targeting sensitive conversations and personal information.
- Traditional chat systems rely heavily on simple encryption, which fails against stronger, modern cyber threats.
- Most messaging apps store chat data centrally, making them vulnerable to server-side attacks, insider threats, or database leaks.
- Researchers have shown that cryptographic mechanisms such as AES encryption, RSA key exchange, and SHA-256 hashing significantly increase communication security.
- Studies highlight that hash chaining makes message tampering detectable, similar to blockchain technology.
- Existing systems lack message-level integrity verification, allowing attackers to modify or remove messages without detection.
- Research suggests that digital signatures and secure key exchange protocols enhance authentication and prevent impersonation attacks.

These findings highlight the need for a secure, tamper-proof, and cryptographically enforced communication model that protects both message confidentiality and integrity.

2.2 Existing System

1. Basic Encryption-Based Chat Systems

- Messages are encrypted using simple algorithms
- Encryption keys may be stored on the server
- Limited protection against message tampering

Limitations:

- Messages can be decrypted if the server is compromised
- No verification if the message has been altered
- Vulnerable to man-in-the-middle attacks
- Keys are not always securely exchanged

2. Server-Based Secure Messaging

- Messages are stored on the server in encrypted format
- Server validates users and manages keys

Limitations:

- Server becomes a single point of failure
- Insider attacks may compromise message privacy
- Attackers who gain server access can modify chat history
- No cryptographic link between messages

3. End-to-End Encrypted Systems (e.g., WhatsApp, Signal)

- Messages encrypted from sender to receiver
- Uses strong cryptographic algorithms
- Supports key verification

Limitations:

- Does not prevent message deletion or post-delivery modification
- No message-chaining to ensure tamper-proof history
- Metadata can still be accessed
- Backup systems may not be fully encrypted

4. Heuristic-Based Message Verification

Some systems use heuristic or rule-based checks:

- Detect suspicious login attempts
- Check for abnormal communication patterns
- Verify session integrity

Limitations:

- Cannot ensure cryptographic tamper-proofing
- Rules may not detect advanced attacks
- High false positives in certain conditions

5. Blockchain or Hash-Chaining Based Secure Messaging

Modern research explores:

- Hash-chaining of messages
- Linking each message to the previous one
- Using cryptographic hashes (SHA-256)
- Ensuring message integrity and detectability

Benefits:

- Any modification breaks the chain
- Attackers cannot delete, edit, or insert messages
- Provides a verifiable, tamper-proof history

Limitations of Existing Implementations:

- High computational cost in some designs
- Difficult to scale in public blockchain environments
- Requires optimized architecture for real-time chat

2.3 Summary

The literature survey focuses on the growing need for secure communication in modern chat applications. As cyberattacks like eavesdropping, message interception, tampering, and man-in-the-middle attacks increase, traditional chat systems are unable to provide full security. Many existing messaging platforms use basic encryption or depend heavily on centralized servers, which makes them vulnerable to data breaches and unauthorized modifications.

Researchers have studied how cryptography and blockchain-inspired hash chaining can strengthen message security. Cryptographic algorithms like AES, RSA, ECC, and hashing methods like SHA-256 help protect message confidentiality and integrity. Hash-chaining links each message to the previous one, making the entire chat history tamper-proof.

Chapter 3

PROBLEM STATEMENT AND SCOPE

3.1 PROBLEM STATEMENT

Modern chat applications often rely on centralized servers and basic encryption, making them vulnerable to cyber threats such as message tampering, interception, deletion, and unauthorized access. Even platforms with end-to-end encryption fail to provide message-level integrity, allowing attackers or compromised servers to alter or remove chat data without detection.There is a need for a secure communication system that not only encrypts messages but also ensures that the entire chat history remains tamper-proof, verifiable, and protected from manipulation.

3.2 PROPOSED SYSTEM DESIGN

In today's digital era, chat applications have become the primary mode of communication for personal, professional, and commercial interactions. However, traditional chat systems rely heavily on centralized servers and basic encryption mechanisms, making them vulnerable to various cyber threats. Attackers can intercept, modify, delete, or insert unauthorized messages during transmission, leading to privacy breaches, impersonation attacks, and tampering of chat history.

Existing messaging platforms often provide end-to-end encryption but still lack message-level integrity verification. This allows malicious actors or even compromised servers to alter chat data without the user's knowledge. Furthermore, centralized databases storing message logs are susceptible to insider attacks, data manipulation, and unauthorized access. As a result, ensuring confidentiality, authenticity, non-repudiation, and tamper-proof communication remains a challenge.

To address these limitations, there is a need for a secure communication model that not only encrypts chat messages but also protects the integrity of the entire conversation. **Chain Chat with Cryptography** aims to provide this security by integrating cryptographic algorithms with a blockchain-inspired message-chaining technique. Each message is encrypted and linked to the previous one using cryptographic hashes, forming a secure, immutable chain. Any modification to a message breaks the chain, making tampering immediately detectable.

