Secure Chat Application - Project Analysis Report

Boureghda Mohamed Islam

2024/2025

Executive Summary

This report provides a comprehensive analysis of a Java-based Secure Chat Application project. The application implements a multi-user chat system with RSA encryption, modern GUI design, and secure authentication mechanisms. The project consists of 15 Java classes that work together to provide a complete secure messaging solution.

Project Overview

Architecture

The application follows a Model-View-Controller (MVC) architectural pattern with the following key components: - **Models**: User, Message, CourseModel, ParticipantListModel, MessageListModel - **Views**: LoginView, RegisterView, WelcomeView, ChatView, CourseView, ParticipantListView - **Utilities**: SecurityUtils, RSAUtils, ModernTheme - **Main Application**: MainApp

Key Features

- RSA encryption for secure messaging
- Multi-user authentication system
- Modern GUI with consistent theming
- Real-time chat functionality
- Course content viewing
- Participant management
- Input validation and security measures

Detailed Class Analysis

Core Application Classes

MainApp.java

Purpose: Entry point of the application **Key Functionality**: - Initializes shared models (ParticipantListModel, MessageListModel) - Calculates screen dimensions for window positioning - Creates two LoginView instances positioned on left and right sides of screen - Provides debug output for window positioning

Methods: - `main(String[] args)`: Application entry point

User.java

Purpose: Represents a user entity with encryption capabilities **Key Attributes**: - `id`: Unique user identifier - `pseudoName`: User's display name - `passwordHash`: SHA-256 hashed password - `privateKey`, `publicKey`: RSA key pair for encryption - `publicKeyString`: Serialized public key for sharing

Key Methods: - `User(String id, String pseudoName, String passwordHash)`: Constructor with input validation - `generateKeyPair()`: Generates RSA key pair using RSAUtils - `encryptMessage(String message, User recipient)`: Encrypts message using recipient's public key - `decryptMessage(String encryptedMessage)`: Decrypts message using user's private key - Getter methods for all attributes

Security Features: - Input validation using SecurityUtils - Automatic RSA key generation - Secure password hashing

View Classes

LoginView.java

Purpose: User authentication interface **Key Features**: - Modern UI with styled components - Input validation for username and password - SQL injection pattern detection - Dual window positioning support - Observer pattern implementation

Key Methods: - `LoginView(ParticipantListModel, MessageListModel, int h, int v)`: Constructor with positioning - `initializeUI()`: Sets up the user interface components - `performLogin(ActionEvent)`: Handles login authentication - `performRegister(ActionEvent)`: Opens registration window - `isLeftSide()`: Determines window position for consistent layout - `showError(String)`: Displays error messages

UI Components: - Username and password fields - Sign In, Create Account, and Exit buttons - Modern themed styling throughout

RegisterView.java

Purpose: New user registration interface **Key Features**: - User ID, username, and password input fields - Comprehensive input validation - Duplicate user checking - Automatic user creation and login

Key Methods: - `RegisterView(ParticipantListModel, MessageListModel, int h, int v)`: Constructor - `initComponents()`: UI initialization - `validateInputs(String id, String username, String password)`: Input validation - `showError(String)`: Error display

Validation Rules: - ID must be numeric - Username must be 1-20 alphanumeric characters - Password cannot contain SQL injection patterns - All fields are required

WelcomeView.java

Purpose: Main dashboard after successful login **Key Features**: - User welcome interface - Application launcher (Chat and Courses) - User information display - Consistent positioning based on login window location

Key Methods: - `WelcomeView(User, ParticipantListModel, MessageListModel, int h, int v, boolean isLeftSide)`: Constructor - `initializeUI()`: Dashboard setup - `createAppCard(String title, String description, Color color)`: Creates application cards - `openChat(ActionEvent)`: Launches chat application - `openCourses(ActionEvent)`: Launches course viewer - `performLogout()`: Returns to login screen

UI Features: - Welcome message with user information - Interactive application cards with hover effects - Logout functionality - Modern card-based design

