

**Information Security Lab**  
**Final Report: ZK-File Share**



**Submitted by:**

Noor Fatima	2023-SE-16
Amber Saleem	2023-SE-19
Abdul Rafeh	2023-SE-26

**Submitted to:**

Ms. Shanfa Irum

**Department of Computer Science**  
**University of Engineering and Technology Lahore, New Campus**

# Project Report: ZK-File Share

---

## 1. Introduction

With the growing reliance on digital communication and the increasing transfer of sensitive data across the internet, security in online file sharing has become a major concern. Traditional methods of sharing files are often vulnerable to threats such as unauthorized access, interception, and data breaches. As the risks associated with unsecured file transfers continue to rise, there is an urgent need for more secure solutions. In response to this need, the ZK-File Share was developed, offering a robust solution for users to upload, share, and download files securely.

The application ensures that files are not only encrypted during transfer but also stored in an encrypted format, making them virtually inaccessible to unauthorized parties. By incorporating modern encryption techniques and secure authentication mechanisms, this web application addresses the challenges of data privacy and integrity in online file sharing. With features such as time-limited access tokens and one-time download links, the application guarantees that files are accessible only by the intended recipient, preventing unauthorized access even if the download link is intercepted.

---

## 2. Problem Statement

In an era of increasing cyber threats, users need a reliable platform to share sensitive files securely with encryption, controlled access, and protection against unauthorized downloads.

---

## 3. Features

- User Signup/Login system with password hashing
  - File encryption using AES before upload
  - Secure upload and storage on AWS S3
  - Time-limited download link generation
  - Download feature using encrypted links
  - Input sanitization and session management
  - MongoDB integration for user and file metadata
- 

## 4. Tools and Technologies

- **Backend Framework:** Python Flask
- **Database:** MongoDB
- **Cloud Storage:** Amazon S3

- **Encryption:** AES (Advanced Encryption Standard)
  - **Frontend:** HTML, CSS (via Flask templates)
  - **Libraries:**
    - Flask
    - Boto3 (for AWS S3)
    - PyCryptodome (for AES encryption)
    - Werkzeug (for password hashing and sessions)
- 

## 6. Detailed Explanation of Project Code

### User Authentication

The user authentication system is handled within the same `app.py` file. It manages the `/signup`, `/login`, and `/logout` routes. For password security, the application uses Werkzeug's `generate_password_hash` and `check_password_hash` functions. This ensures that passwords are never stored in plain text. During signup, user credentials (with the hashed password) are securely stored in the MongoDB `users` collection. During login, the system retrieves the hashed password from MongoDB and verifies it against the input. Flask's session management is used to maintain user sessions securely, and `/logout` clears the session to prevent unauthorized access.

---

### File Upload

The `/upload` route is protected so that only authenticated users can upload files. When a user uploads a file, it is first encrypted using AES encryption implemented within the same file. A randomly generated key and IV ensure strong confidentiality. The encrypted file is then uploaded to an AWS S3 bucket using the `boto3` client. A unique UUID token is generated for each file and stored in MongoDB along with metadata such as the filename, upload timestamp, and expiry time (e.g., 10 minutes). This token is returned to the user so they can share it securely with the intended recipient.

---

### File Download

The file download logic is triggered via the `/download/<token>` route. When a user accesses this route, the application checks MongoDB for a matching, non-expired token. If valid, the corresponding encrypted file is fetched from AWS S3 using its stored key. The system then decrypts the file using the same AES encryption method and streams it back to the user using

Flask's `send_file()` method. If the token is invalid or expired, an error message is displayed. This method ensures secure and temporary access to shared files.

---

## Encryption and Decryption

For encryption and decryption, the application uses the AES cipher from the `Crypto.Cipher` module in CFB (Cipher Feedback) mode. This mode is chosen because it allows encryption of data in smaller blocks, suitable for variable-sized files. Each encryption uses a randomly generated Initialization Vector (IV), which ensures that even the same file will produce different encrypted outputs on each upload. Encrypted binary data is then Base64-encoded to safely store and transmit it. On download, this data is decoded and decrypted back to its original form.

