Blockchain and Split Key Based Secure File Storage for IPFS

Arshida Febin T P
Department of Computer Science and Engineering
Government Engineering College Wayanad
APJ Abdul Kalam Technological University
arshidafebin33@gmail.com

Dhanya Raj P
Department of Computer Science and Engineering
Government Engineering College Wayanad
APJ Abdul Kalam Technological University
dhanyaraj@gecwyd.ac.in

***Abstract*—Storing large files directly on blockchain networks is inefficient due to high storage cost, replication overhead, and scalability limitations. To overcome these challenges, this paper proposes a secure decentralized file storage framework using Ethereum blockchain and InterPlanetary File System (IPFS). In the proposed system, files are stored off-chain in IPFS, while only the generated Content Identifier (CID) is stored on the blockchain to reduce storage requirements and improve performance. Smart contracts are used to implement role-based access control for secure and transparent file sharing. Initially, a basic access-controlled storage mechanism was developed to allow authorized users to retrieve encrypted healthcare records while preventing unauthorized access. During implementation, a security issue related to centralized encryption key leakage was identified. To address this limitation, the system was enhanced using distributed key management, where the encryption key is divided into multiple fragments and stored across the blockchain, backend server, and local machine. The original decryption key can only be reconstructed when all fragments are combined, thereby improving confidentiality and resistance against attacks.**

***Index Terms*—Blockchain, InterPlanetary File System(IPFS), Distributed storage, Content-addressed technique**

I. Introduction

The rapid growth of digital technologies has significantly increased the amount of sensitive data generated, stored, and shared across modern systems. Traditional centralized storage architectures are widely used for managing digital records because they provide easy accessibility and maintenance. However, centralized systems suffer from several limitations, including single points of failure, unauthorized access, data tampering, privacy breaches, and dependency on third-party servers. These issues create serious concerns regarding data confidentiality, integrity, and secure sharing. Blockchain technology has emerged as a promising solution for secure and decentralized data management. Blockchain provides transparency, immutability, traceability, and tamper resistance through distributed ledger mechanisms. Smart contracts further enhance blockchain functionality by enabling automated and secure access control policies. Using smart contracts, only authorized users can access protected information, thereby improving trust and security in decentralized environments. Despite these advantages, storing large files directly on blockchain networks is inefficient because blockchain storage is expensive, limited in capacity, and increases network overhead. Since every node maintains a copy of blockchain data, direct storage of files leads to scalability and performance issues. To overcome this limitation, decentralized storage systems are integrated with blockchain technology. In such systems, files are stored off-chain while only the file reference or cryptographic hash is stored on-chain. This approach reduces storage costs and improves scalability while preserving decentralization.

In the initial phase of this project, a smart contract-based access control mechanism was implemented for secure file storage and retrieval. Encrypted files were uploaded to a decentralized storage system, and access permissions were managed using blockchain-based authorization. The system successfully demonstrated access-granted and access-denied functionalities for different users based on assigned roles. However, during experimentation, a major security issue related to centralized encryption key leakage was identified. If the encryption key becomes compromised, unauthorized users may gain access to confidential information despite the existence of access control mechanisms.

To address this problem, the proposed system was enhanced using a distributed key management approach. Instead of storing the complete encryption key in a single location, the key was divided into multiple fragments and distributed across different storage locations such as blockchain, backend server, and local machine. During authorized access, all fragments are combined to reconstruct the original key required for decryption. This approach eliminates single-point key compromise and significantly strengthens system security. The complete framework was implemented using smart contracts, decentralized storage, encryption algorithms, backend services, and wallet-based authentication mechanisms. Experimental results demonstrate successful contract deployment, encrypted file upload, decentralized storage, secure key reconstruction, authorized data retrieval, and effective prevention of unauthorized access attempts. The proposed system improves scalability, confidentiality, integrity, and secure decentralized data sharing for modern distributed applications.

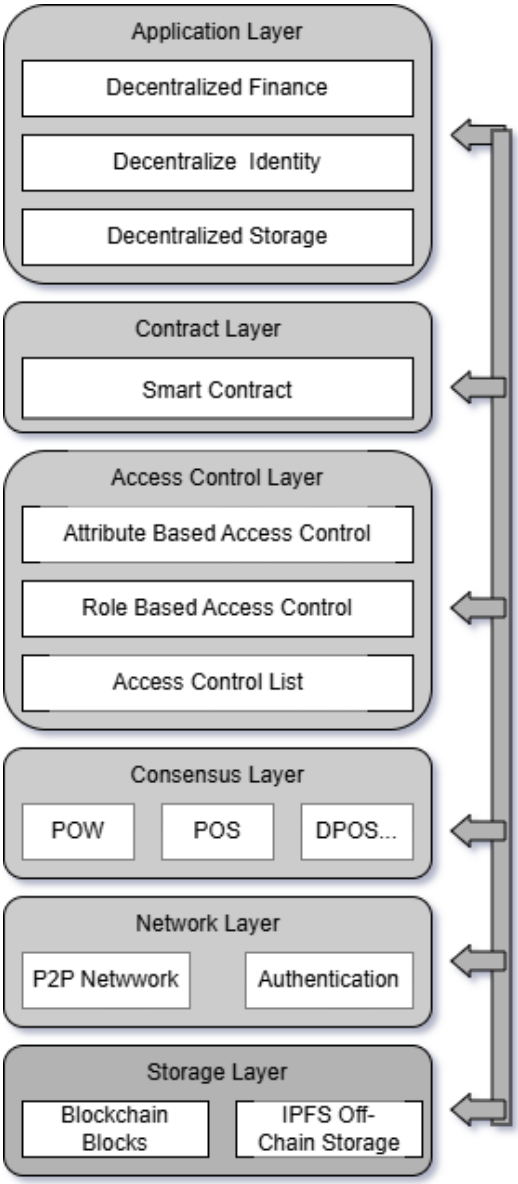


Fig. 1. Layered architecture of blockchain–IPFS based decentralized data storage and access control system

II. BACKGROUND

A. Blockchain Technology

Blockchain is a distributed ledger where transactions are recorded in an immutable chain of blocks. Its properties—decentralization, transparency, tamper resistance, and auditability—make it suitable for trustless access control. Smart contracts enable automated rule enforcement for granting, revoking, and verifying permissions.

