ISSN (Online): 2456-0448
International Journal Of Innovative Research In Management, Engineering And Technology
Vol. 10, Issue 10, October 2025

Decentralized Cloud Storage System

[1] Bakyalakshmi . M , [2] Angelin Rosy.M

[1] Department of Computer Applications Er. Perumal Manimekalai College Of Engineering, Hosur, Tamilnadu, India.

[2] Assistant Professor, Head of the Department of Computer Applications Er. Perumal Manimekalai College Of Engineering, Hosur, Tamilnadu, India.

Abstract: ***The exponential growth of data in the digital age has made secure, efficient, and reliable data storage a crucial challenge. Traditional cloud storage solutions rely on centralized servers, making them vulnerable to single points of failure, data breaches, and privacy concerns. This paper presents a decentralized cloud storage system designed using React for the frontend, Python for backend processing, and Firebase for authentication and metadata management. The proposed system divides files into encrypted chunks distributed across multiple nodes, improving reliability and data security.***

Keyword – Decentralized cloud storage, Firebase authentication, Metadata management, Distributed storage, Data reliability, Data security, Reliable data storage, Centralized servers, Single point of failure.

I. INTRODUCTION

1.1 Cloud Storage Systems

Cloud storage services have revolutionized how individuals and organizations store, manage, and access digital data. These systems allow users to upload and retrieve information from any location with an internet connection. The convenience and scalability of cloud storage have made it a cornerstone of modern computing. However, despite its advantages, traditional cloud models often face challenges related to control, security, and trust.

1.2 Limitations of Centralized Cloud Architecture

Conventional cloud storage platforms depend on centralized servers managed by a single organization. This centralized approach introduces several risks, such as system crashes, server downtime, and data corruption. Moreover, since all data is stored and maintained by a single provider, users are exposed to potential privacy violations, unauthorized access, and data breaches. A single point of failure in centralized systems can lead to complete data loss or service interruption.

1.3 Decentralized Cloud Storage Concept

A decentralized storage system overcomes the limitations of traditional cloud architecture by distributing data across multiple nodes or peers in a network. Instead of storing entire files on a single server, data is split into smaller encrypted chunks that are distributed among different nodes. This design ensures redundancy and fault tolerance—if one node fails, the file can be reconstructed using chunks from other nodes. The decentralized model enhances security, reliability, and user control over data.

1.4 System Features and Design Goals

The system is designed with key goals of security, redundancy, and reliability. Each uploaded file is encrypted and divided into smaller chunks before being distributed across multiple nodes, preventing unauthorized access or data loss. The redundancy feature ensures that even if one node goes offline, the data can be reconstructed from other nodes. The system also emphasizes user-friendly design to make decentralized storage accessible to non-technical users.

II. LITERATURE REVIEW

2.1 Inter Planetary File System (IPFS)

The Inter Planetary File System (IPFS) is a distributed peer-to-peer protocol designed to store and share files using a content-based addressing mechanism. Instead of relying on traditional location-based storage, each file in IPFS is identified by a unique cryptographic hash, ensuring data integrity and immutability. This approach enhances file authenticity, reduces duplication, and allows faster access through distributed nodes.

2.2 Storj
Storj is an open-source decentralized storage platform that encrypts and fragments data before distributing it across multiple independent nodes. The system uses client-side encryption to ensure privacy and reliability. Users can contribute unused storage space to the network and, in return, earn rewards. Storj ensures data redundancy and availability, even if certain nodes become inactive, making it a reliable and secure decentralized solution.

2.3 File coin
File coin extends IPFS by integrating a block chain-based incentive layer. It enables users to rent or provide storage and receive cryptocurrency tokens as rewards for their contributions. The decentralized economy of File coin promotes scalability, reliability, and resource optimization across the network.

2.4 Key Technologies
React (Frontend Development)
React is a powerful JavaScript library developed by Facebook for building dynamic and responsive user interfaces. In this project, React is used to create an intuitive and interactive frontend that allows users to upload, download, and manage files easily.

Python Flask (Backend Framework)
Flask is a lightweight and flexible web framework written in Python. It is used to handle backend logic, including file processing, and communication between the frontend and storage nodes.

III. METHODOLOGY

3.1 Requirement Analysis
In this stage, the functional and non-functional requirements of the system were identified. The system needed to support secure user authentication, file upload and download features, file encryption, chunking, and simulated distributed storage. The core priorities established were security, reliability, and scalability to ensure efficient and safe data handling. The non-functional requirements focused on system performance, reliability, scalability, and security. Reliability was emphasized to guarantee continuous access to files even if some nodes failed. Scalability ensured that the system could accommodate more users and larger data sizes without significant performance degradation.

