



# SecureDocs

**“ Comprehensive Details ”**

**Karim Abdelaziz Elsayed 2205037**

**Taha Abdeldadoud Hassan 2205030**

**Ahmed Sabry Ibrahim 2205007**

**Mohamed Naser Elmasry 2205052**

**Karim Waleed Saad 2205076**



## **Table of Contents**

- Introduction
- Requirements
- Software Development Process
- Architectural Design
- System Modeling
- Design & Implementation
- Testing
- Conclusion

# **Introduction**

## **Purpose**

SecureDocs is a Flask-based web application developed to provide a secure, centralized platform for document management. It addresses the challenge of securely storing and sharing sensitive documents by implementing AES-256 encryption, OpenSSL-based digital signatures, and HMAC for integrity verification. The application enhances security through robust authentication mechanisms, such as Two-Factor Authentication (2FA) and Single Sign-On (SSO), and supports role-based access control to differentiate between regular users and administrators. SecureDocs aims to ensure data confidentiality, integrity, and authenticity while offering a modern, user-friendly interface.

## **Definitions**

- **User:** An individual who can register, log in, and perform actions such as uploading, downloading, verifying, and managing documents.
- **Admin:** A user with elevated privileges to manage other users, view audit logs, and oversee system operations.
- **File:** A document uploaded to the system, encrypted with AES-256, signed with OpenSSL, and stored with metadata.
- **Audit Log:** A record of user actions (e.g., login, file upload) maintained for security and compliance.
- **2FA:** Two-Factor Authentication using Time-based One-Time Passwords (TOTP) for additional login security.
- **SSO:** Single Sign-On, enabling authentication via external providers like Okta, Google, or GitHub.
- **JWT:** JSON Web Tokens used for secure session management.

## **Goals and Objectives**

- User Goals:
  - Securely upload, store, and share documents with encryption and digital signatures.
  - Easily manage personal profiles and access files with confidence in their integrity.
- System Objectives:
  - Provide robust security through encryption, digital signatures, and authentication mechanisms.
  - Ensure accountability with detailed audit logs.
  - Deliver a responsive, intuitive interface compatible with desktop and mobile devices.

## **Plan**

- **Phase 1: Requirement Gathering and Problem Analysis:** Identify user needs and security requirements.
- **Phase 2: UI/UX and Database Design:** Design the interface and database schema.
- **Phase 3: Front-End and Back-End Development:** Build the application components.
- **Phase 4: Security and Performance Testing:** Validate security features and optimize performance.
- **Phase 5: Deployment and Maintenance:** Launch the application and provide ongoing support.

# Requirements

## User Requirements

- Authentication:
  - Register and log in using email and password, with optional 2FA setup.
  - Log in via SSO using providers like Okta.
- File Management:
  - Upload files (e.g., .pdf, .docx, .txt), which are encrypted and signed.
  - Download, verify integrity/authenticity, and delete files.
- Profile Management:
  - Update personal details (e.g., name, email, password).
- Admin Functions:
  - Add, edit, or delete users and assign roles.
  - View audit logs to monitor system activity.

## System Requirements

- Functional Requirements:
  - Secure authentication with 2FA (PyOTP) and SSO (Authlib).
  - File encryption using AES-256, digital signatures with OpenSSL, and HMAC for integrity.
  - Role-based access control distinguishes Users and Admins.
  - Real-time audit logging for actions like login, file upload, and user management.
- Non-Functional Requirements:
  - **Security:** Implement HTTPS, input sanitization, CSRF protection, and secure headers.
  - **Performance:** Handle file operations and database queries efficiently for up to 1,000 concurrent users.
  - **Usability:** Provide a responsive UI with glassmorphism design and intuitive navigation.

# **Software Development Process**

## **Software Process Model**

SecureDocs was developed using an Agile methodology, featuring 2-week sprints with iterative development and regular feedback to ensure alignment with security and usability goals.