Thus, the problem lies in designing a system that ensures:

- End-to-end encryption for message confidentiality
- Tamper-proof storage using message hashing and chaining
- Secure key exchange and authentication
- Protection from common communication attacks (MITM, message injection, replay attacks)
- A scalable and efficient chat mechanism suitable for real-time usage

The proposed Chain Chat system aims to solve these challenges by creating a fully secure, transparent, and integrity-protected digital communication platform.

3.3 SCOPE OF PROPOSED SYSTEM

The proposed Chain Chat with Cryptography system aims to provide a secure, tamper-proof, and efficient communication platform by integrating strong encryption and blockchain-inspired message chaining. Its scope includes:

1. Secure End-to-End Communication

The system ensures complete privacy by encrypting all messages using AES, RSA, or ECC so that only the intended users can read them.

2. Tamper-Proof Message History

Hash chaining (similar to blockchain) links every message to the previous one.

This prevents:

Message modification

Message deletion

Unauthorized message insertion

Any tampering becomes immediately detectable.

3. Secure Key Management

Provides reliable key exchange mechanisms to prevent impersonation or man-in-the-middle attacks. Private keys always remain protected.

4. Real-Time Secure Chat

Supports real-time messaging while maintaining cryptographic security, making it suitable for personal, enterprise, or institutional communication.

5. Safe Message Storage

Messages are stored in an encrypted format along with their hash and previous hash, ensuring long-term integrity even if the database is compromised.

6. Scalable Architecture

The system can be extended for:

Group chats

Organizations

Multi-user environments

Large-scale secure messaging systems

Chapter 4

SYSTEM METHODOLOGY

4.1Module Description

The proposed Chain Chat with Cryptography system is divided into several key modules. Each module performs a specific function to ensure secure, encrypted, and tamper-proof communication.

Module 1. User Authentication Module

This module manages user registration and login.

Functions

Allows users to create accounts using username, email, and password.

Passwords are hashed using secure algorithms (e.g., SHA-256).

Validates credentials during login.

Prevents unauthorized users from accessing the chat system.

Module 2. Key Management Module

This module handles cryptographic key generation and exchange.

Functions

Generates RSA/ECC public–private key pairs.

Exchanges session keys securely using RSA or Diffie–Hellman.

Stores private keys securely on the client side.

Verifies the authenticity of public keys.

Module 3. Message Encryption Module (AES Module)

This module encrypts and decrypts chat messages.

Functions

Uses AES-256 to encrypt user messages.

Decrypts incoming messages using the shared AES key.

Ensures fast and secure real-time message encryption.

Module 4. Hashing & Message Chaining Module

This is the core security module inspired by blockchain.

Functions

Generates SHA-256 hash for each encrypted message.

Attaches previous message's hash to create a chain.

Detects tampering if any message in the chain is modified.

Module 5. Secure Transmission Module

This module ensures safe message delivery over the network.

Functions

Sends encrypted and chained messages over HTTPS/TLS.

Protects against eavesdropping, replay attacks, and MITM attacks.

Manages real-time message sending and receiving.

Module 6. Message Verification & Tamper Detection Module

This module checks every received message for tampering.

Functions

Recalculates hashes of received messages.

Compares computed hash with attached hash.

Detects:

Modified messages

Deleted messages

Inserted fake messages

Module 7. Storage Module

This module saves encrypted chat data securely.

Functions

Stores encrypted messages along with hashes in the database.

Prevents unauthorized reading of stored data.

Maintains a complete tamper-proof chat history.

Module 8. User Interface (UI) Module

This module provides an easy-to-use chat interface.

Functions

Allows users to send/receive messages instantly.

Displays encrypted message workflow.

Provides message verification alerts.

Supports secure logout and session management.

Purpose

To offer a smooth, responsive, and user-friendly chat experience.

4.2 Algorithm

The system uses selected cryptographic algorithms based on performance and security:

- **AES-256** → For message encryption
- **RSA/ECC** → For secure key exchange
- **SHA-256** → For message hashing
- **Hash-chaining technique** → For tamper-proof message linking

These algorithms provide industry-standard security and prevent unauthorized access.