3.2 System Design
The design phase divided the system into three main layers: the frontend, backend, and database. The frontend provides the user interface, the backend handles data processing and encryption, and the database stores authentication details and metadata. This layered structure ensures modularity, easy debugging, and better scalability for future enhancements.

Tools and Technologies Used in System Development

Database and Authentication

- The system employs Firebase Firestore as the database for storing metadata, user credentials, and file hash values.
- Firebase provides real-time synchronization, built-in authentication, and strong security rules that protect user data.
- It ensures efficient and secure data storage while maintaining high availability and reliability.

Development Tools

- The development environment included Visual Studio Code as the primary code editor, GitHub for version control, and Google Chrome Developer Tools for testing and debugging.
- Together, these tools streamlined the development workflow and ensured code consistency across all modules.

Data Collection Methods for Evaluating Impact

1. Evaluating the performance and effectiveness of the proposed decentralized cloud storage system requires a systematic approach to data collection. Various qualitative and quantitative methods were used to assess the system's functionality, security, reliability, and user experience.

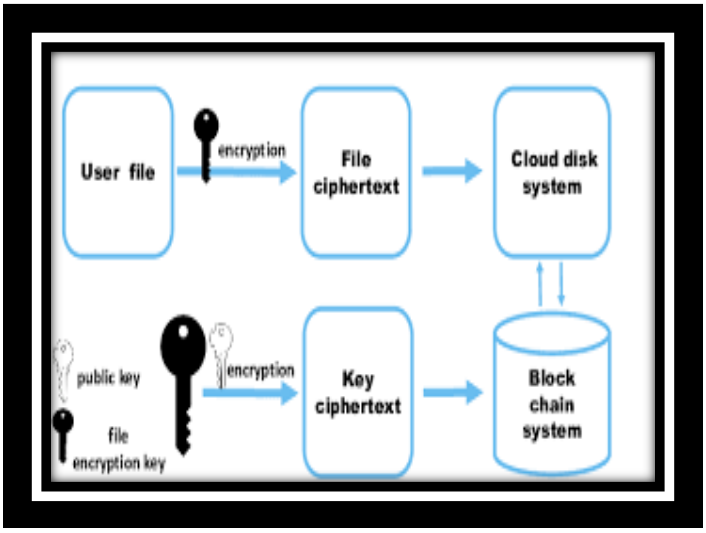
2. User feedback was collected from individuals who interacted with the system to perform file uploads, downloads, and authentication operations. Observations during usage helped identify interface responsiveness, ease of navigation, and overall user satisfaction.

ISSN (Online): 2456-0448
International Journal Of Innovative Research In Management, Engineering And Technology
Vol. 10, Issue 10, October 2025

3. Functional testing was conducted to verify that each module — authentication, file encryption, chunking, and node simulation — operated as expected. Data was collected by recording the success and failure rates of different operations. This helped measure the system's correctness and efficiency in handling user requests.

IV. SYSTEM ARCHITECTURE AND DESIGN

The system architecture of the decentralized cloud storage model is structured into three main layers: frontend, backend, and database, ensuring modularity, scalability, and security. The frontend, developed using React, HTML, and CSS, provides an intuitive interface for users to register, log in, upload, and download files. The backend, implemented using Python Flask, manages the system's logic and operations. It performs essential tasks such as file encryption, chunking, and node management.



Functionalities

1. User Authentication Module

- Provides secure user registration and login using **Firebase Authentication**.
- Users must create an account with a valid email and password.
- Implements **session management**, ensuring that only authenticated users can access storage.
- Prevents unauthorized access and protects user data privacy.

2. File Upload Module

- Enables users to upload files from the local system to the decentralized network.
- Before uploading, the file is **split into multiple chunks** (e.g., 3–5 parts).
- Each chunk is **encrypted** using a Python-based encryption algorithm for data security.
- The encrypted chunks are stored in **different simulated nodes**.

3. File Download Module

- Allows users to download files they have uploaded previously.
- The system fetches all corresponding file chunks from the nodes.
- Chunks are **decrypted** and **reconstructed** into the original file.

- File integrity is verified using the previously stored hash value.

4. File Encryption and Security Module

- Provides **data encryption** before storage and **decryption** before retrieval.
- The attacker cannot reconstruct the file without all chunks and the decryption key.
- Prevents unauthorized access and tampering.

5. Node Management Module

- Manages distributed storage nodes (Node1, Node2, Node3).
- Monitors availability and ensures redundancy.
- Handles fault tolerance if one node fails, files can still be reconstructed other nodes.
- Simulates decentralized storage by maintaining separate node directories.