## **Phases of Development**

### 1. Specification Phase:

- Conducted stakeholder interviews to define requirements.
- Created user stories focusing on secure file management and authentication.

### 2. Design Phase:

- Developed UI prototypes with a glassmorphism aesthetic using tools like Figma.
- Designed a database schema for users, files, audit logs, and password reset tokens using SQLite.

### 3. Implementation Phase:

- Built the front-end with Flask templates, Jinja2, Bootstrap, and custom CSS.
- Developed the back-end using Flask, SQLAlchemy, and Flask-Login.

### 4. Validation Phase:

- Performed unit testing on encryption, authentication, and file operations.
- Conducted integration testing and security audits to ensure system integrity.

## 5. Evolution Phase:

- Planned post-deployment updates based on user feedback, such as additional SSO options.

# Architectural Design

## System Architecture

SecureDocs adopts a three-tier architecture:

1. **Presentation Layer:** Flask templates with Jinja2, styled using Bootstrap and glassmorphism CSS for a modern UI.
2. **Business Logic Layer:** Flask routes handle authentication, file operations, and user requests.
3. **Data Layer:** SQLite database (with potential for PostgreSQL in production) managed via SQLAlchemy ORM.

## Application Architecture

The MVC (Model-View-Controller) pattern was used:

- **Model:** SQLAlchemy models (User, File, AuditLog, PasswordResetToken) for data management.
- **View:** Jinja2 templates render the UI for users.

- **Controller:** Flask routes processing inputs, interacting with models, and updating views.

## Design & Implementation

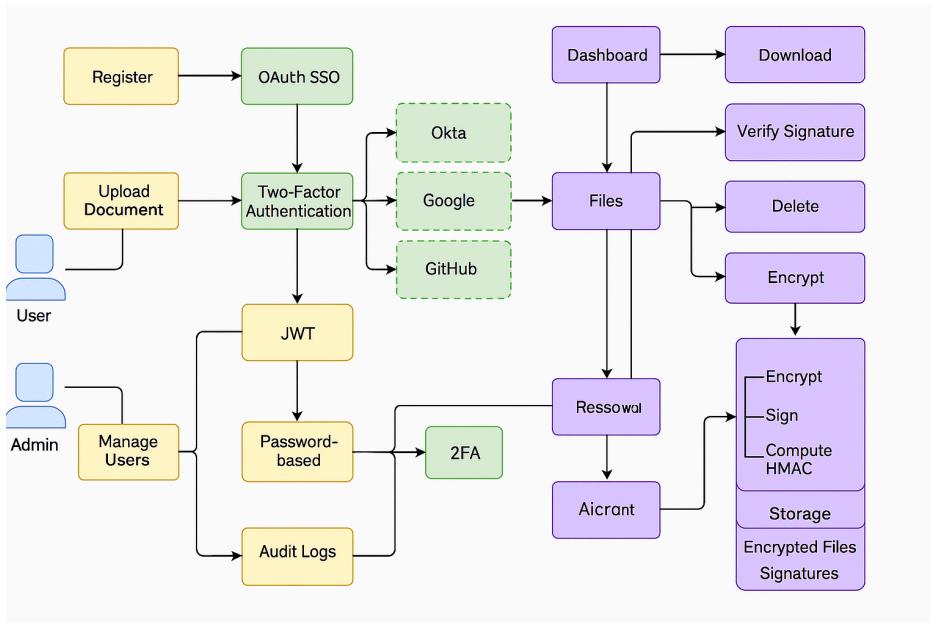
### Design Description

- Front-End:
  - Responsive Flask templates using Bootstrap, enhanced with glassmorphism CSS and animations (GSAP, Animate.css).
  - Features include drag-and-drop uploads, file previews, and modal interactions.
- Back-End:
  - Flask routes for authentication, file management, and admin functions.
  - SQLAlchemy for database operations, with AES-256 encryption and OpenSSL signatures for files.