4.3 SYSTEM IMPLEMENTATION

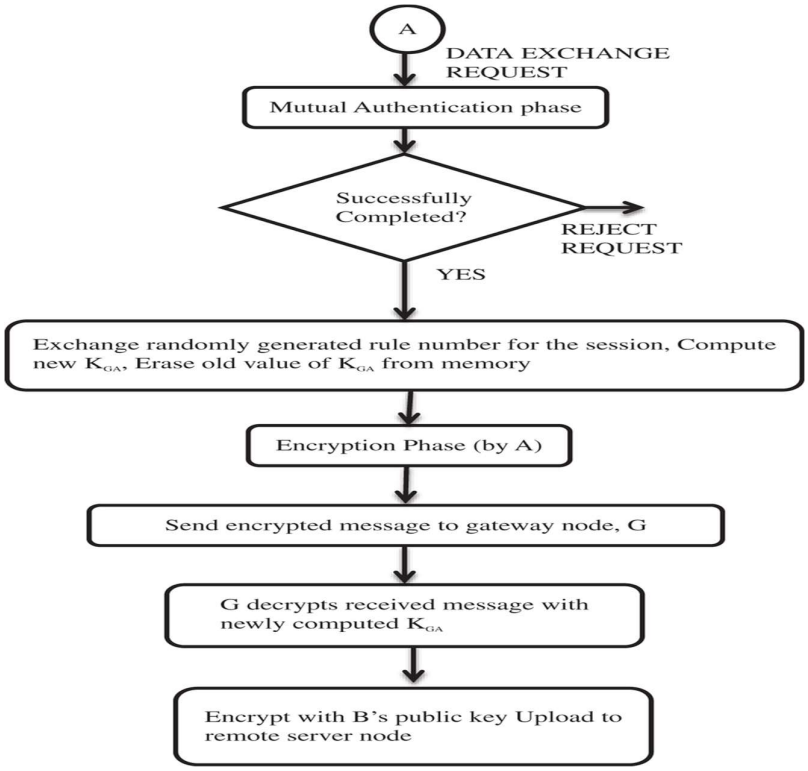
The implementation phase converts the design into working code.

Key implementation steps:

1. **User Authentication:**
    - Passwords stored as salted hashes
    - Secure login verification
2. **Key Exchange:**
    - User keys generated at signup
    - Session keys exchanged for encryption
3. **Message Processing:**
    - User types message → encrypted using AES
    - Hash is generated using SHA-256

4. **Message Transmission:**
    - Data sent over secure channels
    - Stored in encrypted form in database
5. **Receiving Messages:**
    - Receiver decrypts using AES key
    - Hash re-computed to check tampering

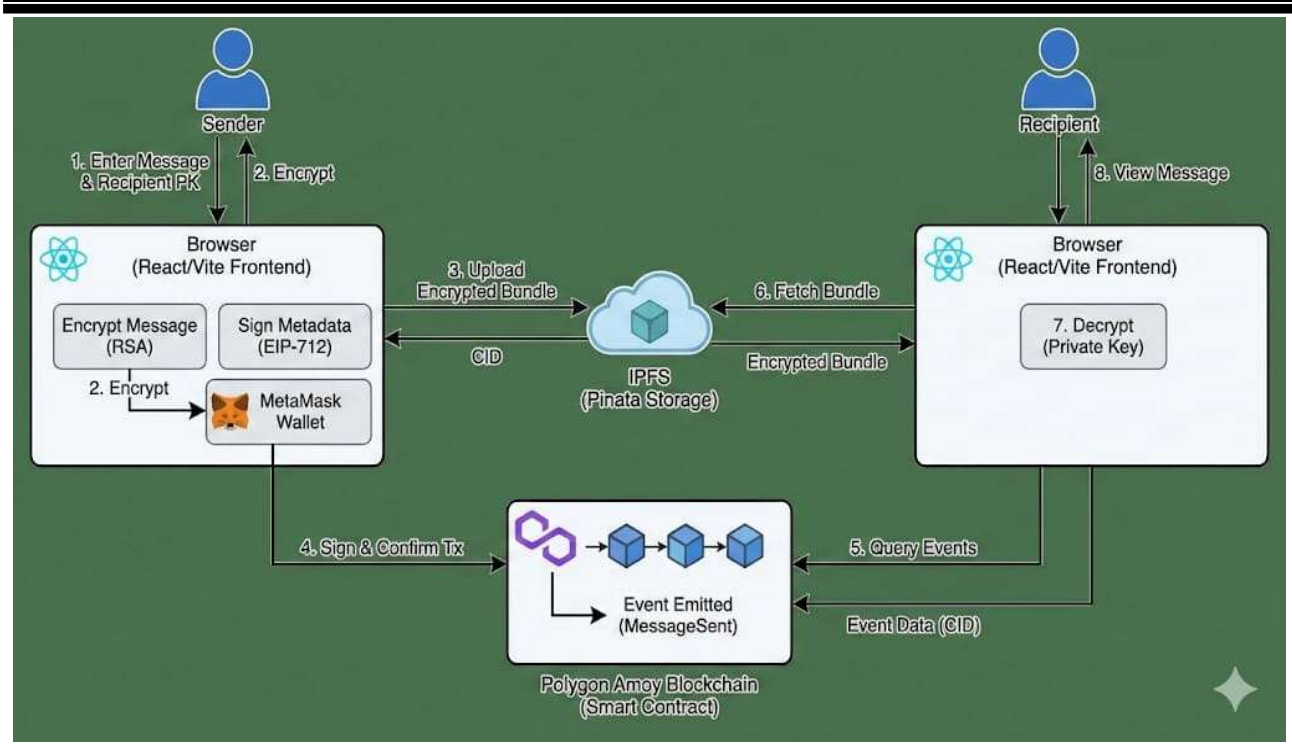
4.4 FLOWCHART



4.5 SYSTEM ARCHITECTURE

The System Architecture of the Chain Chat with Cryptography application is built to provide highly secure, encrypted, and tamper-proof communication. This architecture uses a combination of advanced cryptographic algorithms, blockchain-inspired hash chaining, and modular layered design to protect message confidentiality, authenticity, and integrity.