ChatView.java

Purpose: Real-time encrypted messaging interface **Key Features**: - Real-time message display with encryption status - Participant selection - Message encryption/decryption - Custom message rendering - Observer pattern for real-time updates

Key Methods: - `ChatView(MessageListModel, User, ParticipantListModel, int h, int v)`: Constructor - `initComponents()`: Chat interface setup - `updateParticipantList(Vector<User>)`: Updates available participants - `update(Observable, Object)`: Handles real-time message updates

UI Components: - Message display area with custom cell renderer - Participant selection dropdown - Message input field - Send and Clear buttons - Encryption status indicators

Message Features: - Custom MessageCellRenderer for chat bubbles - Encryption/decryption status display - Time stamps - Sender/receiver identification - Different styling for sent/received messages

CourseView.java

Purpose: Course content viewing interface **Key Features**: - Course content display - Scrollable text area - Course information header - Observer pattern for content updates

Key Methods: - `CourseView(CourseModel, int h, int v)`: Constructor - `initComponents()`: UI setup - `update(Observable, Object)`: Content update handling

ParticipantListView.java

Purpose: User management and participant viewing **Key Features**: - List of all registered participants - Custom participant rendering with avatars - User information display - Modern list styling

Key Methods: - `ParticipantListView(ParticipantListModel, int x, int y)`: Constructor - `initComponents()`: List interface setup - `updateList()`: Refreshes participant list - `update(Observable, Object)`: Handles participant updates

Custom Rendering: - ParticipantCellRenderer with avatar circles - User initials in avatars - User ID and name display - Alternating row colors

Model Classes

ParticipantListModel.java

Purpose: Manages user data and authentication **Key Features**: - User storage and management - Authentication logic - Duplicate prevention - Pre-loaded test users

Key Methods: - `ParticipantListModel()`: Constructor with dummy users - `registerParticipant(String id, String username, String password)`: User registration - `authenticate(String pseudo, String password)`: User authentication - `validateUnique(String id, String username)`: Duplicate checking - `getParticipants()`: Returns all users

Pre-loaded Users: - admin/admin (ID: 1) - test/test (ID: 3) - user1/user1 (ID: 2)

MessageListModel.java

Purpose: Manages encrypted messaging system **Key Features**: - Message storage and retrieval - User registration for encryption - Automatic encryption/decryption - Message filtering by user

Key Methods: - `registerUser(User)`: Registers user for encryption - `sendMessage(String sender, String receiver, String content)`: Sends encrypted message - `decryptMessage(Message, String userPseudoName)`: Decrypts message for user - `getMessagesForUser(String userPseudoName)`: Retrieves user-specific messages - `getMessages()`: Returns all messages

Encryption Features: - Automatic message encryption using recipient's public key - Automatic decryption for intended recipients - Fallback for missing encryption keys - Debug logging for encryption operations

Message.java

Purpose: Represents individual chat messages **Key Attributes**: - `messageId`: Unique message identifier - `sender`: Message sender username - `receiver`: Message recipient username - `encryptedContent`: Encrypted message content - `decryptedContent`: Decrypted content (if available) - `isDecrypted`: Decryption status flag

Key Methods: - `Message(String sender, String receiver, String encryptedContent)`: Constructor - `setDecryptedContent(String)`: Sets decrypted content - `getContent()`: Returns appropriate content based on decryption status - `isDecrypted()`: Checks decryption status

CourseModel.java

Purpose: Represents course content **Key Attributes**: - `CourseId`: Course identifier - `CoursePath`: Path to course materials - `content`: Course content text

Key Methods: - `CourseModel(int CourseId, String CoursePath, String content)`: Constructor - Getter and setter methods for all attributes

Utility Classes

SecurityUtils.java

Purpose: Security and validation utilities **Key Features**: - SHA-256 password hashing - Input validation - SQL injection pattern detection

Key Methods: - `hashSHA256(String input)`: Generates SHA-256 hash - `bytesToHex(byte[] bytes)`: Converts bytes to hexadecimal - `isValidInput(String input)`: Validates alphanumeric input (1-20 chars) - `containsSQLPatterns(String input)`: Detects SQL injection patterns