### Development Environment

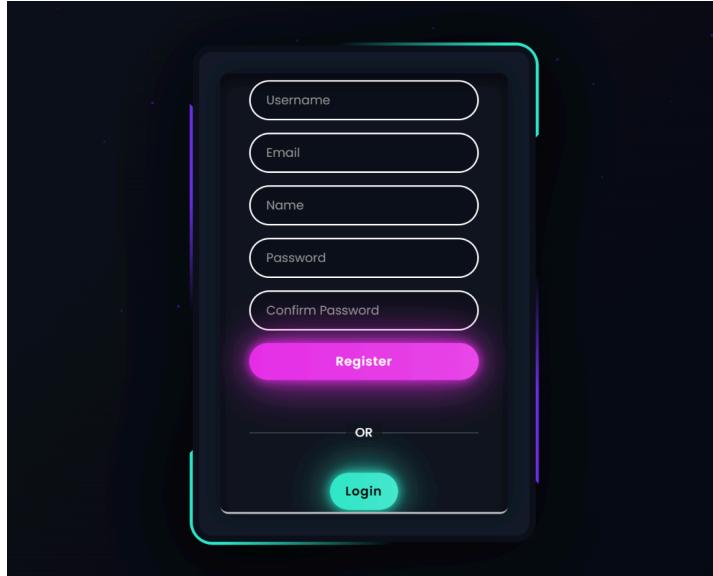
- **IDE:** Visual Studio Code.
- **Database:** SQLite (development), with potential PostgreSQL for production.
- **Tools:** Git for version control, Flask, SQLAlchemy, Flask-Login, PyOTP (2FA), Authlib (SSO).

## Flow Explanation



## Getting Started: Signing Up and Logging In

- Sign Up:



1. A new user provides a username, email, name, and password.
2. They set up Two-Factor Authentication (2FA) by scanning a QR code with an app like Google Authenticator to add an extra layer of security.

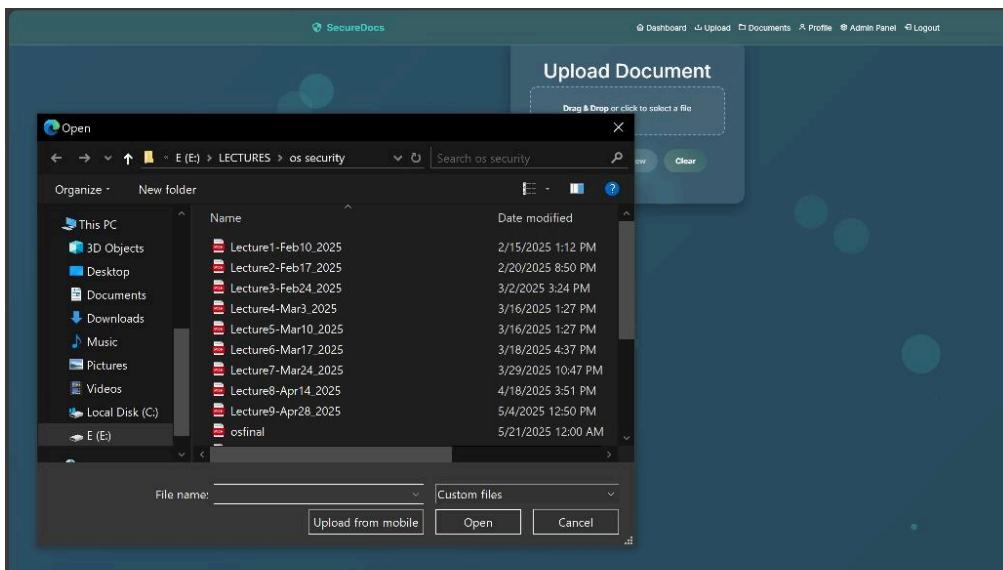
- Login:



- The system gives the user a special token (JWT) to keep them logged in. Right now, this token expires every 30 seconds, so users may need to log in often.