To achieve this, the architecture is divided into multiple layers, where each layer performs a specific security or functionality role. This layered modular architecture improves performance, scalability, maintainability, and overall system security.



4.6 SYSTEM DESIGN MODELS

4.6.1 Use Case Diagram

The Use Case Diagram of the Chain Chat with Cryptography system illustrates the functional requirements of the system from the user's point of view. It identifies all external actors that interact with the system and shows how they use different features of the application.

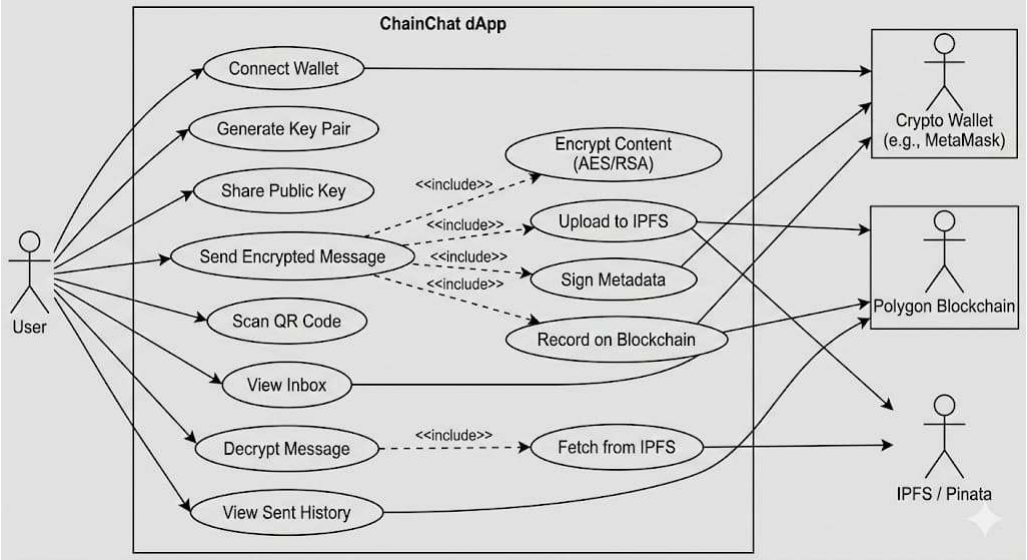
Use case diagrams help in understanding:

What the system should do

How users interact with the system

What functions are essential for secure communication

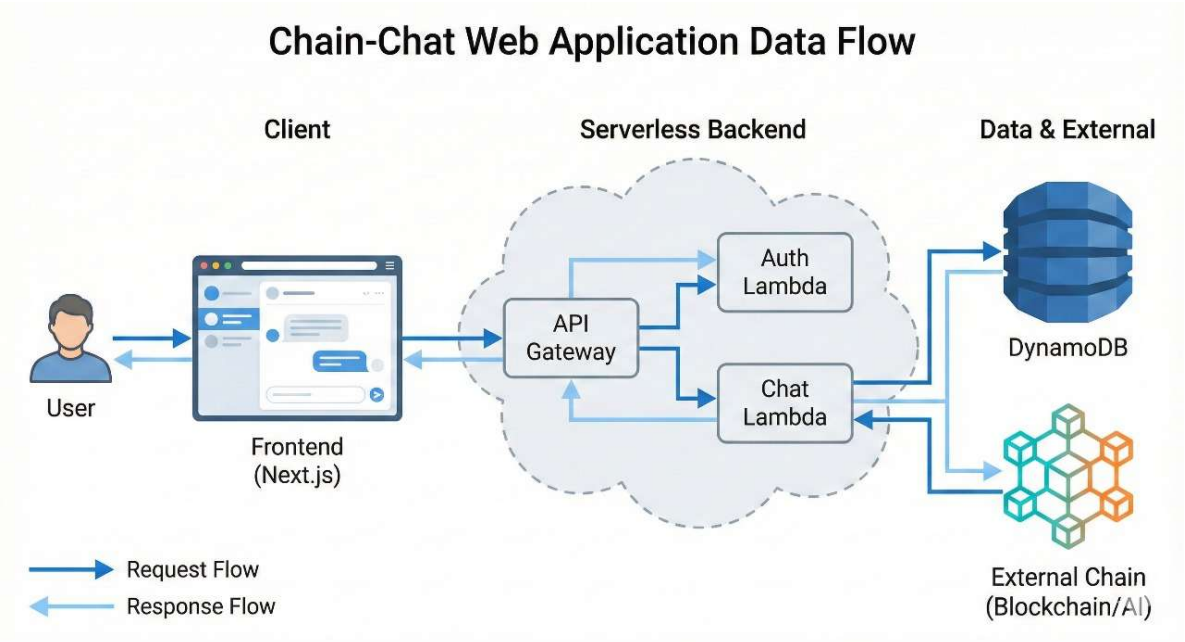
This diagram is typically created during the requirement analysis phase to clearly define the system boundaries and user interactions.



