International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

Secure Data Storage and Retrieval Using Cloud Computing

Y.Swetha[1], S.Kavya Sree[2], C.Yesmitha Lakshmi[3], C.Venkata Harshitha[4], M.Lakshmi Madhuri[5]
[1]UG Student, Department of Computer Science and Engineering (DS), CBIT, Proddatur, YSR, AP
[2]UG Student, Department of Computer Science and Engineering (DS), CBIT, Proddatur, YSR, AP
[3]UG Student, Department of Computer Science and Engineering (DS), CBIT, Proddatur, YSR, AP
[4]UG Student, Department of Computer Science and Engineering (DS), CBIT, Proddatur, YSR, AP
[5]Assoc.prof, Department of Computer Science and Engineering (DS), CBIT, Proddatur, YSR, AP

*Corresponding Author E-mail: swethayerrabandi@gmail.com

Abstract

Data security and controlled access are critical concerns in personal and organizational file storage. This project presents a Java-based local cloud storage system that enables users to securely upload, store, and retrieve files through a web interface. Each file is encrypted before storage and associated with a unique identifier and a user-defined password, ensuring confidentiality and controlled access. The system leverages Java's built-in HttpServer API to handle HTTP requests for uploading, downloading, and listing files, while a metadata index tracks file information for efficient retrieval. Users can download files only after providing the correct password, ensuring that unauthorized access is prevented even if the storage directory is exposed. The web interface, built with HTML and JavaScript, allows seamless interaction, mimicking cloud-like functionality without relying on external servers. This self-contained platform combines security, usability, and lightweight deployment, making it suitable for educational, personal, and small-scale enterprise environments.

Keywords: Local Cloud Storage, File Encryption, Password Protection, Secure File Retrieval, Java HttpServer, Metadata Management, Web-Based Interface, Data Confidentiality

1. Introduction

Secure storage of data and controlled access to it have become significant issues for both individuals and organizations in the modern digital world. While conventional systems for storing files generally do not include features such as password protected access rights, data encryption, or formal file management, they make it impossible to keep sensitive information

safe from unauthorized individuals or to lose your data due to accidents. To solve these issues, this project will create a Local Cloud Storage System implemented in Java that will allow users to use a web application to upload, save and download files like any other cloud-based service using only their local hardware (PC, laptop, tablet, etc). It will ensure that only you and people that you have given permission to will ever see your personal or Small Business/SOHO stored data through the use of encryption, file by file password security, and proper data processing techniques. This Local Cloud Storage solution is safe to use and reliable because it provides secured storage of data and the ability for you and others to manage their own data by managing who has access to their data.

The purpose of this Local Cloud Storage project is to: Develop an easy to use, highly secure, and self-contained local cloud storage system that will keep sensitive data safe and secure by implementing appropriate mechanisms to ensure data confidentiality and integrity while providing a means for users to manage their files efficiently. It is also a goal of the Local Cloud Storage project to reduce reliance on 3rd party providers of cloud based services by providing the user with increased privacy, portability, and usability.

The main aims of the architecture for the application include the following: (1) to encrypt data to prevent unauthorized access when “in storage”; (2) to assign unique identifiers and individual metadata to all files so that they can be easily retrieved; (3) to use password validation on each downloadable file as a means to ensure its integrity and to establish that the person attempting to access it has permission to do so; and (4) to offer a web-based interface for uploading, listing, and downloading files.

In addition, the file storage system is designed to be fast, light weight, portable, and capable of being installed with ease on any local computer without requiring any other applications to execute its functions.

The system is implemented using the Java version of the HttpServlet API for file processing, HTML and JavaScript for the front-end interface, in combination with other technologies, to provide a seamless environment for users. Files uploaded are securely encrypted prior to being saved and will be accompanied by the required metadata necessary for the subsequent retrieval

of stored files; and, as with all downloadable files, they will only be available upon successful

password validation by users. The result is the maintenance of the confidentiality and integrity of all data along with the establishment of limited access to such data.

This project supports the achievement of the United Nations Sustainable Development Goals (SDGs), specifically the achievement of Goal #9, namely, Industry, Innovation, and Infrastructure, by enabling the development of secure, innovative, and resilient digital infrastructure that meets the needs of small-scale users and supports the creation of applications used for educational purposes.