## Working with Files: Upload, Download, Verify, Delete

- Upload a File:

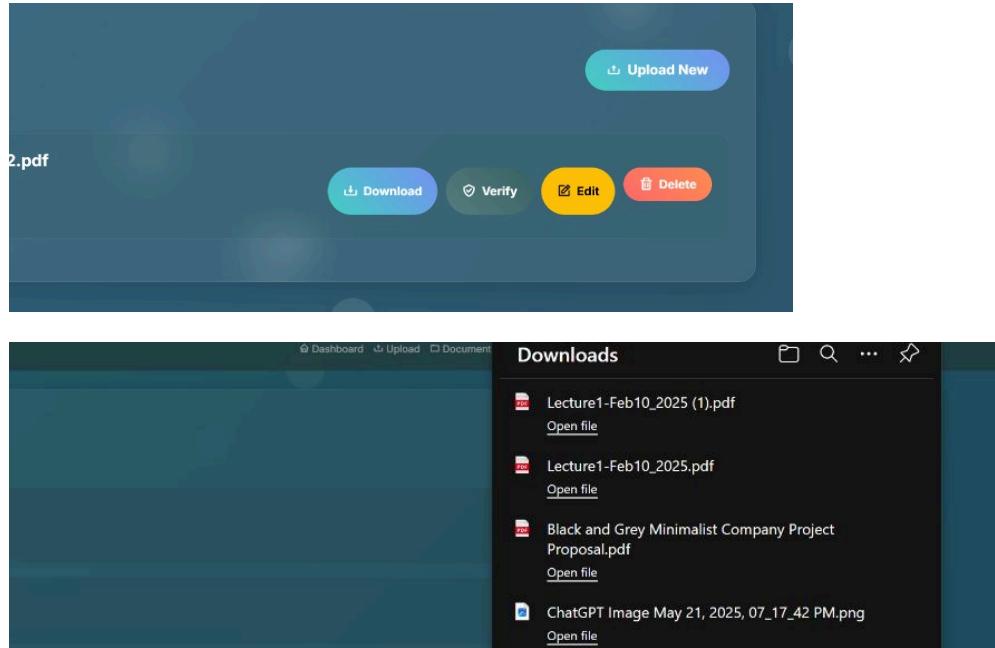


Users select a file (e.g., a PDF or Word document) and can add an optional description.

The system:

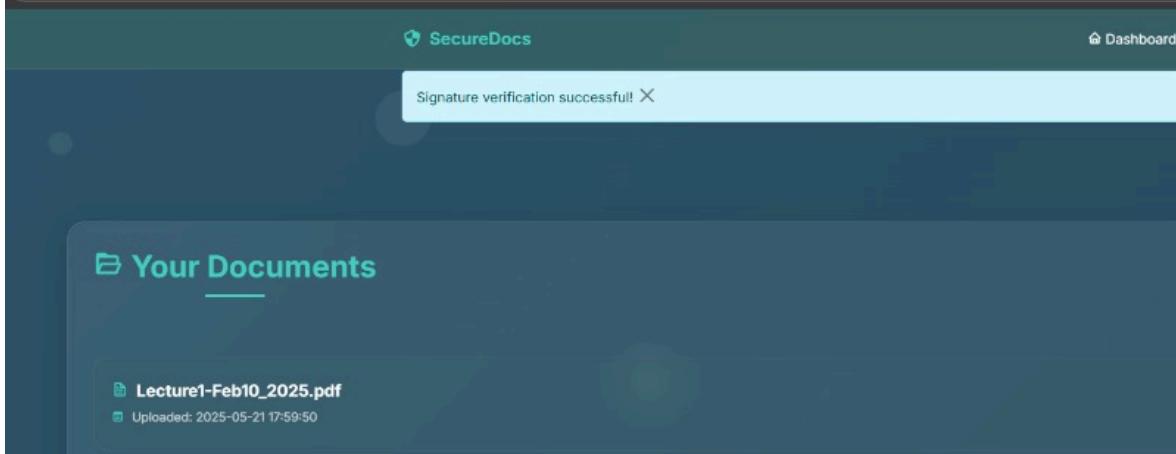
- Encrypts the file using AES-256 to keep it secure.
- Creates a digital signature to prove the file's authenticity.
- Generates an HMAC code to check if the file is altered later.
- Stores the file's details in a database and logs the upload for tracking.