4.6.2 SEQUENCE DIAGRAM

The Sequence Diagram of the Chain Chat with Cryptography system represents the step-by-step interaction between different system components during the process of sending and receiving a secure message. It shows how the sender, receiver, cryptographic modules, hashing modules, and the server cooperate to ensure that every message remains encrypted, authenticated, and tamper-proof.

This diagram helps to visualize the flow of control, message passing, and chronological order in which various operations occur in the system.



Chapter 5

SYSTEM REQUIREMENTS

5.1 HARDWARE REQUIREMENTS

- **CPU:** At least a dual-core processor to handle chat and report generation efficiently.
- **RAM:** Minimum 8 GB; more (16 GB) recommended for smoother performance with blockchain data.
- **Storage:** SSD preferred (fast read/write) for storing logs, reports, and blockchain data.
- **Network:** Stable internet is needed to fetch on-chain data and support real-time chat.

Processor (CPU) Requirements

1. **Minimum Requirement:**
    - Suitable Dual-core CPU, 2.0 GHz or higher
    - for small-scale testing or personal projects. Can handle basic chat operations and simple report generation.
2. **Recommended Requirement:**
    - Quad-core CPU, 3.0 GHz or higher
    - Needed for smoother performance with multiple users, faster report generation, and processing blockchain data efficiently.
3. **Why CPU Matters:**
    - The processor handles all computations, including:
        - Running the chat application in real time.
        - Fetching and processing on-chain data.
        - Generating reports (PDF/Excel) quickly.
    - Faster CPUs reduce latency in chat and speed up report generation, especially for large datasets.
4. **Advanced Consideration:**
    - If integrating analytics or large-scale blockchain operations, a multi-core high-frequency CPU (6+ cores) may be beneficial.
    - For local blockchain nodes or heavy testing, CPU performance directly affects transaction processing and report calculation speed.

5.2 SOFTWARE REQUIREMENTS

- **Operating System:** Windows, Linux, or macOS. Must support the required runtime.
- **Runtime Environment:** Java 17 or Node.js 18+ depending on system implementation.
- **Database:** PostgreSQL, MySQL, or MongoDB to store user info, chat logs, and reports.
- **Web Browser:** Chrome, Firefox, or Edge for accessing the chat/report interface.

5.2.1 Dependencies

- Blockchain SDK (Web3.js / Web3j) to interact with the blockchain.
- Reporting libraries to generate PDF/Excel reports.
- Testing frameworks if automated testing is used.
- Docker if deploying in a containerized environment.

5.2.2 Network Requirements

- Internet connection to fetch blockchain data and support real-time chat.
- RPC endpoints to interact with the blockchain securely.

5.2.3 Security Requirements

- User authentication and role-based access to protect data.
- Encryption (TLS/SSL) for secure communication.
- Key management for blockchain transactions.
- Backup and recovery for reports and logs.