---

## AWS S3 Integration

Integration with Amazon S3 is done using the `boto3` library. The app uses AWS credentials (configured via environment variables) to authenticate with S3 and provides utility functions to upload encrypted binary files and retrieve them during downloads. Files are uploaded using the `put_object` method and retrieved using `get_object`. This ensures reliable cloud storage, and access to the bucket is tightly controlled to prevent unauthorized access.

---

## MongoDB Models and Token System

MongoDB is used to store all user and file-related data. The `users` collection stores usernames and their hashed passwords. The `files` collection stores metadata about uploaded files, including the filename, token, expiry time, and uploader username. During download, MongoDB is queried to ensure the provided token exists and hasn't expired. This token-based access control system provides secure and time-limited file sharing, preventing URL guessing and misuse.

---

## Security Measures

The application implements multiple layers of security. Files are encrypted using AES encryption to ensure confidentiality. User passwords are stored as salted hashes, protecting them even if the database is compromised. Tokens used for file sharing are random UUIDs, making them nearly impossible to guess. Each token is also associated with an expiry time to ensure that access is only available for a short period. Sessions are managed using Flask to keep users authenticated securely without exposing sensitive data. Additionally, original filenames are never exposed publicly, and secure UUIDs are used for file references.

---

## Access Control Method

In this project, a combination of **token-based access control**, **session authentication**, and **light role-based access control (RBAC)** is used to protect file sharing and ensure privacy.

Authenticated users are allowed to upload files, which are encrypted before being stored on AWS S3. A unique token is generated per file with an expiry time, allowing only the intended recipient to download the file. This time-limited token serves as a secure, temporary permission. Additionally, a minimal role-based system is in place: while regular users can only upload and receive files, the **admin can delete any file** from cloud storage but **cannot decrypt or access file contents**, ensuring data privacy is preserved.

---

## Implemented Access Control Points

- **Session-Based Authentication**
    - Implemented using Flask sessions.
    - After login, a session is created for the user and required to access routes like /dashboard, /upload, etc.
  - **Token-Based File Access**
    - A unique UUID token is generated when a file is uploaded.
    - The token is stored with an expiry time in MongoDB.
    - Only users with the valid, non-expired token can download the file via /download/<token> route.
  - **Role-Based Access Control (Light)**
    - Regular users can only upload and download files.
    - Admin can delete files from AWS S3 using a dedicated function.
    - Admin cannot view, decrypt, or download the actual file contents, as files are stored in encrypted form.
  - **File Ownership Isolation**
    - Users can only access files they have the token for.
    - No file listing or directory browsing is allowed, preventing unauthorized access.
- 