- Download a File:



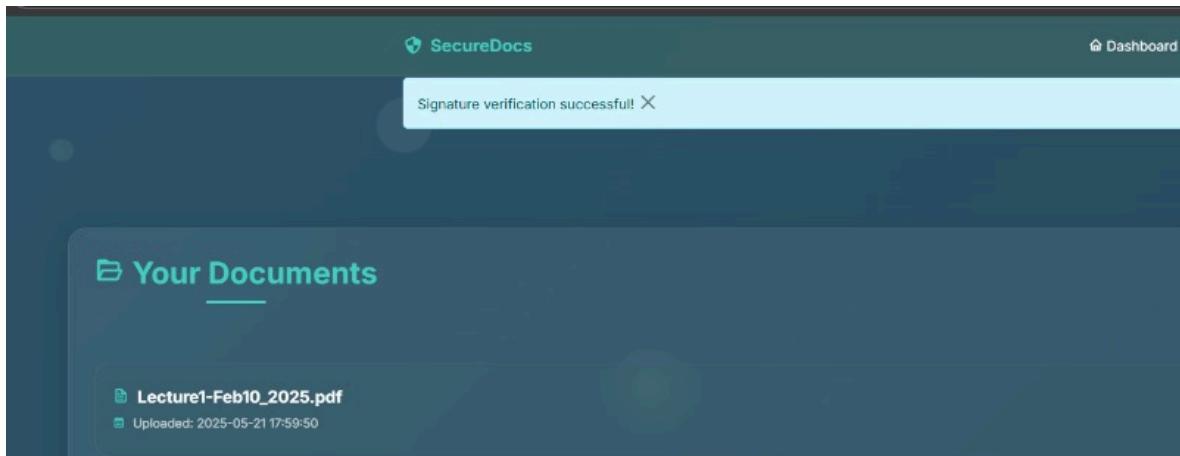
- Users pick a file from their list.
- The system decrypts the file, verifies its HMAC to ensure it hasn't changed, and delivers it to the user.

- Verify a File:



- Users can check a file's digital signature to confirm it's genuine and hasn't been tampered with.

- Delete a File:



- Users can remove a file they no longer need. The system deletes the file, its details, and logs the action.

## Admin Responsibilities: Managing Users and Monitoring

- Manage Users:

The screenshot shows the Admin Panel interface. At the top, there are three navigation buttons: "Manage Users" (highlighted in blue), "Manage Documents", and "Audit Logs". Below the navigation is a section titled "Add New User" with fields for "Username", "Email", "Name", "Password", and "Role" (with a dropdown menu "Select a role"). A green "Add User" button is located at the bottom right of this section. Below this is a section titled "Existing Users" containing a table with two rows of user data. The columns are "ID", "Username", "Email", "Name", "Role", and "Actions". The first user (ID 1) is "admin" with email "admin@example.com", name "Administrator", and role "Admin". The second user (ID 2) is "lkh285@gmail.com" with email "lkh285@gmail.com", name "Ik Th", and role "User". Each user row has a "Edit" and a "Delete" button in the "Actions" column.

ID	Username	Email	Name	Role	Actions
1	admin	admin@example.com	Administrator	Admin	<button>Edit</button> <button>Delete</button>
2	lkh285@gmail.com	lkh285@gmail.com	Ik Th	User	<button>Edit</button> <button>Delete</button>

- Admins can edit user details, remove users, or assign roles (like making someone an admin).
- (Note: The ability to add new users isn't fully built yet.)