Security Patterns Detected: - SQL keywords (DROP, DELETE, INSERT, etc.) - Special characters (semicolons, quotes, comments) - SQL comment patterns

RSAUtils.java

Purpose: RSA encryption/decryption utilities **Key Features**: - RSA key pair generation - Message encryption/decryption - Key serialization/deserialization

Key Methods: - `generateKeyPair()`: Generates 2048-bit RSA key pair - `encrypt(String message, PublicKey publicKey)`: Encrypts message - `decrypt(String encryptedMessage, PrivateKey privateKey)`: Decrypts message - `publicKeyToString(PublicKey)`: Serializes public key to Base64 - `privateKeyToString(PrivateKey)`: Serializes private key to Base64 - `stringToPublicKey(String)`: Deserializes public key from Base64 - `stringToPrivateKey(String)`: Deserializes private key from Base64

Configuration: - Algorithm: RSA - Key size: 2048 bits - Encoding: Base64

ModernTheme.java

Purpose: Consistent UI styling and theming **Key Features**: - Color scheme definition - Font standardization - Component styling methods - Modern UI appearance

Color Scheme: - Primary: Blue (#4285F4) - Secondary: Green (#34A853) - Accent: Red (#EA4335) - Background: Light Gray (#F8F9FA) - Text: Dark Gray (#3C4043)

Font Definitions: - Title Font: Segoe UI Bold 18pt - Header Font: Segoe UI Bold 14pt - Regular Font: Segoe UI Plain 13pt - Small Font: Segoe UI Plain 11pt

Styling Methods: - `styleButton(JButton, boolean isPrimary)`: Button styling with hover effects - `styleTextField(JTextField)`: Text field styling -

`stylePasswordField(JPasswordField)`: Password field styling - `styleLabel(JLabel)`: Label styling - `stylePanel(JPanel)`: Panel styling - `styleFrame(JFrame)`: Frame styling - `styleComboBox(JComboBox)`: Combo box styling - `styleList(JList)`: List styling - `createStyledScrollPane(Component)`: Scroll pane creation - `createHeaderPanel(String)`: Header panel creation

Security Analysis

Encryption Implementation

- **Algorithm**: RSA with 2048-bit keys
- **Key Management**: Automatic key generation per user
- **Message Security**: All messages encrypted with recipient's public key
- **Key Storage**: In-memory storage (not persistent)

Authentication Security

- **Password Hashing**: SHA-256 algorithm
- **Input Validation**: Alphanumeric restrictions
- **SQL Injection Protection**: Pattern detection and blocking
- **Session Management**: Basic user session handling

Security Strengths

- Strong RSA encryption for messages
- Password hashing for storage
- Input validation and sanitization
- SQL injection prevention

Technical Architecture

Design Patterns Used

1. **Observer Pattern**: Used in models for real-time updates
2. **MVC Pattern**: Clear separation of concerns
3. **Singleton-like**: Shared model instances
4. **Factory Pattern**: Component creation in ModernTheme

Data Flow

1. User authentication through LoginView
2. User registration via RegisterView
3. Dashboard access through WelcomeView
4. Chat functionality via ChatView with encrypted messaging
5. Course content access through CourseView
6. Participant management via ParticipantListView