V. IMPLEMENTATION AND INTEGRATION

The implementation of the Decentralized Cloud Storage System is carried out using modern web and backend technologies to ensure security, scalability, and modularity.

1. Frontend Implementation

The **frontend** is built using **HTML, CSS, and React.js** to provide an interactive and user-friendly interface.

- React components are used to create pages for user authentication, file upload, and file retrieval.
- Firebase Authentication SDK handles user login and signup securely.
- Axios library is used to send API requests from React to the Flask backend.

2. Backend Implementation

The **backend** is developed using **Python Flask**.

- It handles core logic such as file encryption, chunking, and storage across simulated nodes.
- When a file is uploaded, Flask splits it into multiple encrypted parts and distributes them among nodes (node1, node2, node3).
- Metadata like filename, hash, and node references are stored in **Firebase Fire store**.
- During download, Flask retrieves all chunks, decrypts, and merges them to reconstruct the original file.

3. Database Integration

Firebase Firestore serves as the backend database.

- It stores user data, file metadata, and node mapping.
- Firebase ensures secure data access and easy integration with both React and Flask via SDKs and REST APIs.

VI. EVALUATION AND RESULTS

The evaluation of the Decentralized Cloud Storage System was carried out to assess its performance, reliability, security, and user experience**.** The system was tested under different scenarios, including file uploads, downloads, and simulated node failures, to ensure its stability and effectiveness.

1. Functional Evaluation

The system successfully performed all the core functions:

- **User Authentication:** Users were able to register, log in, and log out securely through Firebase Authentication.
- **File Upload:** Files were uploaded, split into encrypted chunks, and distributed across multiple nodes without errors.
- **File Download:** The system successfully retrieved file chunks, decrypted them, and reconstructed the original file.
- **Metadata Management:** Firebase stored metadata such as filename, user ID, hash value.

All operations were tested through the React interface and verified using the backend logs.

2. Performance Testing
The system was tested for file handling efficiency and response time. Small to medium files (100 KB to 10 MB) were used during testing.

Test Case	File Size	Upload Time(sec)	Download Time
File 1	500 KB	1.2	1.0
File 2	2 MB	3.6	3.2
File 3	10 MB	8.4	7.9

The performance remained stable for files up to 10 MB. Beyond this size, network and hardware limitations slightly increased processing time. However, the decentralized structure still ensured consistent results and data availability.

3. Reliability Testing
Reliability was tested by temporarily disabling one of the storage nodes during the download process.
Even with one node offline, the system was able to retrieve remaining chunks from available nodes and notify the user of missing data if any. This demonstrates fault tolerance — a key advantage of decentralized architecture.

VII. CONCLUSION

Summary of the Work
The Decentralized Cloud Storage System was developed to provide a secure and efficient alternative to traditional centralized cloud storage platforms. The system splits uploaded files into encrypted chunks and distributes them across multiple storage nodes, ensuring data availability and fault tolerance. The combination of React (frontend), Python Flask (backend), and Firebase (database and authentication) enabled seamless integration and performance.

Key Findings
- The project demonstrates that decentralization enhances data security by removing single points of failure.
- File encryption ensures confidentiality, while hash verification guarantees data integrity.
- Firebase integration simplifies authentication and metadata management.
- The system's modular design allows scalability and easy maintenance.

Future Enhancements
Future development can include the following improvements:
- Blockchain Integration: To provide transparent storage validation and token-based incentives.
- IPFS Integration: To implement true decentralized file storage across a global peer network.
- Advanced Encryption Standards (AES): For stronger file protection.
- Cloud Deployment: Hosting the system on a real cloud environment for scalability testing.
- User Role Management: Implementing admin/user roles for better access control.

References

1. Benet, J. (2014). IPFS - Content Addressed, Versioned, P2P File System. arXiv preprint arXiv:1407.3561.[Online]. Available: https://ipfs.io
2. Wilkinson, S., Boshevski, T., Brandoff, J., & Buterin, V. (2014). Storj: A Peer-to-Peer Cloud Storage Network. [White Paper]. Storj Labs.[Online]. Available: https://storj.io
3. Protocol Labs. (2020). Filecoin: A Decentralized Storage Network. [White Paper].[Online]. Available: https://filecoin.io
4. Google Firebase. (2023). Firebase Documentation. [Online].
Available: https://firebase.google.com/docs
5. Pal, R., & Kumar, A. (2021). A Study on Decentralized Cloud Storage Using Blockchain Technology. International Journal of Computer Science and Engineering, Vol. 9, Issue 5, pp. 45–51.