- View Audit Logs:

The screenshot shows the Admin Panel interface with the "Audit Logs" tab selected (highlighted in blue). At the top, there are three navigation buttons: "Manage Users", "Manage Documents", and "Audit Logs" (highlighted in blue). Below the navigation is a section titled "System Audit Logs" with a table of log entries. The columns are "Timestamp", "User", "Action", and "Details". The log entries show various system events such as failed logins, successful logins, password verifications, and 2FA verifications for users like "admin" and "N/A" over several dates and times.

Timestamp	User	Action	Details
2025-05-21 17:37:51	N/A	Login Attempt	Failed login for username admin
2025-05-21 17:37:51	admin	Login Attempt	Successful password verification
2025-05-21 17:37:51	admin	2FA Login	Successful 2FA verification
2025-05-21 15:32:17	N/A	Login Attempt	Failed login for username admin
2025-05-21 15:32:17	lkh285@gmail.com	Okta SSO Login	User lkh285@gmail.com logged in via Okta
2025-05-21 15:32:17	N/A	Login Attempt	Failed login for username admin
2025-05-21 15:32:17	N/A	Login Attempt	Failed login for username admin
2025-05-21 15:32:17	admin	Login Attempt	Successful password verification
2025-05-21 15:32:17	admin	2FA Login	Successful 2FA verification
2025-05-21 15:08:53	admin	Login Attempt	Successful password verification
2025-05-21 15:08:53	admin	2FA Login	Successful 2FA verification
2025-05-21 14:57:22	N/A	Login Attempt	Failed login for username admin
2025-05-21 14:57:22	admin	Login Attempt	Successful password verification

- Admins can review a log of all activities—like logins, uploads, or deletions—to monitor the system and ensure everything is secure.

# Testing

## Development Testing

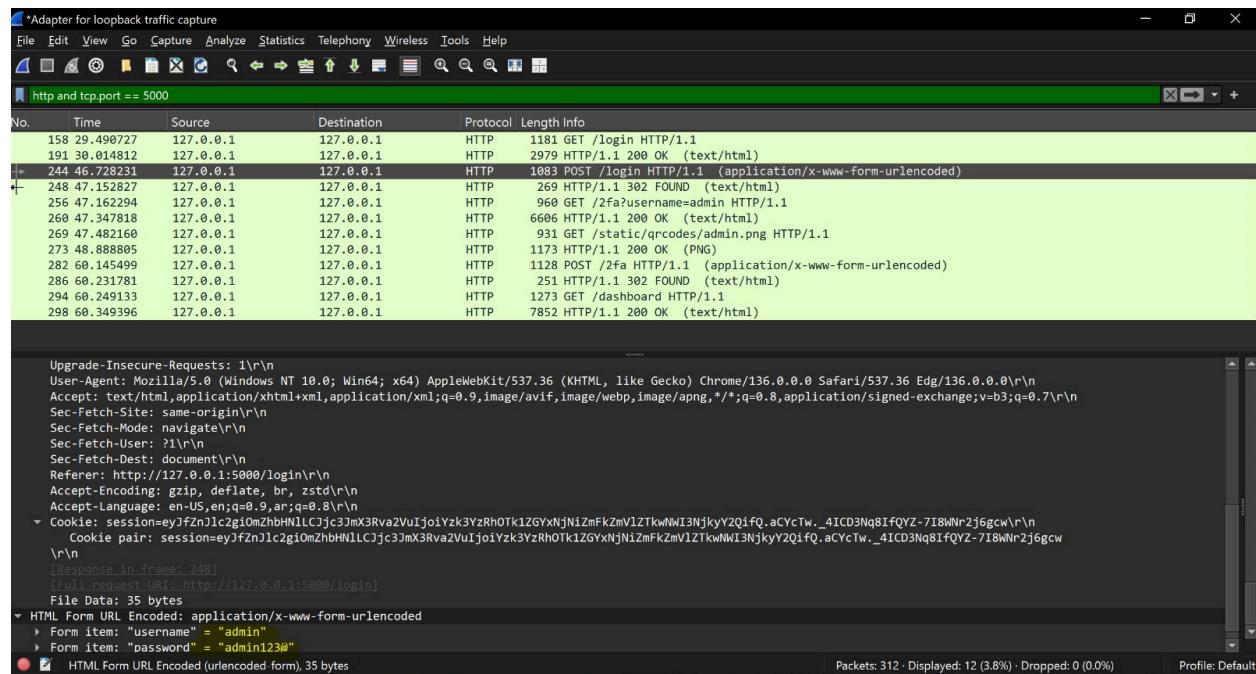
- Unit Testing:
  - Tested encryption/decryption, HMAC generation, and signature verification.
  - Verified authentication modules (password hashing, 2FA).
- Integration Testing:
  - Validated end-to-end file workflows (upload, encrypt, store, retrieve).
  - Ensured front-end and back-end synchronization.