Conclusion

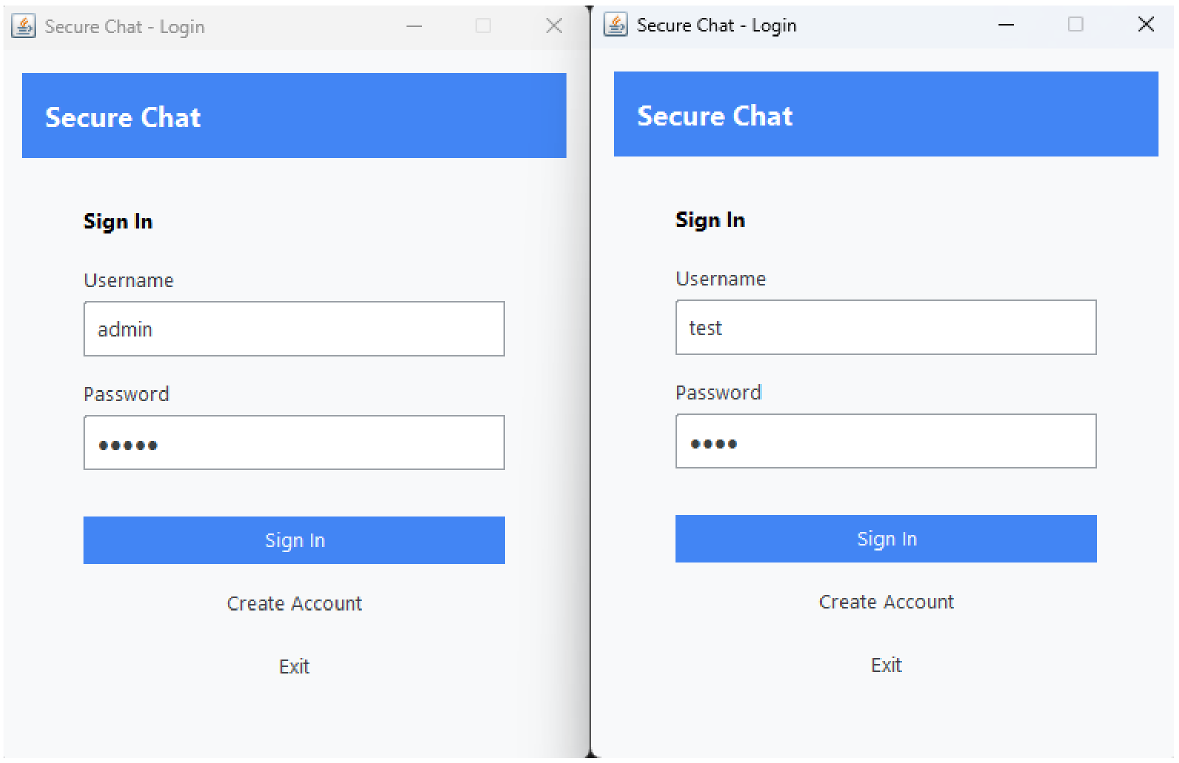
This Secure Chat Application demonstrates a well-structured Java application with strong emphasis on security and user experience. The implementation includes modern encryption techniques, comprehensive input validation, and a polished user interface. The modular design allows for easy maintenance and future enhancements.

Strengths

- Comprehensive security implementation
- Modern, consistent UI design
- Well-organized code structure
- Real-time messaging capabilities
- Robust input validation

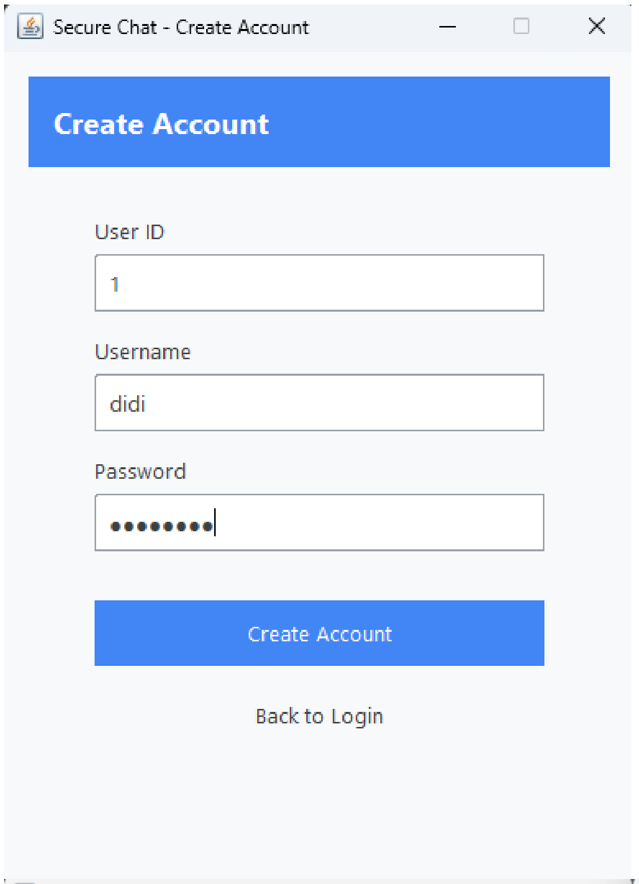
The project successfully demonstrates secure software development principles while maintaining usability and modern design standards.