Blockchain is a distributed ledger technology in which data is organized into a sequence of cryptographically linked blocks, ensuring immutability and tamper resistance. Its foundational structure is supported by the storage layer, which maintains data blocks, chain structures, timestamps, and hash functions to guarantee integrity and verifiability. Above this, the network layer enables decentralized communication through peer-to-peer (P2P) networking, message propagation, and authentication mechanisms, allowing nodes to share and validate information without a central authority. The consensus layer provides the mechanism through which nodes collectively agree on the state of the ledger, using algorithms such as Proof of Work (PoW), Proof of Stake (PoS), and Delegated Proof of Stake (DPoS) to prevent unauthorized modifications and ensure consistent global state. On top of these foundational modules lies the execution layer, which processes transactions and verifies digital signatures, ensuring that only valid operations are committed to the blockchain. The contract layer introduces programmable logic through smart contracts—self-executing code that automates rules for granting, revoking, and verifying access permissions without relying on intermediaries. At the highest level, the application layer supports decentralized applications (dApps) that enable functionalities such as programmable money, decentralized finance, and broader societal applications. Collectively, these layers form a transparent, secure, and trustless architecture, making blockchain an ideal platform for decentralized access control and immutable auditability in modern distributed systems.

The architecture shown in Fig. 1 illustrates a layered framework integrating blockchain technology with the InterPlanetary File System (IPFS) and multiple access control mechanisms. The architecture is organized into six functional layers: Storage Layer, Network Layer, Consensus Layer, Access Control Layer, Contract Layer, and Application Layer. Each layer performs specific operations to ensure secure data storage, transaction validation, and controlled access to decentralized resources.

1) Storage Layer: The storage layer forms the core of the architecture, integrating on-chain blockchain storage with off-chain IPFS-based distributed storage. Blockchain maintains immutable transaction records, metadata, and content identifiers (CIDs), while avoiding direct storage of large files due to scalability and cost constraints. IPFS employs a content-addressable storage mechanism using cryptographic hashing and Merkle Directed Acyclic Graphs (Merkle DAGs) for efficient data distribution. Each uploaded file is chunked, hashed, and assigned a unique CID derived from its content. This CID is stored on the blockchain, enabling secure linkage between on-chain metadata and off-chain data. Data integrity is ensured by verifying the hash of retrieved content against the recorded CID. Any modification results in a hash mismatch, providing tamper-evidence. This hybrid storage model improves scalability, reduces blockchain overhead, and ensures efficient and secure data retrieval in a decentralized environment.

2) Network Layer: The network layer facilitates decentralized communication through a peer-to-peer (P2P) overlay network, enabling efficient propagation of transactions and blocks among distributed nodes. It employs gossip-based dissemination protocols to ensure low-latency and fault-tolerant message broadcasting. Node discovery and topology man-

agement are achieved through distributed mechanisms such as Kademlia-based Distributed Hash Tables (DHTs). Secure communication is enforced using cryptographic authentication protocols, including public key infrastructure (PKI) and digital signatures, to verify node identities and prevent Sybil or spoofing attacks. The layer also supports secure transport protocols (e.g., TLS/SSL) for encrypted data exchange. Synchronization mechanisms ensure ledger consistency by maintaining eventual consistency across nodes through block validation and state updates. Additionally, it handles network resilience, mitigating partitioning and DoS attacks through redundancy and adaptive routing strategies.

3) Consensus Layer: The consensus layer ensures that all nodes in the distributed network agree on the state of the blockchain ledger. Since blockchain operates without a central authority, consensus algorithms are required to validate transactions and determine which block should be added to the chain.

The architecture shows three common consensus mechanisms:

a) Proof of Work (PoW) – Nodes compete to solve a computationally intensive cryptographic puzzle by iteratively computing hash values (e.g., SHA-256) to find a nonce that satisfies a predefined difficulty target. The first node to discover a valid solution gains the right to append the block to the blockchain and broadcast it to the network, ensuring security through computational effort and resistance to Sybil attacks.:

b) Proof of Stake (PoS) – Validators are probabilistically selected to propose and validate new blocks based on their stake (i.e., the amount of cryptocurrency locked in the system), often combined with additional factors such as staking duration or randomness. This mechanism eliminates energy-intensive computations, enhances scalability, and secures the network by economically incentivizing honest behavior and penalizing malicious actions through stake slashing.:

c) Delegated Proof of Stake (DPoS): Token holders elect a limited set of delegates (block producers) through a voting mechanism to propose and validate blocks on behalf of the network. Consensus is achieved via a deterministic scheduling of delegates, enabling high throughput and low latency. The system relies on reputation and continuous voting, where misbehaving or inactive delegates can be replaced, ensuring accountability and efficient block production..

These mechanisms ensure fault tolerance, prevent double spending, and maintain the integrity of the blockchain network.

4) Access Control Layer: The access control layer is responsible for managing permissions and restricting access to stored data and services. This layer integrates multiple access control models to ensure that only authorized users can interact with system resources.

The architecture includes three major access control mechanisms:

a) Attribute-Based Access Control (ABAC): Access decisions are determined through the evaluation of policies defined over attributes associated with subjects, objects, and environmental contexts. Policies are expressed as logical rules that enforce fine-grained and dynamic authorization by combining attribute values, enabling context-aware access control without reliance on predefined roles or identities.

b) Role-Based Access Control (RBAC) : Permissions are assigned to roles rather than individual users, and users are mapped to roles based on organizational policies. Access decisions are enforced through role–permission associations, enabling structured and scalable authorization while supporting constraints such as role hierarchies and separation of duties.

c) Access Control Lists (ACLs): ACLs define access permissions through explicit lists associated with each resource, specifying authorized subjects (users or groups) and their permitted operations (e.g., read, write, execute). Access decisions are enforced by matching the requesting entity against the ACL entries, enabling fine-grained, object-centric authorization control.