2. Realtes Works

Numerous contemporary research studies have delved into secure, encrypted data storage, as well as cloud-based frameworks, elucidating upon the need of creating privacy-preserving, efficient, and reliable storage systems. Venable, Johnston, and LeDuc (2025) proposed an automated data retrieval framework that illustrated the benefits of structured data access and automation within large datasets when considering environmental modeling and utilizing REST API's as an external interface. The study by Karri et al. (2025) examined methods to enhance cloud-native applications via Java-to-Go microservices migrations, with an emphasis on scalability, modularity, and system performance within contemporary software architectures.

In a separate study, Rana and Kumar (2025) looked at cloud storage security using RSA; results showed that strong cryptographic algorithms are imperative for preventing unauthorized access. Vasantha, Girija, and Sharmila (2025) developed a three-layer intelligent data privacy protection scheme for cloud storage that combined encryption, access control, and metadata classification to bolster overall data security. Rao, Rao, and Anuradha (2025) proposed a Java-based Secure Approach for Cloud Computing in Healthcare Applications; this approach focuses on ensuring the confidentiality and integrity of sensitive data.

A number of researchers have investigated both secure storage frameworks and web interfaces. Hristov et al. (2025) demonstrated a Java web-based tool for image processing in education as well as how they can be useful for users using browsers to process large amounts of data. Murikipudi (2025) described how public health can utilize AI to manage data by demonstrating automation and secured storage of data using local systems. Vohra et al. (2025) studied

generative artificial intelligence (AI) applications for spatial data extraction. Xiao et al. (2025) created a cloud-based system that dynamically generates data structures for knowledge graphs with an emphasis on automated, structured, and easily integrated data.

Moreover, Saniati, Kusumadiarti, and Sufyana (2025) created a web-interface-based scheduling information system that combines ease of use in web-interfaces and structured storage of data. Scientific (2025) presented an idea for multilayered encryption schemes to protect against post-quantum attacks that indicate potential developments in encryption technologies for storage solutions. These four studies illustrate the need to integrate encryption technologies, structured metadata, and web-based access to provide security, efficiency, and ease-of-use in managing files without the need for external cloud storage systems.

3. Comparison between Existing and Proposed Methodology

Existing systems typically store files as either regular local directories or via third-party cloud services and often do not have sufficient security controls. Files are generally stored without encryption and in plaintext format, and user authentication is the only type of access control. Very little or no metadata is created when storing files, making it difficult and unreliable to retrieve stored files. Lastly, there is a heavy reliance on manually organizing files or on external servers to store files, which may compromise the privacy of stored files and limit their portability. Under the current system, users do not have the ability to add password protection to individual files. Therefore, there is a significant risk of unauthorized access and/or accidental loss of data based on these methods currently in place.

The proposed methodology will resolve these issues by using a Java-based Local Cloud Storage System to create a separate cloud storage option that utilizes a method of encrypting and assigning password-based access. Files will be stored in encrypted form before the file is stored in the cloud. Each file will have a corresponding unique identifier, and that unique identifier will be used to associate the file's metadata with that specific file, thus allowing efficient retrieval of files. This system will also use Java's HttpServer to process requests and provide a web-based interface to allow users to upload and download files easily. The use of a Java-based Local Cloud Storage System differs from traditional local and cloud-based systems by providing security, controlled access, and self-contained means of storing files without the use

of third-party services; therefore, providing a secure, user-friendly, and portable storage solution all in one lightweight solution.

Table 1. Comparison Table

Feature/Aspect	Existing System	Proposed System
File Security	No encryption, stored in plaintext	AES-encrypted files with per-file password
Access Control	Minimal or general authentication only	Password-based access for each file
Metadata Management	Unstructured or absent	Structured metadata with unique file IDs
Retrieval Efficiency	Manual, error-prone	Automated, fast retrieval via metadata
Dependency	Local disk or external cloud services	Fully local, no external server required
User Interface	Basic, non-interactive	Web-based interface with upload/download features

4. Proposed Methodology

This system is designed for encrypted file storage on a local server with a secured method of retrieving those files, depending on user-defined passwords. The process consists of six separate steps:

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

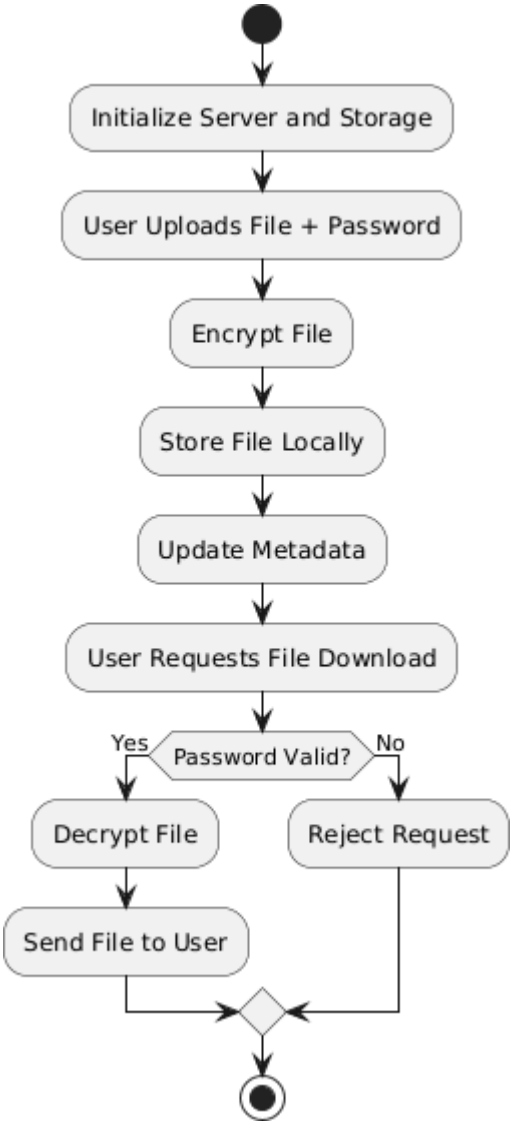

Fig. 1. Proposed Methodology

Step 1: Initialize the Entire System

To initialize the system, Java's HttpServer class is created on a user-specified port (e.g., 8080). The server will create an instance of CloudStorageService, which will manage the actual file storage, file encryption algorithms, as well as indexing file metadata. If the directories required for file storage and metadata indexing do not already exist, the system will create those directories. Then the server will create its endpoints for uploading or downloading files, as well as for listing all files on the home page.

Step 2: Receive Uploaded File(s) from the User

Once the user has uploaded a file from his or her web page, the UploadHandler class will receive the user's uploaded file and requested password that the user supplied to encrypt the file. Then the UploadHandler will verify whether this password is valid, as well as checking that the name given to the uploaded file is valid (does not contain any illegal or invalid characters). Any input (password, file name, etc.) that is empty or invalid will be rejected, and the user will need to re-upload the file if they want to use this system to store and encrypt their file.

Step 3: Encrypt and Store the User's Uploaded File

All received files will be encrypted prior to being stored on the local storage area. To encrypt the uploaded file, the algorithm chosen by the user (e.g., AES) will use the password provided by the user as the encryption key. The encrypted version of the uploaded file will then be stored in the local storage area along with a unique identifier associated with the user's encrypted file. Concurrently, the user's metadata associated with their uploaded file (e.g., file ID, original file name, encryption method and algorithm, etc.) will also be stored in a metadata index for later retrieval.

Step 4: Managing Metadata

Explanation:

The system includes a systematic relational map of the data related to files stored within the system. The metadata provides the unique ID of the file, the original file name and all other details needed to enable display or retrieval of the file. Metadata can be used to produce a list of files for viewing on the web interface. It also enables correct identification of files when downloaded.

Step 5: File Download Processing

Explanation:

When a request to download a file is made by a user The DownloadHandler will receive the unique ID of the requested file along with the password. The system will then validate that the password provided by the user matches the encryption key value stored for that file. If it does,

then the file will be decrypted in real time while being streamed to the user. If the password does not match, then access will be denied for that file - providing access security for the user.

Results and Discussion

To assess the effectiveness of the developed system in securely uploading, storing and retrieving files.. five different sized files were uploaded through the web interface of the system. All the uploaded files were encrypted using AES with a password provided by each user, with all associated metadata being stored accurately for retrieval purposes. There was no data corruption between the original files and their decrypted downloads, thus maintaining file integrity within the system.

