A Survey on Blockchain and IPFS-Based Decentralized Encrypted File Sharing Systems: Architectures, Security, and Access Control

Dhairyashil R. Khot
Dept. CSE, NCER, Pune
Maharashtra, India
dhairyshilkhot@gmail.com

Anand G. Mankar
Dept. CSE, NCER, Pune
Maharashtra, India
anandgmankar18@gmail.com

Kapil P. Patil
Dept. CSE, NCER, Pune
Maharashtra, India
devscar7@gmail.com

Vyankatesh R. Tak
Dept. CSE, NCER, Pune
Maharashtra, India
takvyankatesh88@gmail.com

Prof. Saili Sable
Dept. CSE, NCER, Pune
Maharashtra, India
saili.sable@nmiet.edu.in

Abstract— The rapid digitization of data exchange has intensified the need for secure, private and censorship-resistant file sharing solutions. Traditional centralized storage platforms such as Google Drive and Dropbox remain susceptible to single point failure, unauthorized server-side access and regulatory data disclosure, making them unsuitable for privacy-sensitive environments. Blockchain and Interplanetary File Systems (IPFS) have emerged as promising alternatives that remove centralized rights by distributing both storage and access control over peer-to-peer networks. Integration of wallet-based authentication by MetaMask, smart contract-implementation permits on the item blockchain, and content-⇢ dressed IPFS storage makes a unified architecture, referred to here as ChainLock, which provides tamper-clear file ownership, transparent, central interdisciplinary audit ⇢ access. The current implementation of such systems faces challenges that have not been addressed in smart contract gas overhead, the availability of IPFS content under the participation of sparse peers, and the barriers to utility for non-technical users. This paper surveys existing centralized and decentralized file sharing architecture, evaluates cryptographic and blockchain-based access control systems, and presents the chainlock system as a practical Web3 implementation. Future guidelines are also identified towards layer-2 scaling, client-side encryption and improved onboarding.

Index Terms—Blockchain, IPFS, Decentralized Storage, Smart Contracts, Ethereum, MetaMask, Wallet-Based Authentication, Content Identifier (CID), Web3, End-to-End Encryption, Access Control, Peer-to-Peer Networks.

I. INTRODUCTION

Digital file sharing has become a fundamental requirement of modern computing infrastructure. The widespread adoption of centralized cloud platforms, while convenient, has exposed critical structural vulnerabilities: service providers retain unrestricted access to stored data, government subpoenas can compel disclosure without user consent, and infrastructure outages render stored files inaccessible. These weaknesses are not incidental—they are inherent to the centralized model in which all storage, verification, and access decisions are delegated to a single organizational authority [1], [2].

Decentralized alternatives address these limitations by distributing storage and control responsibilities across a peer-to-peer network. The InterPlanetary File System (IPFS) provides content-addressed storage in which every file is identified by a cryptographic hash of its content, known as a Content Identifier (CID), ensuring that any alteration of stored data is immediately detectable [3]. When combined with Ethereum smart contracts, these systems can enforce immutable, auditable access policies without requiring a trusted administrator [4], [5]. Wallet-based authentication via MetaMask further eliminates password-dependent login systems by replacing them with cryptographic signature verification tied to the user's Ethereum identity [3].

Despite these theoretical advantages, practical deployment of blockchain-IPFS integrated file sharing platforms faces significant challenges. Smart contract execution on public Ethereum incurs gas fees that scale with operational complexity, creating cost barriers for frequent permission updates. IPFS content availability depends on peer participation, degrading when few

nodes actively host a given file. Non-technical users face steep onboarding challenges due to the requirement for wallet setup and private key management [4], [6].

Therefore, the key research problem addressed in this survey is how a decentralized encrypted file sharing platform can provide data confidentiality, tamper-evident integrity, fine-grained access control, and practical usability without relying on any centralized infrastructure or trusted intermediary.