This layer plays a crucial role in protecting sensitive data stored in IPFS and referenced by blockchain transactions.

5) Contract Layer: The contract layer manages the execution of smart contracts, which are self-executing programs stored on the blockchain. Smart contracts define the rules governing transactions and interactions between participants.

In this architecture, smart contracts may be used to enforce access control policies, verify user permissions, manage file references stored in IPFS, and automate transactions between parties. By embedding logic directly in the blockchain, smart contracts eliminate the need for intermediaries and improve system transparency and trust.

6) Application Layer: The application layer represents the top layer where end-user applications interact with the blockchain system. This layer includes decentralized applications (DApps) that provide services to users.

Examples of applications shown in the architecture include:

a) Decentralized Finance (DeFi): Blockchain-based financial systems enabling peer-to-peer transactions without intermediaries.

b) Decentralized Identity (DID): Identity management systems that give users control over their digital identities.

c) Decentralized Storage : Systems that use blockchain and IPFS to store and share data securely in a distributed environment.

The application layer provides the user interface and business logic necessary to interact with the underlying blockchain infrastructure.

B. InterPlanetary File System (IPFS)

The InterPlanetary File System (IPFS) is a peer-to-peer (P2P) distributed file storage and retrieval protocol designed to overcome the limitations of traditional location-based web architectures. Unlike conventional systems where data is retrieved from a central server using URLs, IPFS operates on a content-addressed model, meaning that every file is identified by a unique cryptographic hash known as a Content

Identifier (CID). This ensures that the file content cannot be modified without changing its CID, thereby guaranteeing data integrity and tamper detection.IPFS stores data as a Merkle Directed Acyclic Graph (Merkle-DAG), where each file is broken into chunks, hashed, and linked cryptographically. This makes IPFS inherently versioned, deduplicated, and verifiable. When a user uploads a file, the data is distributed across multiple nodes in the IPFS network, each capable of serving the content. Retrieval is performed by requesting the CID, allowing the nearest or most efficient node holding the data to respond. This results in faster access, reduced network congestion, and improved reliability compared to centralized infrastructures.Another key feature of IPFS is its decentralized resilience: even if one or more nodes go offline, the file remains accessible as long as at least one node continues to store ("pin") it. This eliminates single points of failure and enhances data availability. Furthermore, IPFS can integrate seamlessly with blockchain systems by storing large files off-chain while keeping only their CIDs, metadata, and access rules on-chain. This hybrid model enables scalable storage, cryptographic integrity, and cost-efficient distributed file management, making IPFS especially suitable for decentralized applications, secure data sharing, and access control systems.

III. Related Work

Several researchers have explored the integration of blockchain and decentralized storage technologies to provide secure, scalable, and privacy-preserving data sharing systems. Steichen *et al.* [1] proposed an access-controlled IPFS framework using Ethereum smart contracts to manage secure file sharing in decentralized environments. Their work demonstrated how smart contracts can enforce access permissions for distributed storage systems. Similarly, Kumar and Tripathi [2] implemented a distributed file storage and access framework using blockchain and IPFS, where files were stored off-chain while blockchain maintained file references to improve scalability and reduce storage overhead. Zhang and Zhang [3] introduced a cryptographic blockchain-IPFS framework for secure distributed database storage and access control. Their work focused on combining decentralized storage with cryptographic mechanisms to ensure confidentiality and integrity of stored data. Jayapriya and Jeyanthi [4] proposed a scalable blockchain architecture using off-chain IPFS storage to improve security and privacy in distributed data management systems. Their approach reduced blockchain storage consumption while improving overall system scalability. Several studies have also concentrated on blockchain-based access control mechanisms. Azbeg *et al.* [5] developed a privacy-preserving blockchain-based access control system that enables secure and transparent data sharing. Wang *et al.* [6] proposed a secure cloud storage framework with blockchain-based access control policies to ensure confidentiality and integrity of shared data. These approaches improved decentralized authorization management but still relied on centralized encryption key handling mechanisms. Saviour and Samiappan [7] designed an IPFS-based file storage and authentication model using blockchain techniques for secure data transfer. In another work, Saviour and Samiappan [8] enhanced the framework with optimization-enabled intrusion detection mechanisms to strengthen security during decentralized storage operations. Battah *et al.* [9] proposed a blockchain-based multi-party authorization system for accessing encrypted IPFS data, where multiple users participate in the authorization process before data retrieval is permitted. Encryption-based access control mechanisms have also been widely investigated. Sun *et al.* [10] integrated Ciphertext-Policy Attribute-Based Encryption (CP-ABE) with IPFS for secure file sharing. Chase [11], Bethencourt *et al.* [12], and Lewko and Waters [13] introduced attribute-based encryption mechanisms that provide fine-grained access control for encrypted data. Although these approaches improve confidentiality, they often introduce computational complexity and complex key management challenges. Zyskind *et al.* [14] discussed decentralized privacy-preserving systems using blockchain to protect personal data. Ekblaw *et al.* [15] developed MedRec, a blockchain-based framework for secure medical data management and auditing. Xia *et al.* [16] proposed blockchain and smart contract-based access control frameworks for secure data sharing. Liu *et al.* [17] further explored smart contract mechanisms for enforcing secure sharing agreements in decentralized systems. Lin *et al.* [18] introduced blockchain-based mutual authentication and fine-grained access control mechanisms to improve trust among distributed users. Recent works by Naz *et al.* [19] demonstrated the integration of blockchain and IPFS for secure data exchange and monetization. These studies highlighted the advantages of decentralized storage systems but also identified challenges related to encryption key security and unauthorized access prevention. Although existing works successfully combine blockchain, IPFS, and encryption for decentralized storage and access control, many systems still suffer from centralized encryption key management vulnerabilities. If a single encryption key is compromised, attackers may gain unauthorized access to sensitive data. To overcome this limitation, the proposed work introduces a distributed key fragmentation mechanism where encryption keys are divided and stored across blockchain, backend server, and local machine environments. This approach enhances confidentiality, prevents single-point key leakage, and strengthens secure decentralized data sharing.