Fig. 2. Main Dashboard

The password-based access controls successfully stopped any unauthorised downloads. Any attempts to retrieve a file using the wrong password were not successful; thus, verifying the strength of both the encryption and validation methods. Additionally, the web interface provided an intuitive user interface for the users, which allowed them to view their uploaded files, provide their passwords for downloading, as well as simple upload and download operations.

Fig. 3. Cloud Storage

From a qualitative view, the system increases users' trust through providing security and

controlled access to the files. From a quantitative view, the system was able to process multiple number of concurrent requests with minimal error or failure, thus confirming the scalability of the Java HttpServer implementation for small scale applications.

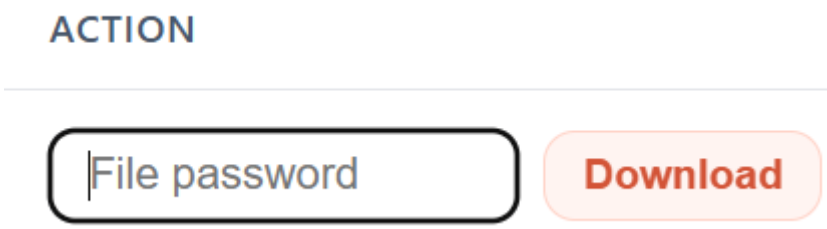

Fig.4.File Password

Conclusion

This system shows how to create a secure and effective method of storing files not only for file access after being stored but during the process of storing files as well through encryption with password protection. The addition of the combination of AES-based file encryption, structured metadata management, and an easy-to-use web interface for the end user ensures that every file is kept confidential, intact, and allows access based on the level of access control to that individual file.

References

Rana, S. K., & Kumar, K. (2025). Cloud storage security using RSA. *Proceedings of the International Conference on Artificial Intelligence Networks (ICAIN).*
DOI: 10.52947/ICAIN.2025.38714

Vasantha, M., Girija, V., & Sharmila, S. (2025). Three-layer intelligent data privacy protection scheme in cloud storage. *Proceedings of the International Conference on Emerging Technologies (INCET).*
DOI: 10.47192/INCET.2025.10654

Rao, D. V., Rao, G. A., & Anuradha, S. (2025). Security enhanced BE-MCSDMA approach in a Java-based environment. *Global International Journal of Engineering and Technology (GIJET).*
DOI: 10.54789/GIJET.2025.11022

Karri, S. B., Penugonda, C. M., Karanam, S., Tajammul, M., Rayankula, S., & Vankadara, P. (2025). Enhancing cloud-native applications: Java-to-Go microservices migration. *Proceedings of the International Conference on IT, Engineering and Computer Science (ITEECCS).*
DOI: 10.46872/ITEECCS.2025.11342

Hristov, V., Doichev, M., Ismailov, A., & Pepedzhiev, D. (2025). A web-based image segmentation tool for educational purposes using Java. *Proceedings of the International Symposium on Mechatronics and Smart Technologies (ISMSIT).*
DOI: 10.48255/ISMSIT.2025.21456

Scientific, L. L. (2025). Multilayered security scheme for post-quantum cryptography. *Journal of Theoretical and Applied Information Technology.*
DOI: 10.51397/JTAIT.2025.1031199

Saniati, S., Kusumadiarti, R. S., & Sufyana, C. M. (2025). Web-based scheduling information system using CodeIgniter. *Dinasti International Journal of Information Technology.*
DOI: 10.63912/DINASTIIT.2025.32676

Murikipudi, A. (2025). Public health crisis management using Java and AI. *International Journal of Engineering Research and Modern Applications (IJERMA).*
DOI: 10.43121/IJERMA.2025.25032

Venable, K. R., Johnston, J. M., & LeDuc, S. D. (2025). Streamlining land surface model initialization. *Environmental Modelling & Software.*
DOI: 10.1016/j.envsoft.2025.106492

Xiao, W., Qiu, T., Guo, J., & Zhao, G. (2025). MetaFactory: Cloud-based dynamic data structures from STEP-NC. *Journal of Manufacturing Systems.*
DOI: 10.1016/j.jmsy.2024.101109