

Information Security Project

Implementation of Zero Trust Architecture on SecureChat (Cybersecurity Chatbot)



Session: 2022 – 2026

Submitted by:

Saad Ahmad Malik	2022-CS-1
Waleed Ahmed	2022-CS-41

Submitted to:

Mr. Waqas Ali

Department of Computer Science
University of Engineering and Technology
Lahore Pakistan

Abstract

This report presents the design and implementation of SecureChat, a Flask-based web application that integrates Zero Trust Architecture (ZTA) to secure an AI-driven chatbot focused on Information Security, Networking, and Ethical Hacking. The project addresses the need for robust security in AI systems by implementing features such as SMTP-based email verification, OTP authentication, role-based access control, custom RSA and AES encryption, session management, password hashing, and defenses against cross-site scripting (XSS) and injection attacks. The report details the technology stack, customized encryption models, ZTA principles, and concluding with a description of the project's functionality.

Contents

1 Introduction.....	4
1.1 The Role of Information Security	4
1.2 Overview of SecureChat	4
1.3 Motivation for Zero Trust Architecture.....	4
2 Zero Trust Architecture: Principles, Components, and Relevance to AI	5
2.1 Core Principles of Zero Trust Architecture	5
2.2 Key Components of Zero Trust Architecture.....	5
2.3 Need for Zero Trust in AI Systems	6
3 Project Architecture	6
3.1 Technology Stack	6
3.2 Models Used	7
3.3 Customized Models	7
3.4 Frontend Components	7
4 Implementation of Zero Trust Architecture and Security	8
5 Project Working	9
6 Project Links	12

1 Introduction

1.1 The Role of Information Security

Information security has become a cornerstone of modern digital systems, protecting sensitive data from unauthorized access, breaches, and cyber threats. It has enabled secure communication, safeguarded user privacy, and fostered trust in online platforms, particularly in fields like cybersecurity, networking, and ethical hacking, where sensitive knowledge is shared. SecureChat leverages these principles to create a safe environment for users to engage with an AI chatbot focused on these domains.

1.2 Overview of SecureChat

SecureChat is a Flask-based web application designed to provide a secure, real-time communication platform for enthusiasts, professionals, and learners in Information Security, Networking, and Ethical Hacking. The application integrates a Groq-based AI chatbot, restricted to these topics, and uses MongoDB for data storage.

Key features include:

- User authentication with SMTP-based email verification and OTP.
- Role-based access control (admin and user roles).
- Secure chat functionality with message encryption using custom AES.
- Sensitive data encryption (e.g., OTPs) using custom RSA.
- Password hashing using Werkzeug's `generate_password_hash`.
- Session management for secure user sessions.
- Admin controls for user management (edit roles, delete accounts).
- Profile management for users to update their details.
- Frontend validation and sanitization to prevent XSS and injection attacks.
- Responsive UI with interactive dashboards, chat interfaces, and forms.

All features are fortified with Zero Trust Architecture to ensure robust security.

1.3 Motivation for Zero Trust Architecture

The decision to implement Zero Trust Architecture in SecureChat was driven by the increasing vulnerabilities in AI systems, such as data breaches, unauthorized access to sensitive user interactions, and the potential for AI model manipulation. Traditional security models assume trust within the network, which is insufficient for AI-driven applications handling sensitive data. ZTA's "never trust, always verify" approach ensures that every user, request, and data transaction is authenticated, authorized, and encrypted, making it ideal for protecting SecureChat's AI model and user data.

2 Zero Trust Architecture: Principles, Components, and Relevance to AI

Zero Trust Architecture (ZTA) is a security paradigm that eliminates the concept of implicit trust within a network, operating on the principle of “never trust, always verify.” Unlike traditional perimeter-based security models that assume internal entities are safe, ZTA assumes that threats can originate from both inside and outside the network, requiring continuous verification of every user, device, and transaction.