IV. System Architecture

Fig.1 illustrates the overall architecture of the proposed decentralized secure file storage and access control framework. The architecture integrates Ethereum blockchain, IPFS distributed storage, AES-256 encryption, Node.js applications, and smart contracts to provide secure, decentralized, and controlled file sharing.

The system consists of three major entities namely the users, decentralized storage network, and blockchain-based access control layer. Different users such as admin, doctor, and

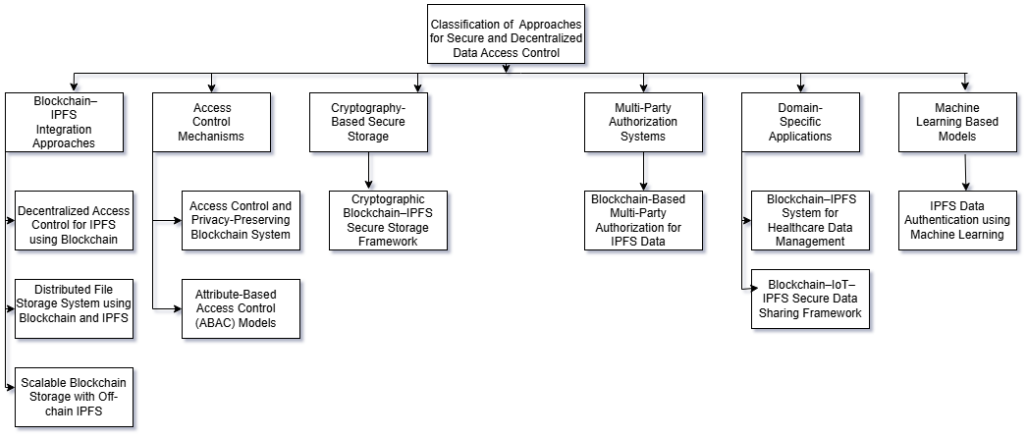


Fig. 2. Classification of Approaches for Secure and Decentralized Data Access Control

patient interact with the system through a Node.js application interface. The Node.js application acts as the middleware between the blockchain network, IPFS storage, and the encryption module. It performs major operations such as smart contract deployment, file upload, file retrieval, encryption, and decryption.

Before storing any file in the decentralized storage, the file is encrypted using the AES-256 encryption algorithm. The encryption module converts the original plaintext file into ciphertext format using a secret cryptographic key. This process ensures confidentiality of the stored content and prevents unauthorized disclosure of sensitive information. After encryption, the encrypted file is uploaded to the InterPlanetary File System (IPFS). IPFS stores the file in a distributed peer-to-peer network and generates a unique Content Identifier (CID) based on the cryptographic hash of the file content. The CID acts as a permanent reference for retrieving the encrypted file from the distributed storage network. Instead of storing the complete file on the blockchain, only the generated CID is stored on the Ethereum blockchain. This significantly reduces blockchain storage overhead and improves scalability.

The Ethereum Sepolia blockchain network is used for creating blockchain accounts and deploying smart contracts. The smart contract is responsible for managing user permissions and enforcing access control policies. It verifies whether a requesting user possesses valid authorization to access a particular file. During file retrieval, the user sends a request through the Node.js application. The smart contract checks the permission level associated with the user account. If the user is authorized, the CID is returned from the blockchain and the encrypted file is retrieved from IPFS. The AES decryption module then reconstructs the original file using the decryption key.

The architecture also supports secure permission management through grant and deny access functions implemented inside the smart contract. Unauthorized users attempting to access the file are denied permission, thereby ensuring secure and controlled decentralized file sharing. The proposed architecture combines the advantages of blockchain immutability, IPFS decentralized storage, and cryptographic security to provide a scalable and secure distributed storage framework.

V. Implementation and Results

The proposed system is implemented through a sequence of secure and decentralized processes involving blockchain, IPFS, smart contracts, encryption, and distributed key management. Each phase of the methodology contributes to ensuring confidentiality, integrity, decentralized storage, and secure access control.

1. Smart Contract Development and Deployment:

The first step of the proposed work involves the development of Ethereum smart contracts using the Solidity programming language. The smart contract is responsible for maintaining access permissions and role-based authorization within the decentralized network. Different blockchain accounts are assigned predefined roles such as owner, authorized user, and unauthorized user. The smart contract was deployed on the Ethereum Sepolia test network using the Hardhat framework. Deployment on the blockchain ensures immutability, transparency, and decentralized verification of access permissions. Once deployed, the smart contract stores permission data and verifies whether a requesting user is authorized to access a particular file.

2. Role Assignment and Access Control:

After successful deployment of the smart contract, blockchain addresses are assigned specific access permissions. The owner account grants permissions to selected users through smart contract functions. This mechanism establishes a role-based access control model where only authorized blockchain addresses can retrieve and decrypt stored files. Whenever a user attempts to access a file, the smart contract checks the permission level associated with the requesting wallet address. If the permission level is valid, access is granted; otherwise, the request is denied. This decentralized authorization mechanism eliminates centralized dependency and prevents unauthorized file retrieval.