Chapter 6

IMPLEMENTATION AND RESULTS

6.1 IMPLEMENTATION

```
import React, { createContext, useContext, useState, useCallback, useEffect, ReactNode } from "react";
import { BrowserProvider, JsonRpcSigner, Contract } from "ethers";
import { CONTRACT_ADDRESS, CONTRACT_ABI, POLYGON_AMOY_CHAIN_ID,
POLYGON_AMOY_RPC } from "@/lib/constants";
import { toast } from "@/hooks/use-toast";
declare global {
  interface Window {
    ethereum?: {
      request: (args: { method: string; params?: unknown[] }) => Promise<unknown>;
      on: (event: string, handler: (...args: unknown[]) => void) => void;
      removeListener: (event: string, handler: (...args: unknown[]) => void) => void;
      isMetaMask?: boolean;
    };
  }
}
interface WalletContextType {
  account: string | null;
  provider: BrowserProvider | null;
  signer: JsonRpcSigner | null;
  contract: Contract | null;
  chainId: number | null;
  isConnecting: boolean;
  isCorrectNetwork: boolean;
  connect: () => Promise<void>;
  disconnect: () => void;
  switchNetwork: () => Promise<void>;
}
const WalletContext = createContext<WalletContextType | undefined>(undefined);
export function WalletProvider({ children }: { children: ReactNode }) {
  const [account, setAccount] = useState<string | null>(null);
```

```
const [provider, setProvider] = useState<BrowserProvider | null>(null);
const [signer, setSigner] = useState<JsonRpcSigner | null>(null);
const [contract, setContract] = useState<Contract | null>(null);
const [chainId, setChainId] = useState<number | null>(null);
const [isConnecting, setIsConnecting] = useState(false);
const isCorrectNetwork = chainId === POLYGON_AMOY_CHAIN_ID;
const switchNetwork = useCallback(async () => {
 if (!window.ethereum) return;
 try {
  await window.ethereum.request({
   method: "wallet_switchEthereumChain",
   params: [{ chainId: 0x${POLYGON_AMOY_CHAIN_ID.toString(16)} }],
  });
 } catch (error: unknown) {
  // If chain doesn't exist, add it
  if ((error as { code?: number })?.code === 4902) {
   await window.ethereum.request({
    method: "wallet_addEthereumChain",
    params: [
     {
      chainId: 0x${POLYGON_AMOY_CHAIN_ID.toString(16)},
      chainName: "Polygon Amoy Testnet",
      nativeCurrency: {
       name: "POL",
       symbol: "POL",
       decimals: 18,
      },
      rpcUrls: [POLYGON_AMOY_RPC],
      blockExplorerUrls: ["https://amoy.polygonscan.com"],
     },
    ],
   });
  }
 }
```

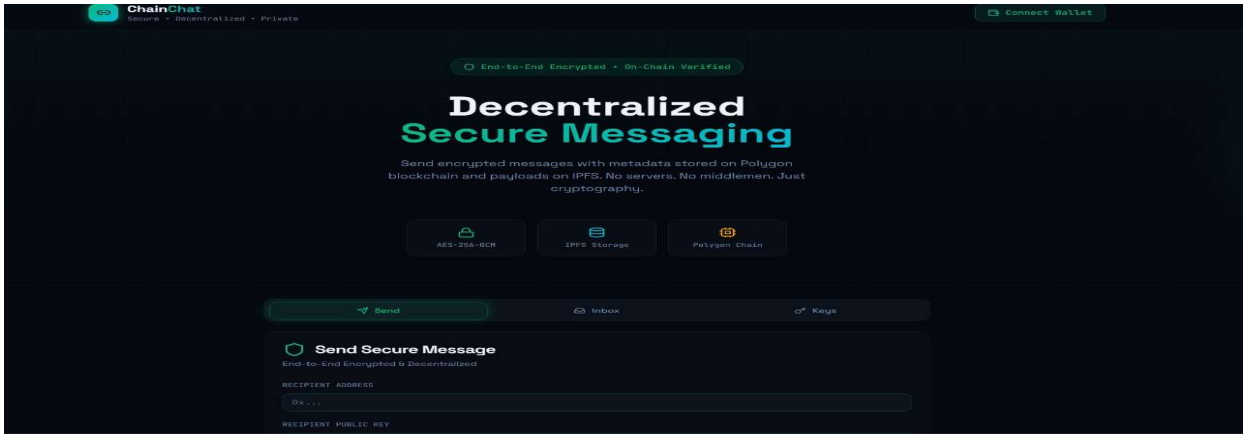
```
}, []);
const connect = useCallback(async () => {
 if (!window.ethereum) {
  toast({
   title: "MetaMask Required",
   description: "Please install MetaMask to use ChainChain",
   variant: "destructive",
  });
  return;
 }
 setIsConnecting(true);
 try {
  const accounts = await window.ethereum.request({
   method: "eth_requestAccounts",
  }) as string[];
  const browserProvider = new BrowserProvider(window.ethereum);
  const userSigner = await browserProvider.getSigner();
  const network = await browserProvider.getNetwork();
  const chainContract = new Contract(CONTRACT_ADDRESS, CONTRACT_ABI, userSigner);
  setAccount(accounts[0]);
  setProvider(browserProvider);
  setSigner(userSigner);
  setContract(chainContract);
  setChainId(Number(network.chainId));
  toast({
   title: "Wallet Connected",
   description: Connected to ${accounts[0].slice(0, 6)}...${accounts[0].slice(-4)},
  });
  if (Number(network.chainId) !== POLYGON_AMOY_CHAIN_ID) {
   toast({
    title: "Wrong Network",
    description: "Please switch to Polygon Amoy Testnet",
    variant: "destructive",
   });
```