2.1 Core Principles of Zero Trust Architecture

ZTA is built on the following foundational principles:

- **Verify Explicitly:** Every access request must be authenticated and authorized using multiple factors, such as identity, device health, and context (e.g., location, time). In SecureChat, this is achieved through SMTP-based email verification, OTP authentication, and session validation for user login.
- **Least Privilege Access:** Users and systems are granted the minimum level of access necessary to perform their tasks, reducing the attack surface. SecureChat implements role-based access control, ensuring that only admins can access user management features.
- **Assume Breach:** ZTA operates under the assumption that a breach has already occurred, emphasizing micro-segmentation, encryption, and monitoring to limit damage. SecureChat’s use of custom AES and RSA encryption ensures that even if the database is compromised, the data remains unreadable.
- **Continuous Monitoring:** Real-time monitoring and logging of all activities help detect and respond to anomalies. While SecureChat does not yet fully implement this, it could be enhanced with logging of login attempts and chat interactions.

2.2 Key Components of Zero Trust Architecture

ZTA relies on several components to enforce its principles:

- **Identity and Access Management (IAM):** Strong authentication mechanisms, such as multi-factor authentication (MFA), ensure that only verified users gain access. SecureChat uses OTP-based MFA alongside email verification and session management.
- **Data Encryption:** Data must be encrypted both at rest and in transit to prevent unauthorized access. SecureChat employs custom AES for chat messages and custom RSA for sensitive fields like OTPs.
- **Micro-Segmentation:** Dividing the system into smaller segments limits lateral movement by attackers. In SecureChat, this is achieved through role-based dashboards and route restrictions (e.g., admin-only access to /users).

- **Security Policies and Analytics:** Dynamic policies based on user behavior and context, coupled with analytics, help detect threats. SecureChat could benefit from implementing behavioral analytics to monitor user activity.
- **Endpoint Security:** Ensuring that devices accessing the system are secure and compliant. SecureChat currently lacks device verification but could integrate checks for device health in the future.

2.3 Need for Zero Trust in AI Systems

AI systems like SecureChat face unique security challenges that necessitate ZTA:

- **Sensitive Data Exposure:** AI models process sensitive user data (e.g., chat messages in SecureChat), making them prime targets for data breaches. ZTA's encryption and access controls mitigate this risk.
- **Model Integrity Threats:** Adversaries may attempt to manipulate AI models through data poisoning or unauthorized access. ZTA's strict IAM ensures that only authorized users can interact with the system.
- **Evolving Threat Landscape:** AI systems are vulnerable to advanced attacks like model inversion and adversarial inputs. ZTA's continuous monitoring and assume breach mindset help detect and respond to such threats.
- **Regulatory Compliance:** AI applications often need to comply with data protection regulations (e.g., GDPR). ZTA's focus on encryption and least privilege helps meet these requirements.

In SecureChat, ZTA ensures that the Groq-based AI chatbot and MongoDB database are protected against unauthorized access, data breaches, and other cyber threats, aligning with the project's goal of providing a secure platform for cybersecurity discussions.

3 Project Architecture

3.1 Technology Stack

SecureChat is built using the following technologies:

- **Flask:** A Python web framework for building the application's backend, handling routes, and rendering templates.
- **MongoDB:** A NoSQL database for storing user data and chat history, accessed via Flask-PyMongo.
- **Groq API:** Provides the AI chatbot functionality, restricted to Information Security, Networking, and Ethical Hacking topics.
- **HTML/CSS/JavaScript:** Frontend technologies for creating a responsive and interactive user interface, including dashboards, chat interfaces, and forms.

- **Flask-Mail:** Facilitates SMTP-based email verification for OTP authentication using Gmail SMTP.
- **Werkzeug:** Provides password hashing (generate_password_hash, check_password_hash) for secure password storage.
- **Flask-Cors:** Enables Cross-Origin Resource Sharing for secure API interactions.
- **Python-Dotenv:** Manages environment variables (e.g., API keys, MongoDB URI) securely.
- **Gunicorn:** Used as a WSGI server for deploying the application in production.