3. File Encryption Using AES Algorithm:

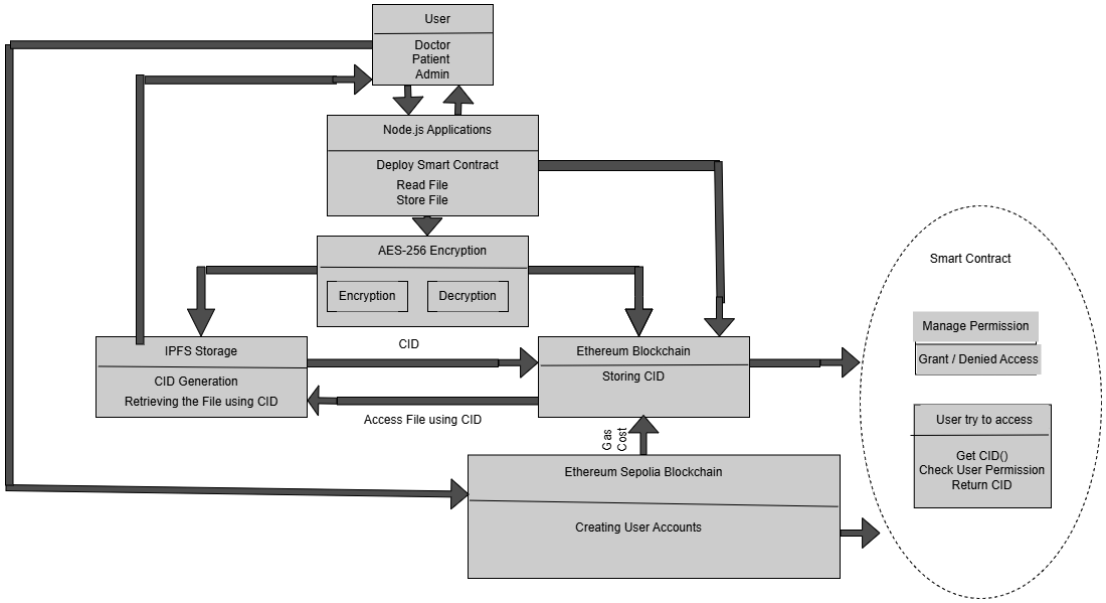


Fig. 3. Working Model of Blockchain and IPFS

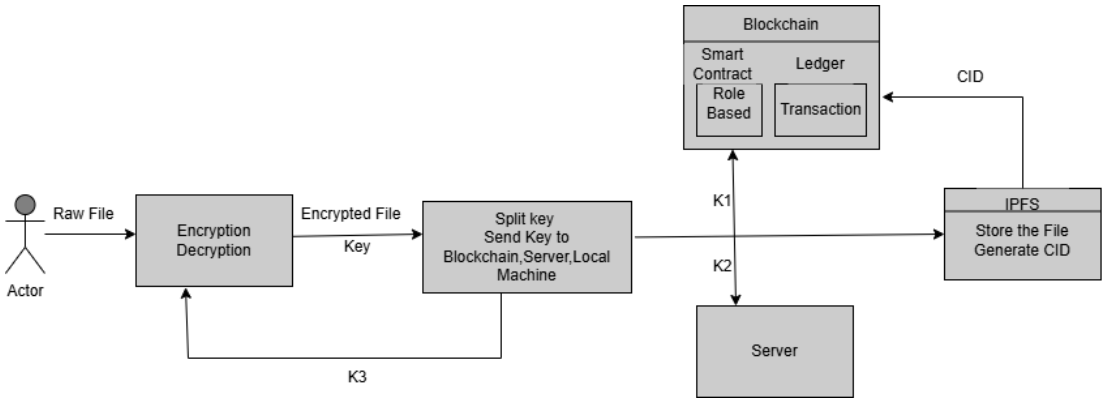


Fig. 4. End-to-End Upload Process Integrating AES Encryption, IPFS Storage, and Blockchain-Based Key Management

Before storing any file in the distributed storage network, the original file is encrypted using the AES-256 symmetric encryption algorithm. AES encryption converts the original plaintext data into ciphertext using a secure cryptographic key. This step ensures confidentiality of the stored content even if the encrypted file becomes publicly accessible through IPFS. The encrypted file cannot be interpreted without the correct decryption key. Encryption is performed locally before uploading the file to the distributed environment, thereby preventing exposure of sensitive data during transmission or storage.

4. Uploading Encrypted File to IPFS:

Once encryption is completed, the encrypted file is uploaded into the InterPlanetary File System (IPFS). IPFS is a decentralized peer-to-peer distributed storage network that stores files using content-addressing mechanisms instead of traditional location-based addressing. After uploading the encrypted file, IPFS generates a unique Content Identifier (CID) based on the cryptographic hash of the file content. This CID acts as a permanent reference for retrieving the encrypted file from the distributed network. Since only the encrypted file is uploaded to IPFS, data confidentiality is preserved while reducing blockchain storage overhead.

5. Storage of CID on Blockchain:

After successful generation of the CID from IPFS, the CID is stored on the Ethereum blockchain through the deployed smart contract. Instead of storing the complete file on-chain, only the lightweight CID value is recorded on the blockchain. This significantly reduces storage consumption and transaction cost while maintaining permanent traceability of the uploaded file. The blockchain acts as a decentralized metadata registry that securely maintains the mapping between authorized users and the corresponding file CID.