## Release Testing

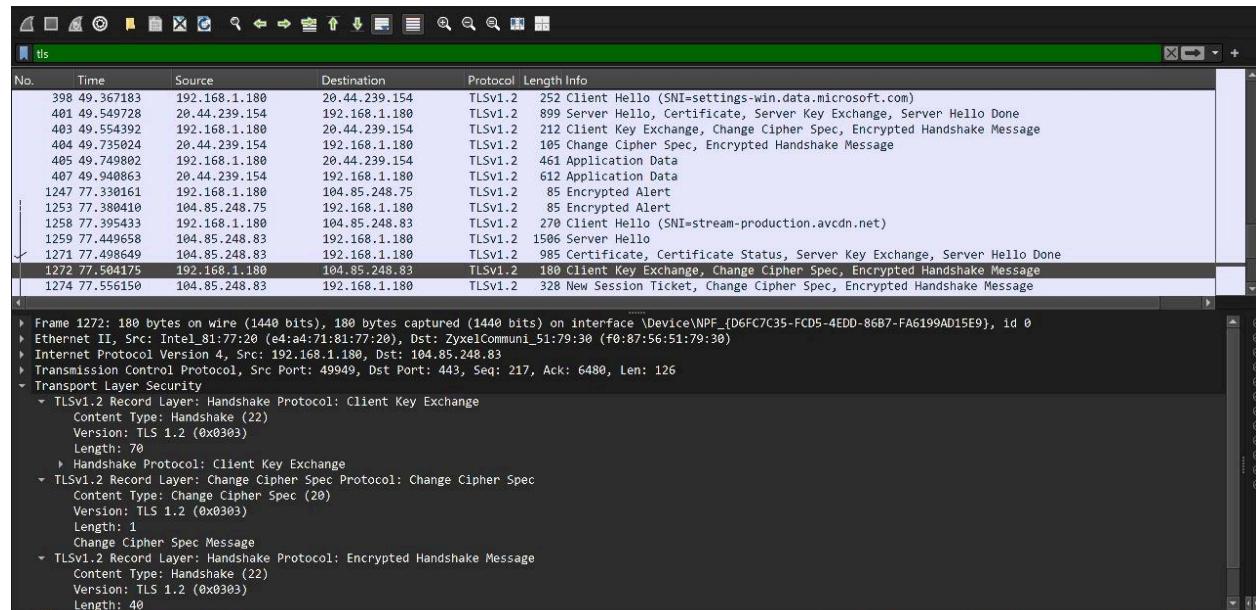
- Security Testing:
  - Conducted penetration testing with tools like OWASP ZAP to identify vulnerabilities.
  - Audited SSO and JWT implementations for secure token handling.
- Performance Testing:
  - Simulated 1,000 concurrent users to optimize file operations and query performance.