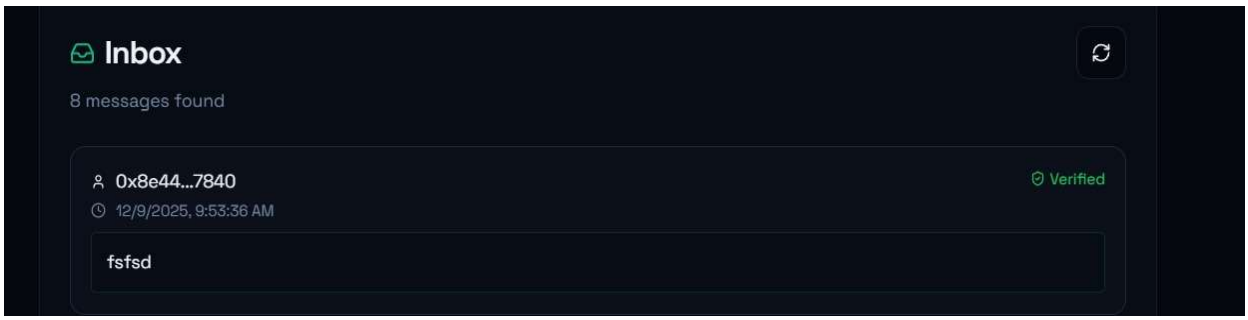
```
      }
    } catch (error) {
      console.error("Connection error:", error);
      toast({
        title: "Connection Failed",
        description: "Failed to connect wallet",
        variant: "destructive",
      });
    } finally {
      setIsConnecting(false);
    }
  }, []);
  const disconnect = useCallback(() => {
    setAccount(null);
    setProvider(null);
    setSigner(null);
    setContract(null);
    setChainId(null);
    toast({
      title: "Disconnected",
      description: "Wallet disconnected successfully",
    });
  }, []);
  // Listen for account and chain changes
  useEffect(() => {
    if (!window.ethereum) return;
    const handleAccountsChanged = (accounts: unknown) => {
      const accountList = accounts as string[];
      if (accountList.length === 0) {
        disconnect();
      } else if (accountList[0] !== account) {
        setAccount(accountList[0]);
      }
    };
```

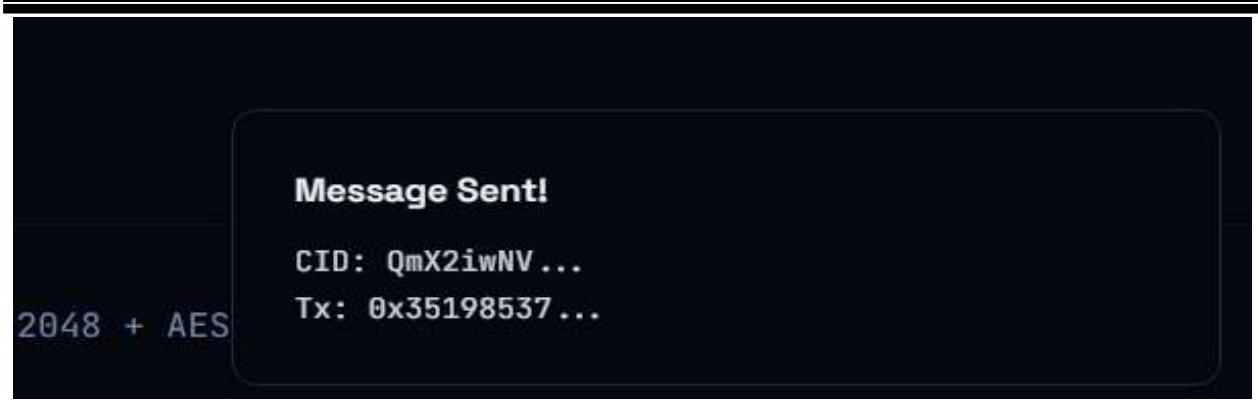
```
    const handleChainChanged = (newChainId: unknown) => {
      setChainId(parseInt(newChainId as string, 16));
    };
    window.ethereum.on("accountsChanged", handleAccountsChanged);
    window.ethereum.on("chainChanged", handleChainChanged);
    return () => {
      window.ethereum?.removeListener("accountsChanged", handleAccountsChanged);
      window.ethereum?.removeListener("chainChanged", handleChainChanged);
    };
  }, [account, disconnect]);
  return (
    <WalletContext.Provider
      value={{
        account,
        provider,
        signer,
        contract,
        chainId,
        isConnecting,
        isCorrectNetwork,
        connect,
        disconnect,
        switchNetwork,
      }}
    >
      {children}
    </WalletContext.Provider>
  );
}
export function useWallet() {
  const context = useContext(WalletContext);
  if (context === undefined) {
    throw new Error("useWallet must be used within a WalletProvider");
  }
```

