IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

SECURE FILE SHARING SYSTEM

Using blockchain and IPFS

[1]Mrs. CH. D. V. P. Kumari, [2]D. K. K. Devika, [3]V. Hari Bhuvana,
[4]R. Praneeth Varma, [5]T. Hema Kiran

Assistant Professor of CSE(AI&ML), Aditya College of Engineering and Technology(A), Surampalem, A.P, India, 533437.
UG Scholar, Dept of CSE(AI&ML), Aditya College of Engineering and Technology(A), Surampalem, A.P, India, 533437.
UG Scholar, Dept of CSE(AI&ML), Aditya College of Engineering and Technology(A), Surampalem, A.P, India, 533437.
UG Scholar, Dept of CSE(AI&ML), Aditya College of Engineering and Technology(A), Surampalem, A.P, India, 533437.
UG Scholar, Dept of CSE(AI&ML), Aditya College of Engineering and Technology(A), Surampalem, A.P, India, 533437.

Abstract: In today's digital era, data security has become a crucial concern as the risk of unauthorized access and data breaches continues to rise. The Secure File Sharing System provides a reliable platform that ensures the safe transmission and storage of files through strong encryption and controlled access mechanisms. Files are encrypted using advanced cryptographic algorithms such as AES and RSA before being uploaded, ensuring confidentiality and integrity. Authorized users can securely retrieve files using OTP or key based authentication, while access logs and expiry-based links prevent misuse and maintain accountability. This system demonstrates an effective approach to protecting sensitive information and promoting secure data exchange in cloud-based environments. By integrating strong authentication mechanisms and secure communication protocols, the system maintains data integrity and user privacy throughout the sharing process. Overall, this project demonstrates a practical and robust approach to secure file exchange, addressing key cybersecurity challenges such as data leakage, unauthorized sharing, and identity verification. It aims to build user trust by offering a transparent, safe, and user-friendly environment for sharing confidential information across digital networks.

Index Terms – Blockchain, IPFS, AES-256 Encryption, Decentralized File Sharing, Flask Framework, Secure Cloud Storage.

I. INTRODUCTION

The modern digital workspace is effectively tethered to cloud platforms like Google Drive and OneDrive. While these services have undeniably streamlined global collaboration, they operate on a fundamentally flawed architecture of centralized trust. This model forces users to "blindly hand over" their most sensitive data—from proprietary research to corporate secrets—to a handful of massive tech entities. The danger here is the "single point of failure." If a provider's central hub is hit by a breach or a server goes dark, the user's privacy and access vanish instantly. We are currently operating in a landscape where data security is a promise made by a corporation rather than a guarantee provided by the architecture itself.

Our project was born from the need to reclaim data sovereignty. We've engineered a system that stops asking users to "trust" a company and starts letting them trust mathematical protocols. By merging Blockchain with IPFS, we've moved entirely away from the risky "all-eggs-in-one-basket" storage method. In our framework, a file isn't just "uploaded." Before it ever leaves the user's local hardware, it is locked behind AES-256 encryption. This ensures that the network nodes—the very machines storing the data—never actually see the content. They are holding encrypted "noise," while the actual keys remain solely with the user. This shifts the balance of power from the service provider back to the data owner.

The real innovation here lies in how we handle data location. Traditional web protocols (HTTP) look for a file based on *where* it is (a specific URL). Our system uses the InterPlanetary File System (IPFS) to find files based on *what* they are. Each file fragment is assigned a unique Content Identifier (CID). This CID acts as a cryptographic fingerprint. Because the file is broken into pieces and scattered across a peer-to-peer network, it becomes virtually impossible to censor or lose. Even if a third of the network goes offline, the fragments are still retrievable from other peers. This is a level of fault tolerance that a centralized server simply cannot match.

While IPFS handles the heavy lifting of storage, the Blockchain serves as our immutable notary. Every single action—every upload, share, or download—is etched into a SHA-256 blockchain ledger. We don't store the files on the chain, as that would be slow and expensive. Instead, we record the "metadata": who owns the file, who is allowed to see it, and what its CID is. This creates a transparent, tamper-proof audit trail. If someone tries to modify a file or forge an access request, the hash won't match the ledger, and the system will instantly reject the interaction. It is a self-policing ecosystem where the math does the auditing.