Screenshots



Username: admin
Password: •••••

Username: test
Password: ••••



Secure Chat - Create Account

Create Account

User ID

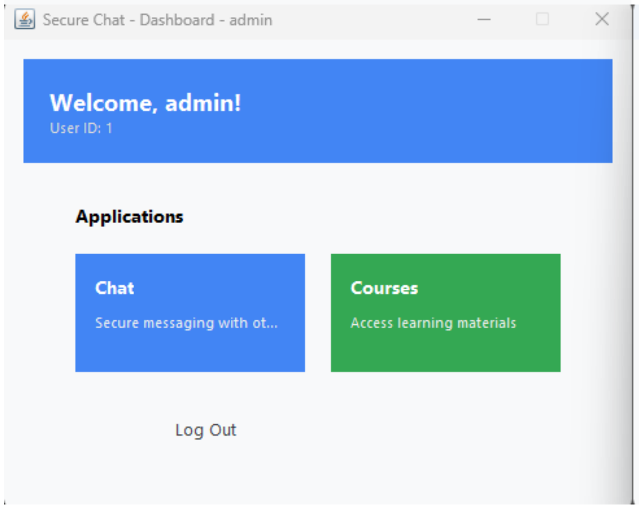
1

Username

didi

Password

••••••••



Secure Chat - Dashboard - admin

Welcome, admin!

User ID: 1

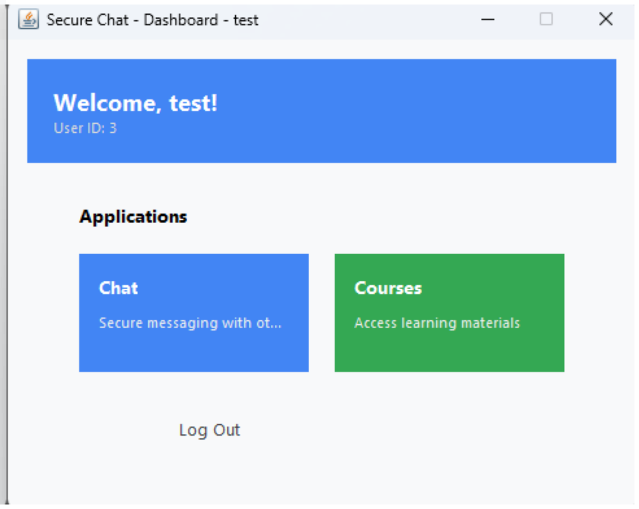
Applications

Chat

Secure messaging with ot...

Courses

Access learning materials



Secure Chat - Dashboard - test

Welcome, test!