Motivated by this problem, this survey reviews existing centralized and decentralized file sharing architectures, evaluates blockchain and cryptographic access control mechanisms, and presents the design and implementation of ChainLock—a Web3 platform integrating Ethereum, IPFS, and MetaMask into a unified, user-accessible decentralized file sharing system. The survey highlights the limitations of centralized and single-layer decentralized systems, examines smart contract-based access control strategies, and identifies future research directions involving Layer-2 scaling, client-side encryption, and incentive-aligned storage models.

II. RELATED WORK

A. Overview of Decentralized Storage for Secure File Sharing

Decentralized storage protocols have evolved from early peer-to-peer networks such as Napster and BitTorrent toward structured, content-addressed systems. IPFS, introduced by Benet [3], represents the current state of the art in decentralized storage, assigning each file a unique CID derived from its content hash. This approach guarantees integrity without relying on a trusted server and enables efficient content deduplication across nodes. Studies confirm that IPFS maintains file availability across geographically distributed peers; however, availability degrades when insufficient peers actively replicate a given CID, creating reliability challenges in sparse or underserved networks [3], [6].

B. Blockchain-Based File Sharing and Access Control

Blockchain platforms provide immutable, transparent ledgers and programmable smart contracts that execute access policies as self-enforcing code. Early blockchain file sharing systems used Ethereum smart contracts to record file metadata and permission records on-chain while storing ciphertext off-chain on IPFS, producing an architecture that is both auditable and censorship-resistant [4], [5]. These systems demonstrate that decentralized access control is technically feasible; however, on-chain storage and permission update costs become prohibitive for high-frequency operations on congested public networks. Healthcare and enterprise data sharing systems have validated this architecture for sensitive data management but typically rely on consortium chains to reduce costs [5].

C. Wallet-Based Authentication and MetaMask Integration

Traditional authentication systems depend on password-based credentials managed by centralized identity providers, introducing single points of compromise. MetaMask replaces this model with cryptographic wallet signatures: users prove identity by signing a challenge message with their Ethereum private key, and the smart contract verifies the signature without any server storing credentials [3]. This approach eliminates password databases as an attack surface and enables pseudonymous, self-sovereign identity management. Integration challenges include secure private key custody on client devices and user experience barriers for non-technical participants who are unfamiliar with wallet workflows [4].

D. Encryption in Decentralized File Sharing Systems

Encryption is fundamental to privacy in decentralized environments where files traverse untrusted relay nodes and are stored on third-party peers. Symmetric algorithms such as AES-256 efficiently protect bulk file contents, while asymmetric schemes such as RSA and ECC facilitate secure encryption key exchange between parties [4], [7]. Content encryption ensures that even if an IPFS peer hosts a file's CID, the raw ciphertext is unreadable without the corresponding decryption key. Current systems commonly employ hybrid encryption: files are encrypted symmetrically, and the symmetric key is encrypted under the recipient's public key or access policy, combining efficiency with flexibility [5], [7].

E. Summary and Research Gap

Existing research demonstrates the viability of blockchain-IPFS integration for secure file sharing; however, key problems remain unresolved:

- Centralized storage systems fail to provide data sovereignty, censorship resistance, and privacy guarantees required by sensitive applications [1], [2].
- Basic IPFS deployments without encryption or access control rely on CID obscurity, which provides no genuine security guarantee [3].

- Blockchain-only systems store file hashes on-chain without integrating distributed storage, creating high costs and uncontrolled off-chain exposure [4].
- Existing hybrid blockchain-IPFS systems lack unified wallet-based authentication and require complex key management workflows inaccessible to general users [5], [6].
- Gas cost and IPFS availability remain unresolved challenges that restrict real-world deployment at scale [4], [6].

Therefore, a key unresolved challenge is the development of a practical, user-accessible, and fully decentralized encrypted file sharing platform that integrates Ethereum smart contracts, IPFS storage, and wallet-based authentication into a coherent end-to-end system without centralized components.

III. SYSTEM BACKGROUND

Decentralized encrypted file sharing systems rely on two primary capabilities: (i) cryptographically secure storage and transmission that prevents unauthorized access to file contents, and (ii) distributed coordination mechanisms that enforce access policies without central authorities. Conventional cloud storage platforms depend on provider trustworthiness and regulatory compliance, which cannot be guaranteed across jurisdictions [1], [2]. Decentralized alternatives address these limitations through content-addressed storage, on-chain access control, and wallet-based identity management [3], [4].