6. Initial Key Storage and Security Issue:

In the initial implementation of the proposed system, the AES encryption key was stored in a single storage location

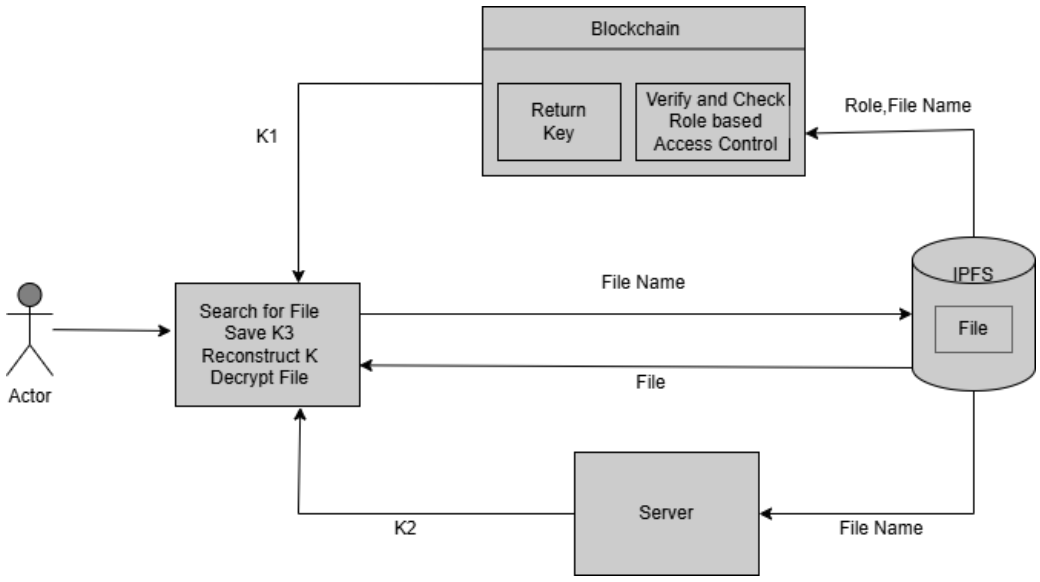
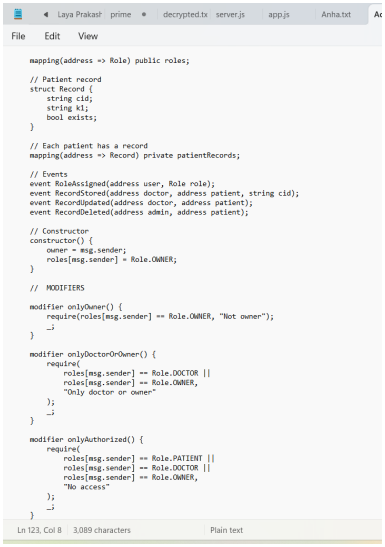
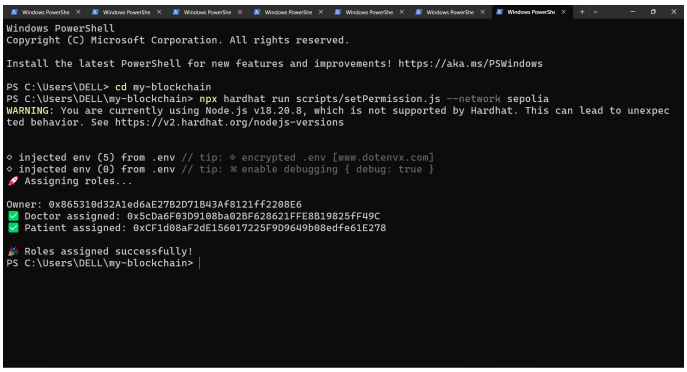


Fig. 5. Secure File Access and Decryption Process using Blockchain-Based Authorization and Distributed Key Reconstruction (K1, K2, K3)



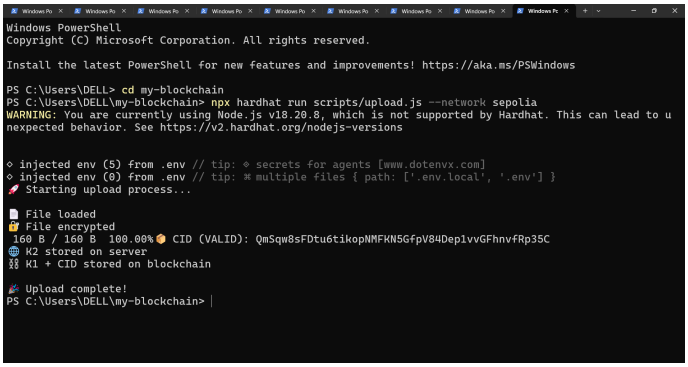
```
mapping(address => Role) public roles;

// Patient record
struct Record {
    string cid;
    string k1;
    bool exists;
}

// Each patient has a record
mapping(address => Record) private patientRecords;

// Events
event RoleAssigned(address user, Role role);
event RecordStored(address doctor, address patient, string cid);
event RecordUpdated(address doctor, address patient);
event RecordDeleted(address admin, address patient);

// Constructor
constructor() {
    owner = msg.sender;
    roles[msg.sender] = Role.OWNER;
}

//  MODIFIERS

modifier onlyOwner() {
    require(roles[msg.sender] == Role.OWNER, "Not owner");
    _;
}

modifier onlyDoctorOrOwner() {
    require(
        roles[msg.sender] == Role.DOCTOR ||
        roles[msg.sender] == Role.OWNER,
        "Only doctor or owner"
    );
    _;
}

modifier onlyAuthorized() {
    require(
        roles[msg.sender] == Role.PATIENT ||
        roles[msg.sender] == Role.DOCTOR ||
        roles[msg.sender] == Role.OWNER,
        "No access"
    );
    _;
}
```

Fig. 6. Smart contract implementation for role-based access control and secure record management in the Ethereum blockchain network.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\DELL> cd my-blockchain
PS C:\Users\DELL\my-blockchain> npx hardhat run scripts/setPermission.js --network sepolia
WARNING: You are currently using Node.js v18.20.8, which is not supported by Hardhat. This can lead to unexpec
ted behavior. See https://v2.hardhat.org/nodejs-versions


◇ injected env (5) from .env // tip: ⌘ encrypted .env [www.dotenvx.com]
◇ injected env (0) from .env // tip: ⌘ enable debugging { debug: true }
🚀 Assigning roles...