3.2 Models Used

The project integrates the following models:

- **Groq AI Model:** The “meta-llama/llama-4-maverick-17b-128e-instruct” model powers the chatbot, with a system prompt restricting responses to cybersecurity topics.
- **MongoDB Collections:** Two collections are used: users for user data (username, email, hashed password, role, OTP, verification status) and Chathistory for storing user-AI interactions (user ID, encrypted messages, timestamp).

3.3 Customized Models

SecureChat implements custom encryption algorithms to enhance security:

- **Custom AES:** A symmetric encryption algorithm for encrypting chat messages in the Chathistory collection, ensuring data confidentiality at rest.
- **Custom RSA:** An asymmetric encryption algorithm for encrypting sensitive fields like OTPs in the users collection, facilitating secure key exchange and data protection.

3.4 Frontend Components

The frontend, built with HTML, CSS, and JavaScript, includes:

- **User and Admin Dashboards:** Display platform features and provide navigation to chat, profile, and logout functionalities.
- **Secure Chat Interface:** A responsive chat UI with message history, user input, and bot responses, styled with animations and icons.
- **Profile Management:** Allows users to view and edit their details, with client-side validation for password updates.
- **Admin User Management:** A dedicated interface for admins to manage users, including search, filter, and edit/delete options.
- **Forms:** Login, signup, OTP verification, password reset, and profile update forms, with client-side validation (e.g., password matching, visibility toggling)

4 Implementation of Zero Trust Architecture and Security

SecureChat implements ZTA through a comprehensive set of security mechanisms:

- **SMTP-Based Email Verification and OTP Authentication:** Users must verify their email via a 6-digit OTP sent through Flask-Mail (using Gmail SMTP). Only verified users can log in, enforcing strict identity verification.
- **Role-Based Authentication:** Users are assigned roles (admin or user), with admins having access to user management features (/users route). This ensures least privilege access, a core ZTA principle.
- **Custom RSA and AES Encryption:** Chat messages are encrypted with custom AES, and sensitive data like OTPs are encrypted with custom RSA before storage in MongoDB, ensuring data protection at rest.
- **Password Hashing:** Passwords are hashed using Werkzeug's `generate_password_hash` before storage, protecting them from unauthorized access even if the database is compromised.
- **Session Management:** Flask's session management ensures secure user sessions, with checks for logged-in status and role-based redirection (e.g., `@login_required` decorator).
- **Cross-Site Scripting (XSS) Prevention:** Input validation and sanitization are applied to user inputs (e.g., chat messages, form fields). Flask's template engine escapes HTML characters, and JavaScript functions (e.g., `validatePassword` in `script.js`) ensure safe input handling, preventing script injection in URLs or inputs.
- **Injection Attack Prevention:** Since SecureChat uses MongoDB, SQL injection is not applicable. However, MongoDB queries are parameterized using FlaskPyMongo to prevent NoSQL injection attacks (e.g., avoiding raw query execution with user input).
- **Email Domain Whitelisting:** During signup, only trusted email domains (Gmail, Yahoo, Hotmail) are allowed, reducing the risk of fake accounts.
- **Strong Password Requirements:** The signup and password reset forms enforce strict password criteria (e.g., minimum length, special characters), displayed to users via `password-requirements` in `signup.html` and `reset.html`.
- **Client-Side Enhancements:** JavaScript (`script.js`, `chat.js`) provides password visibility toggling, form switching, and real-time chat functionality, with validation to prevent malicious input before it reaches the server.

These features achieve ZTA by ensuring that:

- Every user is authenticated and verified (via OTP, email, and session checks).
- Access is restricted based on roles (admin vs. user).
- Data is encrypted at rest (using custom AES and RSA) and protected through hashed passwords.
- Threats like XSS and injection attacks are mitigated through input validation, sanitization, and secure query practices.


5 Project Working

SecureChat operates as follows:


1. **User Registration and Verification:** Users sign up with a username, email, and password, adhering to strong password requirements. An OTP is sent via email (using Flask-Mail), which they must verify to activate their account.