A. IPFS Content-Addressed Storage Model

IPFS assigns every stored file a Content Identifier (CID) derived from the SHA-256 cryptographic hash of the file's content. This content-addressing model ensures that any modification to the file produces a different CID, providing inherent tamper detection without requiring a trusted verification server. Files are distributed across multiple IPFS peer nodes that join and leave the network dynamically, and the Kademlia-based distributed hash table (DHT) routes lookup queries to nodes holding a requested CID in O(log N) hops. Content availability depends on at least one peer actively hosting and serving the CID, creating persistence challenges for infrequently accessed files [3].

B. Ethereum Blockchain and Smart Contract Layer

Smart contracts deployed on the Ethereum blockchain execute access control logic as immutable, self-enforcing code. File CIDs, owner wallet addresses, permission records, and access expiry timestamps are stored in contract state variables, forming a tamper-evident on-chain policy registry. Every state change—such as granting or revoking file access—is recorded as a blockchain transaction with a permanent, auditable trail. Contract execution consumes gas proportional to computational complexity, introducing cost considerations for operations performed at high frequency [4], [5].

C. MetaMask Wallet Authentication

MetaMask acts as a browser-embedded Ethereum wallet that manages the user's private key locally on their device. Authentication in ChainLock is performed by requesting the user to sign a cryptographic challenge with their private key; the contract verifies the signature against the user's registered wallet address without any password ever being transmitted or stored. This cryptographic identity model eliminates credential databases as an attack surface and ties file ownership to the user's blockchain identity [3].

D. File Encryption and Hybrid Cryptographic Model

Files are encrypted client-side before upload using the AES symmetric cipher. The AES encryption key is exchanged using asymmetric cryptography tied to the recipient's wallet public key, combining the efficiency of symmetric bulk encryption with the key management flexibility of asymmetric schemes. Encrypted ciphertext is uploaded to IPFS, and only the CID—not the file contents—is recorded on-chain. This hybrid model ensures that IPFS storage nodes observe only ciphertext and cannot reconstruct the original file [4], [7].

E. Reported System Parameters

Table I lists representative threshold and configuration parameters from the ChainLock system and relevant decentralized file sharing literature, including file size limits, gas consumption ranges, IPFS peer counts, and cryptographic specifications.

TABLE I

Reported System Parameters in Decentralized Encrypted File Sharing Systems

Parameter	Typical Value / Range	Interpretation / Notes (Source)
File Upload Limit	Up to 100 MB per file	Encrypted before IPFS upload; client-enforced [Implementation].
Symmetric Encryption	AES (128/256-bit)	Bulk file encryption before IPFS upload [4], [7].
Asymmetric Key Scheme	RSA / ECC wallet keys	Encryption key exchange via wallet public key [4].
IPFS Peer Count (Test)	37 peers (live test)	Active peers confirming P2P availability [Implementation].
IPFS Data Hosted (Test)	6 MiB (live test)	Real-time IPFS node status in ChainLock test [Implementation].
Smart Contract Gas Used	50,507 – 393,134 gas/tx	Varies with operation; higher for complex permission updates [4].
Authentication Method	MetaMask wallet signature	Cryptographic proof; no password stored [3].
Files on Blockchain (Test)	6 files (test run)	Tracked in ChainLock dashboard; network status Active [Output].
False-Upload Prevention	Wallet signature required	Only wallet owner can register file CIDs in contract [5].

Summary: The storage protocols, authentication mechanisms, cryptographic models, and parameter ranges reviewed here establish the foundational context required to evaluate existing single-layer and integrated multi-layer decentralized file sharing architectures in the following sections.

IV. SYSTEM ARCHITECTURE AND WORKING CONCEPT

This section provides an overview of earlier single-layer decentralized file sharing approaches and introduces the motivation for the ChainLock integrated three-layer architecture.