User ID: 3

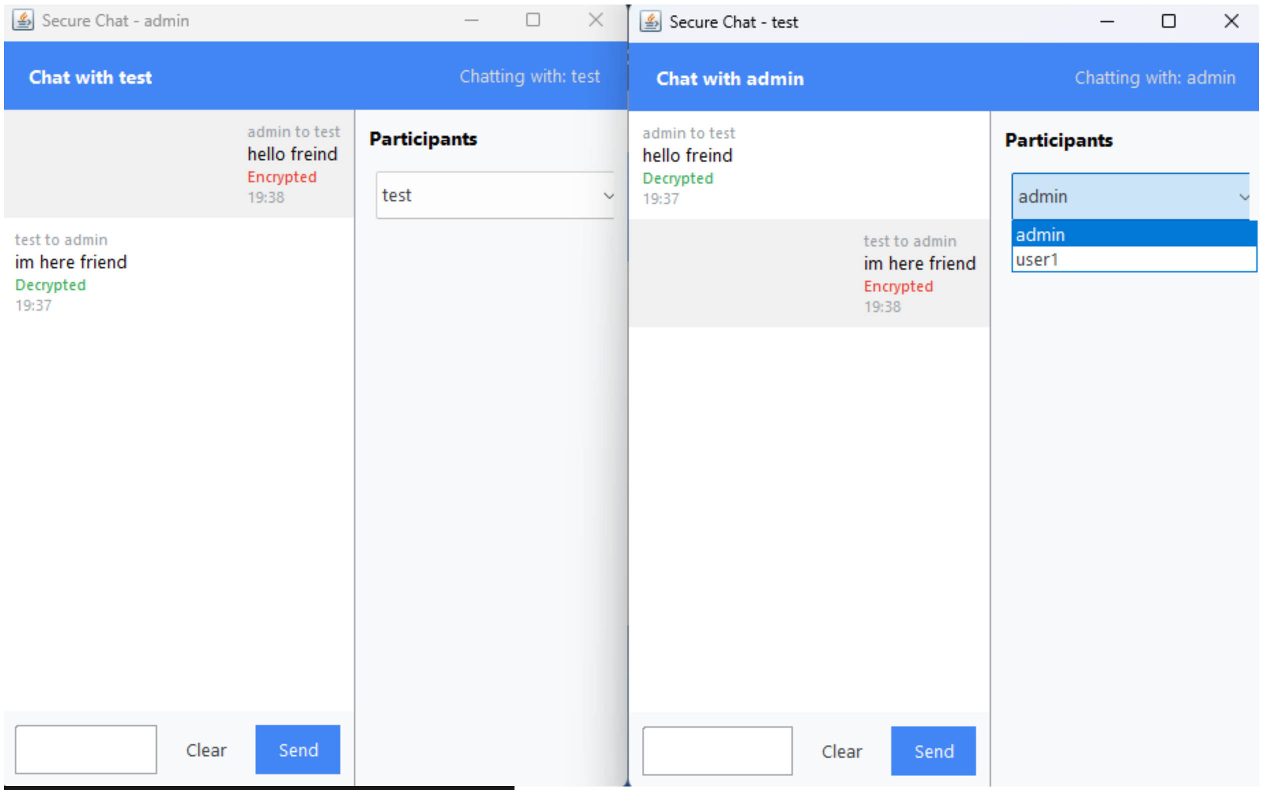
Applications

Chat

Secure messaging with ot...

Courses

Access learning materials


Secure Chat - admin

Chat with test

Chatting with: test

admin to test
hello freind
Encrypted
19:38

test to admin
im here friend
Decrypted
19:37

Participants

test

Clear Send

Secure Chat - test

Chat with admin

Chatting with: admin

admin to test
hello freind
Decrypted
19:37

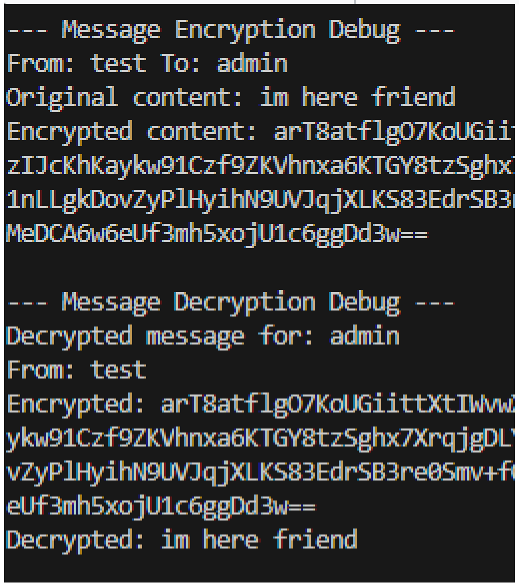
test to admin
im here friend
Encrypted
19:38