II. THEORETICAL BACKGROUND

2.1 Blockchain Technology

In this architecture, the Blockchain serves as the system's "source of truth." Rather than storing bulky file data (which would lead to network congestion and high costs), the blockchain acts as a lightweight, distributed notary. Each time a file is shared, the transaction metadata—including the owner's public key and the file's unique hash—is committed to a chronological chain. Because each block is cryptographically linked to its predecessor, altering a historical record is computationally unfeasible. This provides a permanent, transparent audit trail essential for high-stakes collaborative environments.

2.2 InterPlanetary File System (IPFS)

Unlike the traditional web (HTTP), which locates a file based on its "address" or URL, the InterPlanetary File System (IPFS) identifies a file by its **Content Identifier (CID)**. This CID is a cryptographic fingerprint generated via SHA-256. When a file is uploaded, it is broken into smaller blocks and distributed across multiple peers. This "content-addressing" ensures that if even a single byte of the file is altered, the CID changes entirely. Consequently, when a user requests a file, the system pulls it from the nearest available nodes, ensuring high speed and absolute data integrity.

2.3 AES-256 Encryption Algorithm

The primary layer of data confidentiality is enforced through AES-256, a symmetric-key block cipher that utilizes a 256-bit key to execute 14 rigorous rounds of mathematical transformations. These rounds include byte substitution, row shifting, and column mixing, ensuring that files are converted into an impenetrable "ciphertext" before ever leaving the user's local hardware. Because the encryption process is decentralized at the edge, the network nodes facilitate storage without ever possessing the decryption keys. This architecture effectively eliminates the risk of bulk data exposure typically associated with centralized breaches, as the stored information remains computationally unfeasible to crack through brute-force methods.

2.4 SHA-256 Hashing Algorithm

While AES-256 shields the content, SHA-256 serves as a one-way cryptographic "digital seal" to guarantee absolute data integrity. This algorithm condenses data of any size into a unique, fixed-length 64-character hexadecimal fingerprint, characterized by the "avalanche effect"—where even a single-bit alteration in the source file results in a radically different hash output. In our framework, SHA-256 generates the Content Identifier (CID) for IPFS retrieval and links transactions on the blockchain ledger. Upon retrieval, the system re-hashes the file and compares it against the immutable blockchain record; any mismatch instantly flags a "man-in-the-middle" intervention, ensuring the file remains untouched.

III. REQUIREMENT ANALYSIS

3.1 Functional Requirements

The system is engineered to facilitate a seamless yet highly secure end-to-end file-sharing workflow. Core capabilities include mandatory client-side AES-256 encryption, ensuring that no raw data ever touches the network. We've integrated the IPFS protocol to handle decentralized storage and retrieval, alongside an automated mechanism for generating and committing Content Identifiers (CIDs) to the blockchain. To govern data access, the framework employs smart-contract logic for role-based permissions, allowing only authorized recipients to initiate the decryption and download process. Finally, a real-time monitoring module tracks the health and status of individual network nodes to ensure persistent availability.

3.2 Non-Functional Requirements

Beyond basic operations, we prioritized the user experience through a highly responsive Flask-based web interface. Reliability is anchored in IPFS data replication and the inherent immutability of the Blockchain ledger. To handle high traffic, we optimized the backend for concurrent operations, ensuring performance doesn't degrade during multiple simultaneous uploads. Most importantly, the architecture is strictly hardened against common attack vectors—such as unauthorized access attempts and data tampering—while maintaining a modular design that allows for straightforward future scalability.

IV. SYSTEM ANALYSIS

4.1 Introduction

Before moving into the development phase, we conducted a comprehensive system analysis to map out every user interaction. This allowed us to align our functional flows with our primary objectives: ironclad security and intuitive usability. By identifying potential bottlenecks early, we ensured the final design would remain efficient under real-world conditions.