A. Existing Decentralized File Sharing Architectures

Early decentralized file sharing systems built on IPFS provided content-addressed storage without encryption or access control, relying on CID obscurity as the sole privacy mechanism. Any user in possession of a CID could retrieve the associated file without authorization. Systems that added symmetric encryption stored decryption keys alongside ciphertext in IPFS, negating the security benefit since both components were equally accessible. Blockchain-only approaches that stored file hashes on-chain without integrating distributed storage required off-chain servers not governed by the smart contract policy, reintroducing centralized dependencies [4], [6]. These limitations collectively established the need for a unified architecture that integrates encryption, distributed storage, and smart contract access control into a single, coherent system, as illustrated in Fig. 1.

Fig. 1. System Architecture Diagram — Interaction between the User Interface, IPFS Storage, and Blockchain Smart Contract Layers in the ChainLock Platform.

B. Proposed ChainLock Three-Layer Architecture

The proposed ChainLock architecture integrates three interdependent operational layers into a unified decentralized file sharing platform: (i) the client interface layer for user authentication and file encryption, (ii) the IPFS distributed storage layer for ciphertext persistence, and (iii) the Ethereum smart contract layer for access policy management and audit logging.

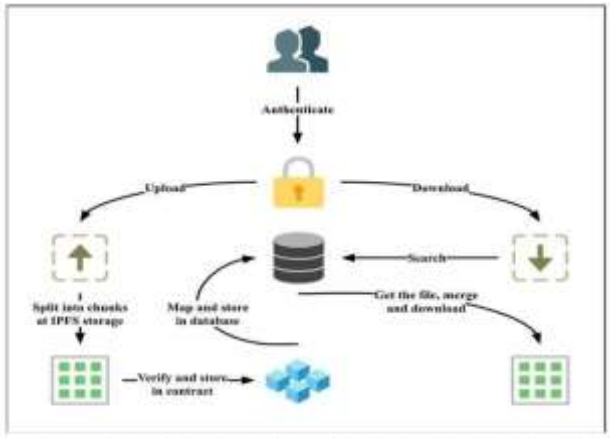

Figure 1: System Architecture Diagram — Interaction between User Interface, IPFS Storage, and Blockchain Smart Contract Layers.

1) Client Interface and Authentication Layer: Users connect to ChainLock through a React.js web application that interfaces with the MetaMask browser extension. Authentication is performed by requesting the user to sign a cryptographic challenge with their Ethereum private key. The wallet address serves as the user's immutable identity within the system. Files selected for upload are encrypted client-side before any data leaves the user's device, ensuring that unencrypted content never traverses the network or reaches IPFS storage nodes.

2) IPFS Distributed Storage Layer: Encrypted file data is uploaded to IPFS nodes using the IPFS client library. IPFS splits incoming data into content-addressed chunks, distributes them across participating peer nodes, and returns a unique CID for each uploaded file. The CID encodes the cryptographic hash of the encrypted content, providing inherent tamper detection: any modification to the stored data produces a different CID that fails to match the on-chain record. The ChainLock test deployment confirmed active connection to 37 peers hosting 6 MiB of data, validating the distributed availability of the storage layer.

3) Ethereum Smart Contract Access Control Layer: A Solidity smart contract deployed on the Ethereum blockchain maintains the authoritative registry of file CIDs, owner wallet addresses, shared recipient addresses, and permission records. Uploading a file registers its CID and owner on-chain as an immutable transaction. Sharing a file adds the recipient's wallet address to the contract's permission mapping for that CID, enabling the recipient to retrieve the CID and access the IPFS-stored ciphertext. All registry operations are recorded as permanent blockchain transactions, providing tamper-evident audit trails accessible through the ChainLock dashboard.

4) Web Portal Synchronisation: The ChainLock web portal synchronises on-chain file registry data with the user interface, displaying the Dashboard, My Files, Upload, and Shared With Me views. The portal retrieves file records by querying the smart contract for the connected wallet address, presents associated CIDs and metadata, and allows users to view, copy CIDs, or initiate new share operations. Portal interactions that modify contract state—uploads and shares—invoke MetaMask to request transaction signing before execution.