Owner: 0x865310d32A1ed6aE27B2D71B43Af8121ff2208E6
✅ Doctor assigned: 0x5cDa6F03D9108ba02BF628621FFE8B19825fF49C
✅ Patient assigned: 0xCF1d08aF2dE156017225F9D9649b08edfe61E278

🎉 Roles assigned successfully!
PS C:\Users\DELL\my-blockchain>
```

Fig. 7. Role Assignment and Permission Management using Smart Contract on the Ethereum Sepolia Network



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\DELL> cd my-blockchain
PS C:\Users\DELL\my-blockchain> npx hardhat run scripts/upload.js --network sepolia
WARNING: You are currently using Node.js v18.20.8, which is not supported by Hardhat. This can lead to u
nexpected behavior. See https://v2.hardhat.org/nodejs-versions


◇ injected env (5) from .env // tip: ⌘ secrets for agents [www.dotenvx.com]
◇ injected env (0) from .env // tip: ⌘ multiple files { path: ['.env.local', '.env'] }
🚀 Starting upload process...

📄 File loaded
🔐 File encrypted
 160 B / 160 B  100.00%📦 CID (VALID): QmSqw8sFDtu6tikopNMFKN5GfpV84Dep1vvGFhnvfRp35C
🌐 K2 stored on server
⛓️ K1 + CID stored on blockchain

🎉 Upload complete!
PS C:\Users\DELL\my-blockchain>
```

Fig. 8. Uploading Encrypted File to IPFS and Storing Distributed Key Fragments on Blockchain and Server

for future decryption. Although the smart contract successfully enforced access permissions, this approach introduced a security vulnerability known as key leakage. If an attacker obtained access to the stored encryption key, the encrypted file could be decrypted without authorization. This issue highlighted the limitations of centralized key storage and motivated the development of a more secure distributed key management mechanism.

7. Distributed Key Splitting Mechanism:

To overcome the key leakage problem, the proposed framework introduces a distributed key management strategy. The original AES encryption key is divided into three independent fragments namely K1, K2, and K3. Each fragment is stored in a separate storage location to avoid centralized exposure of the complete key. The first fragment (K1) is stored on the blockchain along with the CID, the second fragment (K2) is stored on the backend server, and the third fragment (K3) is stored locally on the authorized client machine. This distributed storage architecture ensures that compromising a

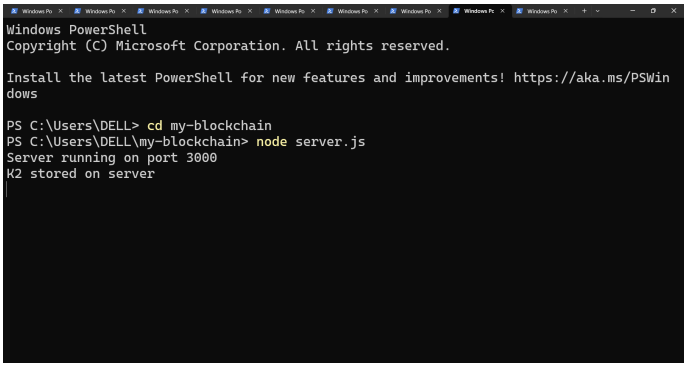


Fig. 9. Backend Server Initialization for Secure Storage and Management of Key Fragment K2

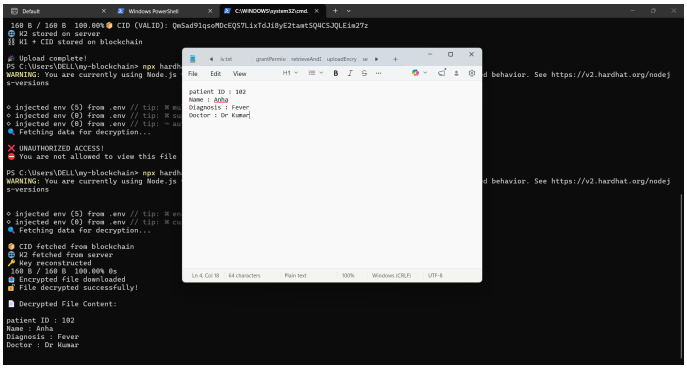


Fig. 10. Authorized File Retrieval and Successful Decryption using Distributed Key Reconstruction Mechanism

single storage location cannot reveal the complete encryption key.

8. Backend Server Initialization:

A dedicated backend server was implemented using Node.js to securely store and manage the K2 key fragment. The backend server runs independently and provides secure retrieval of the required key fragment during authorized decryption requests. The server acts as an intermediate trusted component within the framework while maintaining separation from blockchain and local storage components. The server initialization process also verifies secure connectivity between the decentralized storage network and the application layer.

9. Secure File Retrieval and Key Reconstruction:

During file retrieval, the requesting user initiates a decryption request through the blockchain application. The smart contract first verifies whether the requesting blockchain address possesses valid access permission. If the user is authorized, the encrypted file is downloaded from IPFS using the stored CID. Simultaneously, the three distributed key fragments are collected from the blockchain, backend server, and local machine. These fragments are then combined to reconstruct the original AES encryption key. The reconstructed key is finally used to decrypt the encrypted file and restore the original plaintext content.

10. Unauthorized Access Prevention:

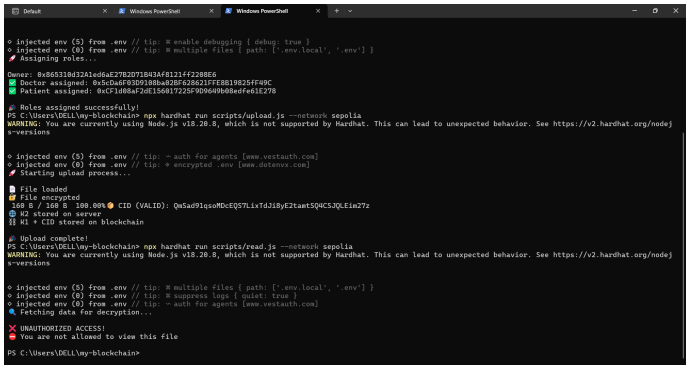


Fig. 11. Unauthorized Access Attempt Rejected by Smart Contract-based Access Control Mechanism

The proposed system also demonstrates effective prevention of unauthorized access attempts. If a user without valid blockchain permissions attempts to retrieve a file, the smart contract immediately rejects the request and prevents access to the decryption process. Since the complete encryption key is never stored in a single location, unauthorized users cannot reconstruct the AES key even if partial information is compromised. This mechanism strengthens overall system security and ensures confidentiality of sensitive stored data.