Sign Up



Username

 Mr_Saad

Email



 ecosprit@gmail.com

Password

- At least 8 characters long
- At least one uppercase letter
- At least one lowercase letter
- At least one digit
- At least one special character (e.g., !@#%&*)

Confirm Password


[Register](#)

[Already have an account? Login](#)

Verify Email

Invalid OTP. Please try again.

Enter OTP


 574337

[Verify](#)



Login

Email verified successfully! You can now login.

Username or Email

 Mr_Saad

Password

[Login](#)

[Forgot Password?](#)

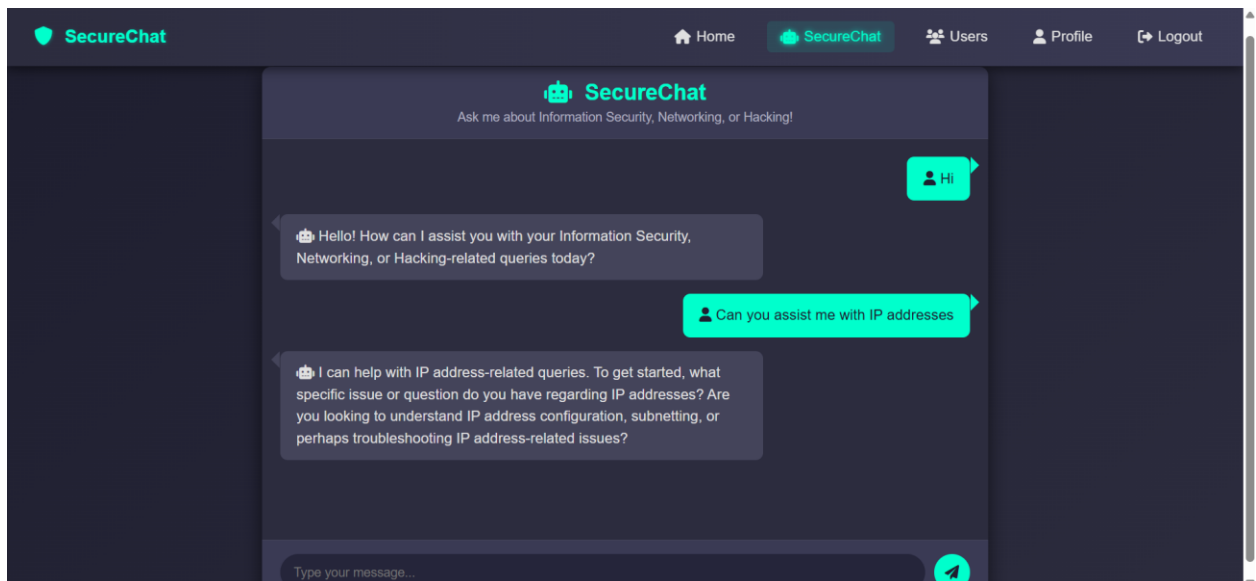
[Don't have an account? Sign Up](#)

2. **Login and Role-Based Access:** Verified users log in with their credentials. Passwords are checked against hashed values, and sessions are created. Based on their role, users are directed to either the user dashboard or admin panel.