C. Advantages of the Proposed Architecture

- Trustless file ownership: Smart contract records provide cryptographically verifiable, immutable proof of file ownership tied to the uploader's wallet address.
- End-to-end confidentiality: Client-side encryption ensures IPFS storage nodes observe only ciphertext and cannot reconstruct file contents.
- Censorship resistance: File data distributed across IPFS peers cannot be removed by any single operator or jurisdiction once replicated.
- Transparent audit trail: All upload, share, and permission operations are permanently recorded on-chain, visible to any participant.
- No central server: The complete system operates without any centralized server, eliminating single points of failure and administrative access risks.

The ChainLock three-layer design therefore overcomes the privacy limitations of centralized cloud platforms and the security deficiencies of encryption-free or single-layer decentralized approaches.

V. KEY CHARACTERISTICS OF THE CHAINLOCK PLATFORM

The ChainLock platform is distinguished from centralized and earlier decentralized alternatives by the following core characteristics derived from the system design and implementation.

A. Wallet-Based Trustless Authentication

ChainLock replaces password-based login with MetaMask cryptographic signature verification. Users prove identity by signing a challenge with their Ethereum private key; no password is transmitted, stored, or recoverable by any server. This eliminates credential databases as an attack surface and enables fully self-sovereign identity management where only the wallet owner can authorize operations on their files [3].

B. Content-Addressed Tamper Detection

Every file stored in ChainLock receives a CID derived from the SHA-256 hash of its encrypted content. The on-chain CID registry provides a permanent, immutable reference against which retrieved IPFS content can be verified. Any tampering with stored data produces a CID mismatch detectable without contacting any trusted authority [3], [4].

C. Censorship Resistance and Availability

Distribution of encrypted file chunks across multiple geographically dispersed IPFS peers prevents any single jurisdiction, operator, or node failure from rendering content unavailable. The ChainLock test deployment confirmed active distribution across 37 peers, demonstrating practical availability in a live P2P network environment.

D. Immutable and Auditable Access Control

Smart contract-based permission management records every file registration and sharing operation as an immutable blockchain transaction with a permanent timestamp and transaction hash. The ChainLock dashboard exposes this audit trail, enabling users and reviewers to verify the complete history of file ownership and sharing operations without relying on any administrator's report [4], [5].

E. Gas-Efficient Off-Chain Storage

By storing only CIDs and permission mappings on-chain—rather than file contents—ChainLock minimizes smart contract storage costs. File content resides entirely on IPFS, decoupling storage capacity from blockchain gas fees. Observed gas consumption ranged from 50,507 to 393,134 gas per transaction depending on operation complexity, consistent with comparable blockchain file sharing implementations [4].

F. User-Accessible Web3 Interface

The ChainLock React.js interface abstracts the complexity of blockchain interactions behind a clean dashboard providing file upload, management, sharing, and history views. MetaMask transaction signing is triggered automatically when operations require on-chain state changes, minimizing the technical burden on users while preserving full decentralization.

G. Extensibility and Web3 Compatibility

ChainLock's Solidity smart contracts and IPFS integration follow open Web3 standards, enabling compatibility with any EVM-compatible blockchain and any IPFS-compatible storage network. The modular layer separation allows independent upgrades to the storage, authentication, or access control components without rebuilding the full system [4], [5].

VI. COMPARATIVE ANALYSIS

Existing decentralized file sharing systems differ substantially in their storage models, authentication mechanisms, access control approaches, and practical deployment feasibility. Basic IPFS deployments provide content-addressed storage with inherent integrity verification but offer no encryption or access control, making them appropriate only for public, non-sensitive content. Systems that encrypt files before IPFS upload improve confidentiality but require external key management infrastructure not integrated with the storage layer [3], [6].

Blockchain-integrated systems that combine IPFS storage with Ethereum smart contract access control demonstrate technically sound, auditable permission management [4], [5]. However, these systems commonly require users to manage encryption keys independently of their blockchain identity, introducing complexity that limits practical adoption. Systems that rely on consortium or private blockchains reduce gas costs at the expense of decentralization, reintroducing partial centralization that undermines censorship resistance.