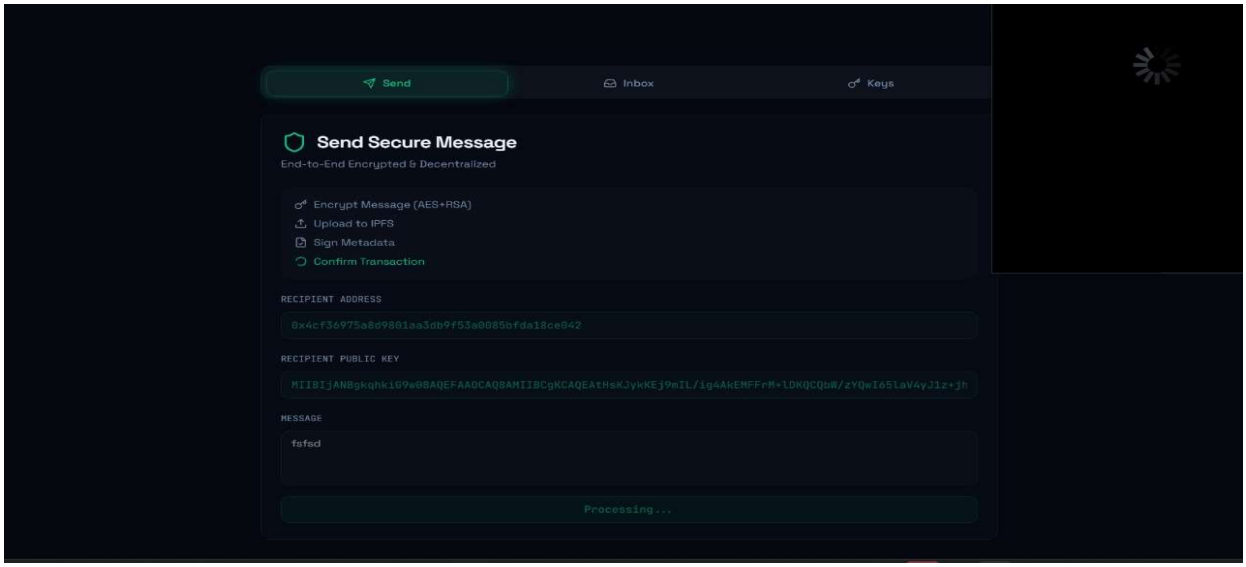
```
  return context;
}
```

6.2 RESULTS









CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

7.1 CONCLUSION

The Chain Chat with Cryptography system successfully demonstrates how modern cryptographic techniques and blockchain-inspired mechanisms can be integrated to build a highly secure and tamper-proof communication platform. The primary objective of this project was to develop a messaging system where data confidentiality, integrity, authentication, and non-repudiation are ensured at every stage of communication. By combining AES encryption, RSA/ECC key exchange, and SHA-256 hash chaining, the system provides a multi-layered security model capable of protecting messages from unauthorized access and malicious manipulation.

One of the major achievements of this project is the implementation of hash chaining, a concept taken from blockchain technology, which ensures that every message in the chat is linked to the previous one. This chaining mechanism creates a tamper-evident message history, where any alteration—even in a single message—breaks the chain and becomes immediately detectable. This feature significantly improves trust,

accountability, and data integrity in communication systems, especially in environments where message authenticity is crucial.

7.2 FUTURE ENHANCEMENT

Although the system is fully functional and secure in its current form, there are several enhancements that can improve performance, scalability, and user experience. These future improvements can extend the system into a more advanced and feature-rich secure communication platform.

1. Multi-User and Group Chat Support

Currently, the system supports one-to-one communication. Future versions can introduce:

Secure group chats

Broadcast messaging

Encrypted group key management

This would make the application suitable for organizations, teams, and enterprises.

2. File and Media Encryption

Future enhancements can include secure transfer of:

Images

Documents

Audio and video files

All files can be encrypted using AES and linked with hash chains to maintain tamper-proof history.

3. End-to-End Digital Signatures

Digital signatures can be added for:

Stronger user authentication

Preventing impersonation

Ensuring message origin verification

This will add an additional layer of security beyond encryption.

Chapter 8

REFERENCES

8.1 PAPER REFERENCES

1. William Stallings, Cryptography and Network Security: Principles and Practice, 7th Edition, Pearson Education, 2017.
2. Behrouz A. Forouzan, Cryptography and Network Security, McGraw-Hill, 2015.
3. Bruce Schneier, Applied Cryptography: Protocols, Algorithms and Source Code in C, John Wiley & Sons, 1996.
4. Diffie, W. & Hellman, M., "New Directions in Cryptography," IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644–654, 1976

8.2 WEB LINKS

- https://www.geeksforgeeks.org/ethical-hacking/cryptography-in-blockchain/
- https://ijcrt.org/papers/IJCRT2311267.pdf
- https://ieeexplore.ieee.org/document/10112278

8.3 YOUTUBE LINK

- https://www.youtube.com/watch?v=eh9Z7D3Uuco
- https://www.youtube.com/watch?v=Ta8uo7u8WeE
- https://www.youtube.com/watch?v=YqbhSIgimcg