Participants

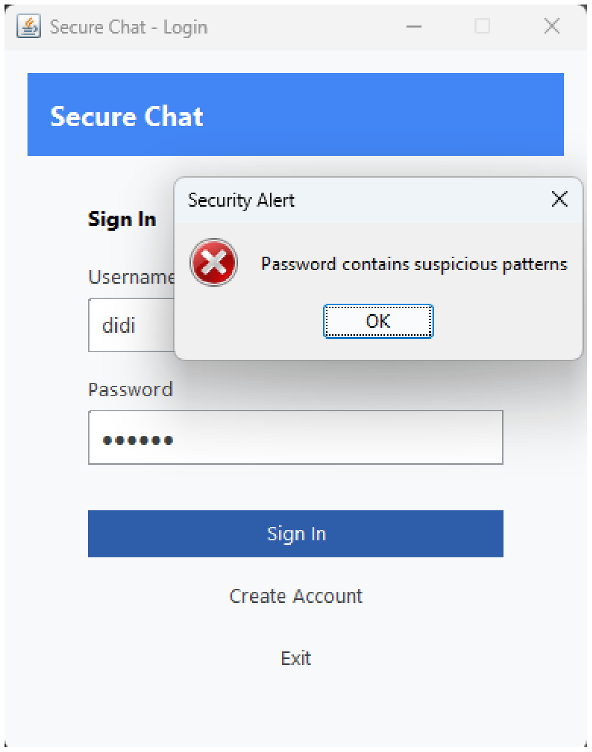
admin

admin
user1

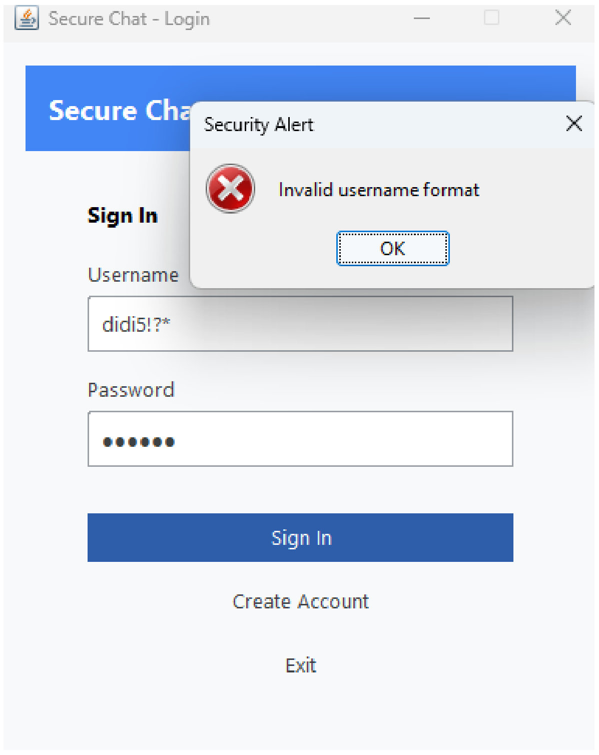
Clear Send


```
--- Message Encryption Debug ---
From: test To: admin
Original content: im here friend
Encrypted content: arT8atflg07KoUGii
zIJcKhKaykw91Czf9ZKVhnxa6KTGY8tzSghx
1nLLgkDovZyPlHyihN9UVJqjXLKS83EdrSB3
MeDCA6w6eUf3mh5xojU1c6ggDd3w==

--- Message Decryption Debug ---
Decrypted message for: admin
From: test
Encrypted: arT8atflg07KoUGiittXtIWvw
ykw91Czf9ZKVhnxa6KTGY8tzSghx7XrqjgDL
vZyPlHyihN9UVJqjXLKS83EdrSB3re0Smv+f
eUf3mh5xojU1c6ggDd3w==
Decrypted: im here friend
```



Password contains suspicious patterns



Invalid username format