11. Experimental Validation:

The complete framework was experimentally implemented using Node.js, Solidity, Hardhat, MetaMask wallet, Ethereum Sepolia test network, and IPFS. Experimental outputs confirmed successful smart contract deployment, role assignment, AES encryption, CID generation, distributed key storage, secure file retrieval, authorized decryption, and unauthorized access denial. The obtained results validate that the proposed decentralized framework provides secure, scalable, and efficient file sharing with enhanced resistance against key leakage attacks.

VI. Conclusion

This paper presented a secure and decentralized file storage framework using Ethereum blockchain and IPFS. The proposed system successfully combines smart contract-based access control with distributed storage to achieve secure file sharing. Initially, the framework implemented role-based authorization where only permitted users could retrieve encrypted files from IPFS. However, the study identified the issue of key leakage due to centralized storage of encryption keys. To overcome this limitation, a distributed key management mechanism was introduced by splitting the encryption key into multiple fragments and storing them across blockchain, backend server, and local storage. The proposed enhancement significantly improved confidentiality and resistance against unauthorized access. Experimental results demonstrated successful smart contract deployment, CID generation, encrypted file upload, secure retrieval, authorized decryption, and prevention of unauthorized access attempts. The framework reduced blockchain storage overhead by storing only the CID on-chain while maintaining decentralized accessibility through IPFS.

The integration of AES encryption, blockchain immutability, and distributed key splitting provides a scalable and secure decentralized storage solution. Therefore, the proposed framework offers an efficient approach for secure decentralized file sharing with enhanced protection against key compromise attacks. This survey examined the emerging integration of blockchain and IPFS as a foundation for secure, decentralized access control. By analyzing existing architectures, models, and domain-specific frameworks, we highlighted how immutable ledgers, smart-contract–based enforcement, and content-addressed storage collectively enhance integrity, auditability, and resilience in distributed environments. At the same time, our review showed that major challenges—such as scalable policy updates, efficient revocation, multi-authority coordination, and ensuring data availability—remain open research problems. Overall, blockchain–IPFS systems demonstrate strong potential for next-generation secure data sharing, but further work is required to develop lightweight, interoperable, and production-ready access control solutions.

REFERENCES

[1] M. Steichen, B. Fiz, R. Norvill, W. Shbair, and R. State, "Access-controlled IPFS via smart contracts," in Proc. IEEE Cybermatics / IoT-Related Conferences, 2018.

[2] R. Kumar and R. Tripathi, "Implementation of distributed file storage and access framework using IPFS and blockchain," in Proc. 5th Int. Conf. Image Inf. Process. (ICIIP), 2019.

[3] F. Zhang and L. Zhang, "A cryptographic blockchain–IPFS framework for secure distributed database storage and access control," Informatica, vol. 49, pp. 159–176, 2025.

[4] J. Jayapriya and N. Jeyanthi, "Scalable blockchain model using off-chain IPFS storage for healthcare data security and privacy," J. Parallel Distrib. Comput., vol. 164, pp. 152–167, 2022.

[5] K. Azbeg, O. Ouchetto, and S. J. Andaloussi, "Access control and privacy-preserving blockchain-based system for diseases management," IEEE Trans. Comput. Social Syst., vol. 10, no. 4, pp. 1515–1528, 2023.

[6] S. Wang, X. Wang, and Y. Zhang, "A secure cloud storage framework with access control based on blockchain," IEEE Access, vol. 7, pp. 112713–112727, 2019.

[7] M. A. Saviour and D. Samiappan, "IPFS based file storage access control and authentication model for secure data transfer using blockchain technique," Concurrency Comput.: Pract. Exper., 2023.

[8] M. A. Saviour and D. Samiappan, "IPFS based storage authentication and access control model with optimization enabled deep learning for intrusion detection," Adv. Eng. Softw., vol. 176, p. 103369, 2023.

[9] A. A. Battah, M. M. Madine, H. Alzaabi, I. Yaqoob, K. Salah, and R. Jayaraman, "Blockchain-based multi-party authorization for accessing IPFS encrypted data," IEEE Access, vol. 8, pp. 196813–196830, 2020.

[10] J. Sun et al., "Ciphertext-policy ABE + IPFS for healthcare/insurance sharing," Various Conferences/Journals, 2019.

[11] M. Chase, "Multi-authority attribute-based encryption," in Proc. CRYPTO / LNCS, 2007.

[12] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in Proc. IEEE Symp. Security Privacy, 2007.

[13] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in Adv. Cryptology (LNCS), 2011.

[14] G. Zyskind, O. Nathan, and A. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in Proc. IEEE Security Privacy Workshops, 2015.

[15] A. Ekblaw et al., "MedRec: Blockchain for medical data access and audit," in Proc. IEEE/ACM Health Informatics Workshops, 2016.

[16] Q. Xia et al., "Blockchain and smart-contract based access control and revocation frameworks," Various Journals/Conferences, 2017–2019.

[17] X. Liu et al., "Smart contract frameworks for tracking and enforcing data sharing agreements," Various Journals/Conferences, 2018–2020.

[18] C. Lin et al., "Blockchain-based mutual authentication and fine-grained access control," Various Journals/Conferences, 2018.

[19] M. Naz et al., "Blockchain and IPFS combined for secure data exchange and monetization," Various Journals/Conferences, 2019–2020.

[20] P. Kumar and C. P. Katti, "A parallel-SQLIA detector for web security," I.J. Inf. Eng. Electron. Bus., vol. 2, pp. 66–75, 2016.