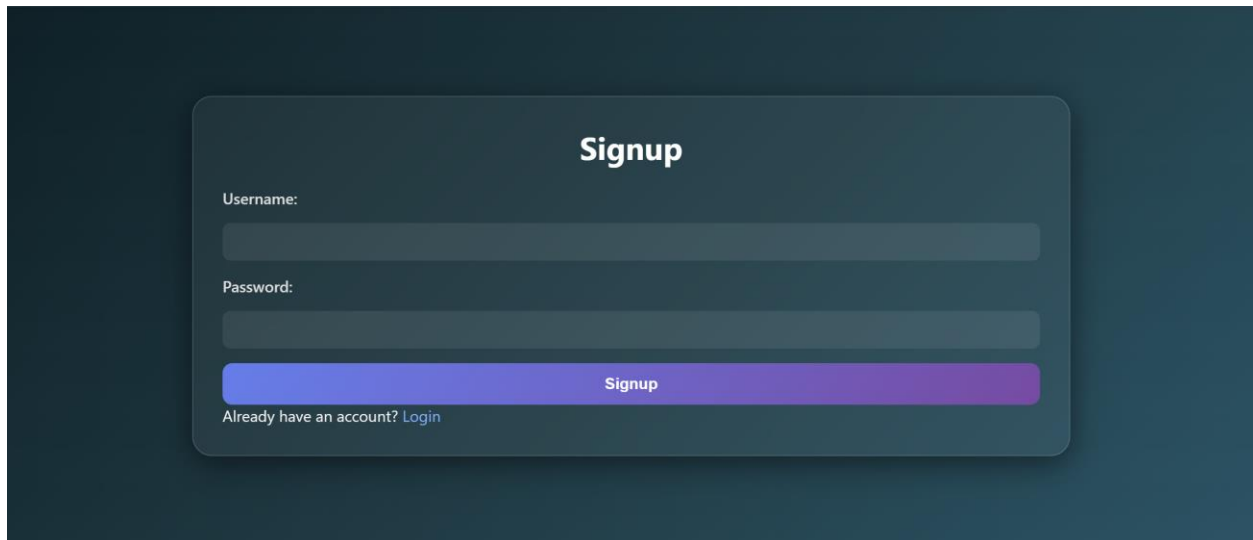
## 9. Conclusion

The Secure File Sharing Web Application successfully demonstrates how cryptographic techniques and secure web practices can be combined to build a privacy-focused file-sharing system. With real-time encryption, secure storage, and strict access control, the project aligns with modern data security expectations and serves as a solid foundation for real-world secure file transfer systems.

---

## 10. Screenshots

### Signup:



A screenshot of a dark-themed 'Signup' form. The form is centered on a dark blue background. It features a title 'Signup' in white. Below the title are two input fields: 'Username:' and 'Password:'. A prominent purple 'Signup' button is positioned below the password field. At the bottom, there is a link that says 'Already have an account? Login'.

Signup

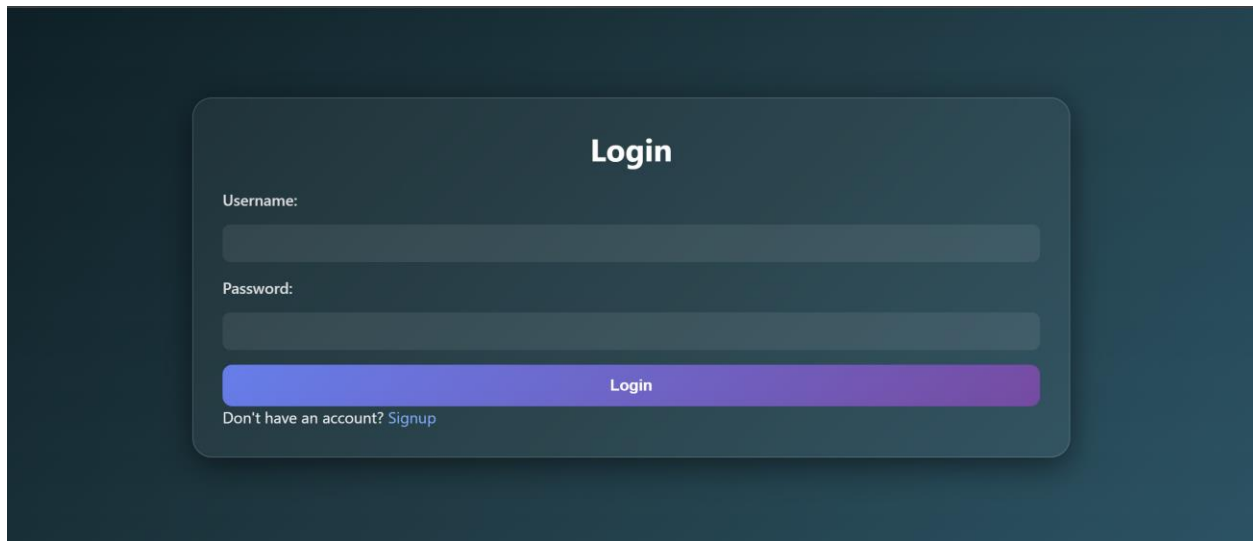
Username:

Password:

Signup

Already have an account? [Login](#)

### Login:



A screenshot of a dark-themed 'Login' form. The form is centered on a dark blue background. It features a title 'Login' in white. Below the title are two input fields: 'Username:' and 'Password:'. A prominent purple 'Login' button is positioned below the password field. At the bottom, there is a link that says 'Don't have an account? Signup'.

Login

Username:

Password:

Login

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

## Sending Dashboard:

### Dashboard

#### Sending Panel

Upload File:

Choose File

No file chosen

Encrypt File:

☐

Send

#### Receiving Panel

Enter Code:

Decrypt File:

☐

Receive

Your Code: N47YYV

Expires in: 586 seconds

## Receiving Dashboard:

### Dashboard

#### Sending Panel

Upload File:

Choose File

No file chosen

Encrypt File:

☐

Send

#### Receiving Panel

Enter Code:

N47YYV

Decrypt File:

☒

Receive