## Man In The Middle

- HTTP:



- HTTPS - Mobile to Laptop:



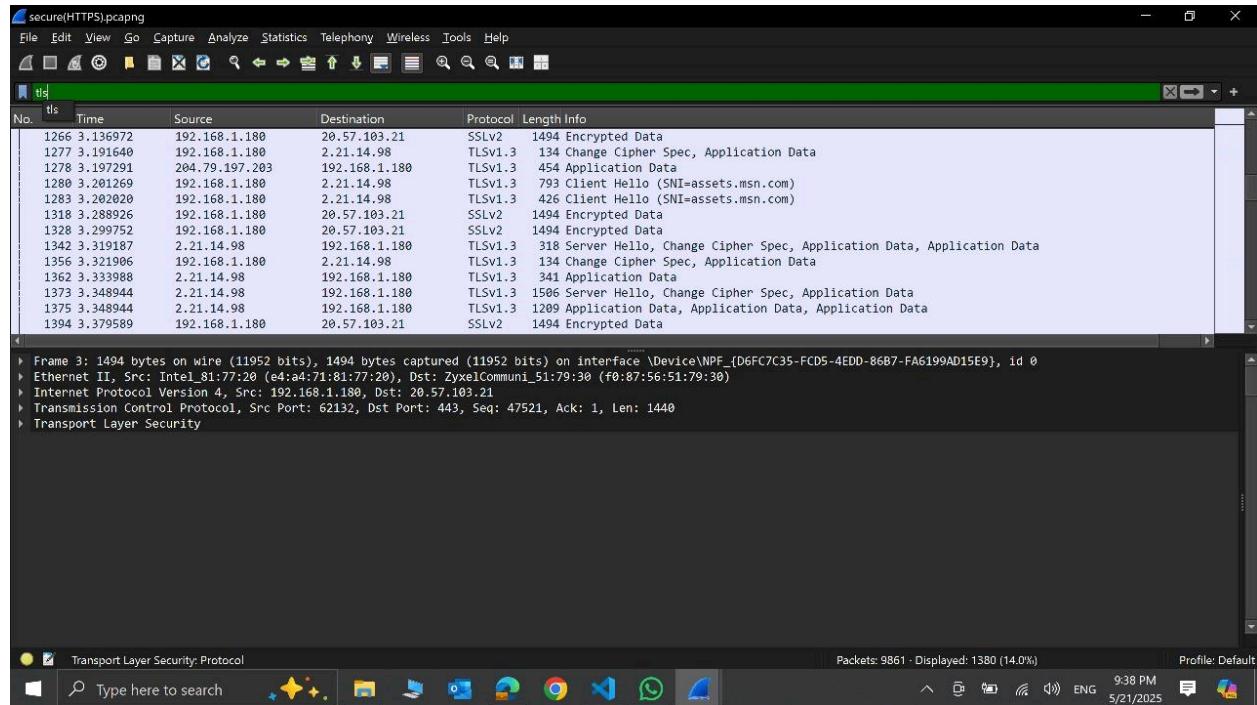
## ● Certificate:

The Wireshark interface displays a capture file named "ts.handshake.certificate". The packet list shows a sequence of frames, primarily TLSv1.2, exchanged between two hosts. The details pane provides a hierarchical breakdown of the handshake process, starting with the Server Hello message and including Certificate, Certificate Status, Server Key Exchange, and Server Hello Done messages.

## ● Encrypted Data:

The Wireshark interface displays a capture file named "Packet 1296 - HTTPS.pcapng". The packet list shows a single frame (Frame 1296) which is identified as an "Encrypted Alert". The bytes pane shows the raw hex and ASCII representation of this message, which consists of a single byte value (26) indicating an alert level of "Alert" (21).

- Local Host to Laptop:



# Conclusion

SecureDocs effectively delivers a secure document management solution with encryption, digital signatures, and strong authentication. The application centralizes file handling while ensuring usability through a modern interface. However, to reach production readiness, the following improvements are recommended:

- **Missing Features:** Add routes for password reset, user management by admins, and file editing.
- **Security Fixes:** Replace hardcoded keys with a key management system and extend JWT expiration.
- **UI Consistency:** Standardize templates (e.g., register.html, reset\_password.html) and remove redundant code.

Future enhancements could include additional SSO providers, file versioning, and scalability optimizations. With these refinements, SecureDocs will fully meet its objectives of security, usability, and reliability.