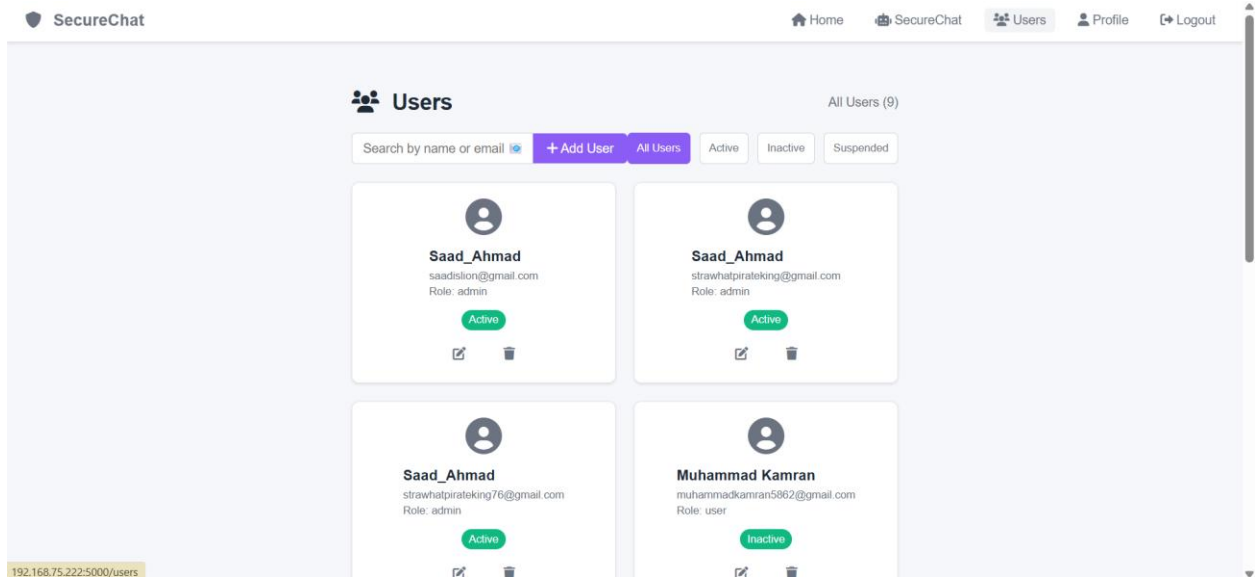
```
@app.route('/dashboard')
def dashboard():
    if 'user_id' not in session:
        return redirect(url_for('login'))

    role = session.get('role')
    if role == 'admin':
        return render_template('dashboard/admin.html')
    elif role == 'user':
        return render_template('dashboard/user.html')
    else:
        return "Unauthorized", 403
```

3. **Secure Chat Interaction:** Users interact with the Groq-based AI chatbot via the /secure_chat route. Messages are encrypted with custom AES before storage in MongoDB and decrypted upon retrieval. Chat history is fetched and displayed with user and bot icons.



4. **Admin Features:** Admins can manage users (edit roles, delete accounts) via the /users route, with features like search, filter, and a modal for editing user details.



5. **Profile Management:** Users can update their profile details, including username, email, and password, with all sensitive data (e.g., updated passwords) encrypted using custom RSA.

The image shows two versions of the 'Admin Profile' form. The left version displays the current profile information:

- Username:** Saad_Ahmad
- Email:** saadislion@gmail.com
- Role:** admin
- Verified:** Yes

Below this information is a red 'Edit Profile' button. The right version shows the form with input fields for updating the profile:

- Username:** Input field containing 'Saad_Ahmad'
- Email:** Input field containing 'saadislion@gmail.com'
- New Password (leave blank to keep current):** Input field with placeholder 'Enter new password'
- Confirm New Password:** Input field with placeholder 'Confirm new password'

At the bottom of the right form are two buttons: a red 'Save Changes' button and a blue 'Cancel' button.

6. **Password Reset:** Users can reset their password (forgot.html, reset.html) by requesting an OTP via email, ensuring secure recovery.

The image displays two side-by-side screenshots of a web application interface for password recovery.

Left Screenshot: Forgot Password

- Title: **Forgot Password**
- Label: Email
- Input field: ecosprit@gmail.com
- Button: **Send OTP**
- Link: Remembered? [Login](#)

Right Screenshot: Verify Email

- Message: OTP sent to your email. Please verify to reset password.
- Label: Enter OTP
- Input field: 520485
- Button: **Verify**

7. **Security Measures:** XSS and injection attacks are prevented through input sanitization, email domain whitelisting, and parameterized MongoDB queries. Data is secured with encryption, password hashing, and session management.

The project successfully delivers a secure platform for cybersecurity discussions, with ZTA ensuring that trust is never assumed, and every interaction is verified and protected.

6 Project Links

For further exploration of SecureChat, the following links provide access to the project's source code, deployed application, and demonstration video:

- GitHub Repository: <https://github.com/fsfrahmad/accessguard>
- Deployed Application on Render: <https://accessguard.onrender.com>
- Project Video Demonstration: <https://example.com/securechat-video>