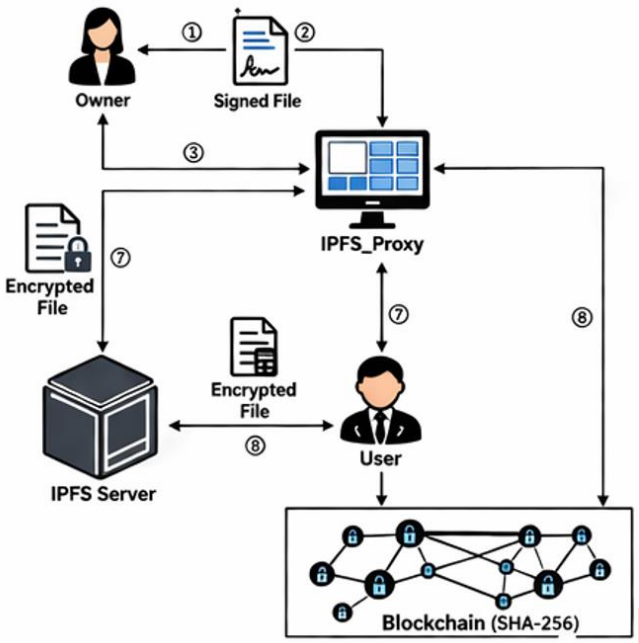
4.2 Use Cases

The system identifies two primary human actors: the **File Owner**, responsible for the initial encryption and distribution, and the **File User**, who retrieves and decrypts the shared assets. These actors interact with three critical subsystems: the **Main Coordination Server**, the **IPFS Storage Network**, and the **Blockchain Ledger**. Key use cases involve a rigorous authentication sequence, secure file discovery through metadata searches, and the continuous logging of every transaction to maintain a transparent, tamper-proof audit trail.

V. SYSTEM DESIGN

5.1 System Architecture

We adopted a multi-layered architectural approach to decouple the interface from the security logic. The **Client Layer** provides the browser-based interaction, while the **Application Server** (built on Flask with WebSocket support) manages the heavy lifting of backend coordination. Data itself resides in the **IPFS Storage Layer**, while the **Blockchain Layer** handles the decentralized integrity checks. When



a user uploads a file, the system triggers a chain reaction: local AES-256 encryption, followed immediately by IPFS fragment distribution and the immutable recording of the transaction on the ledger.

Fig 5.1.1 System Architecture

5.2 Object Model and Dynamic Model

To ensure structural integrity, our class diagrams define the exact relationships between Users, Files, and Blockchain entities. We utilized sequence and activity diagrams to trace the precise, step-by-step execution of the upload and download cycles. This modelling was essential for identifying decision points—such as authentication failures or node timeouts—to ensure the system handles errors gracefully and efficiently.

VI. IMPLEMENTATION

The prototype was developed using **Python 3.8+**, leveraging the **Flask** framework for its lightweight and modular nature. We incorporated **Flask-SocketIO** to enable real-time, bidirectional communication between the server and client. The cryptographic heavy lifting is performed by the pyAesCrypt library, while IPFS connectivity is managed via the **Kubo daemon's HTTP API**. Operatively, the primary server listens on port 5111, with client nodes utilizing sequential ports starting at 5113. This setup automates the entire CID generation and blockchain logging process, providing a fluid experience that masks the complex cryptography happening in the background.

VII. CONCLUSION

This research validates that a decentralized approach can effectively resolve the inherent security flaws found in traditional, centralized cloud platforms. By merging **AES-256 encryption** with **IPFS** and **Blockchain**, we've created a resilient environment where data integrity and user privacy are prioritized by design. The Flask-based implementation proves that military-grade security does not have to come at the cost of usability. As we move forward, this modular architecture provides a foundation for even more advanced features, such as multi-node scalability and complex access policies, making it a viable tool for any institution that demands trustworthy digital collaboration.

REFERENCES

[1] Benet, J. (2014). IPFS - Content Addressed, Versioned, P2P File System. arXiv preprint arXiv:1407.3561.
[2] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System.
[3] NIST. (2001). Advanced Encryption Standard (AES). FIPS PUB 197.