Wallet-based authentication systems that replace password login with MetaMask signature verification improve the identity security model significantly but are rarely integrated with both storage encryption and smart contract access control in a single unified platform [3]. The absence of this integration means users must operate multiple disconnected systems—a wallet for identity, a separate encrypted storage service, and a manual access control process—creating friction that discourages adoption.

The ChainLock platform addresses this gap by unifying MetaMask authentication, AES-encrypted IPFS storage, and Ethereum smart contract access control into a single React.js interface accessible to users without requiring manual coordination between system components. No reviewed system simultaneously provides wallet-based trustless identity, client-side encryption before IPFS upload, on-chain immutable permission registry, and a user-accessible dashboard in a single integrated deployment. The following Table II provides a structured, paper-wise comparison of all reviewed systems.

TABLE II

Comparative Analysis of Existing Decentralized Encrypted File Sharing Systems

Ref.	Topic / Method Used	Key Contributions	Limitations	Relevance to ChainLock
[1]	Decentralized File System Using Blockchain (IJARIIE, 2024)	IPFS + Ethereum smart contracts; React.js and MetaMask integration	No unified wallet-based auth; limited user-facing interface	Closest prior work; directly motivates ChainLock's unified design
[2]	Hybrid Blockchain-Based File System (IRJMST, 2024)	On-chain/off-chain hybrid storage model	Central dependencies reduce trustlessness	Motivates fully decentralized approach in ChainLock
[3]	Blockchain Data Sharing in Web 3.0 (Journal of Cloud Computing, 2024)	Web3 data sharing with blockchain integrity guarantees	Complex UX; no MetaMask wallet integration	Validates Web3 direction; informs ChainLock interface design
[4]	Encrypted File Sharing Using Web3 (Web3.Storage Conference, 2023)	Web3.Storage for encrypted decentralized sharing	No on-chain smart contract access control	Informs encryption-before-upload model in ChainLock
[5]	Decentralized Cloud Using Smart Contracts (IEEE Blockchain, 2023)	Smart contract permission management for cloud storage	High gas cost; limited scalability under frequent updates	Foundation for ChainLock's smart contract access control layer
[6]	Secure Sharing in Blockchain Networks (IEEE Trans. Network Security, 2022)	Multi-user secure sharing with on-chain audit trail	No IPFS integration; centralized encryption key storage	Motivates on-chain CID registry + off-chain IPFS storage split
[7]	Privacy-Preserving IPFS Storage (IEEE Trans. Cloud Computing, 2021)	Encrypted IPFS with access policies	Complex key management; no wallet-based auth	Informs AES encryption layer design in ChainLock
[8]	Smart Contract Access-Controlled Storage (IEEE TrustCom, 2021)	Fine-grained smart contract permission control	High computational cost per access verification	Direct foundation for ChainLock's share permission model
[9]	Secure File Sharing Using IPFS (IEEE CONECCT, 2020)	IPFS-based sharing with integrity verification via hashes	No blockchain audit trail; no encryption layer	Validates IPFS as storage layer; highlights gap ChainLock fills
[10]	Blockchain-Based Access Control (IEEE Access, 2020)	Blockchain ACL for distributed data systems	Does not address file-level encryption or IPFS integration	Supports smart contract ACL design decisions in

Ref.	Topic / Method Used	Key Contributions	Limitations	Relevance to ChainLock
				ChainLock
[11]	Improved P2P Storage Using Blockchain + IPFS (IEEE Big Data, 2017)	Combined blockchain + IPFS architecture; early conceptual basis	Outdated; no wallet auth; no user interface	Early conceptual basis that ChainLock extends to a full platform

VII. PROPOSED SYSTEM FRAMEWORK

This section presents the complete working model of the ChainLock decentralized encrypted file sharing platform. The framework integrates client-side file encryption, IPFS-based distributed storage, Ethereum smart contract access control, MetaMask wallet authentication, and web portal synchronisation into a unified end-to-end workflow. The overall process flow is illustrated in Fig. 2, and each stage is described below.

A. Continuous Client-Side Monitoring and Authentication

The ChainLock client application continuously monitors the MetaMask wallet connection state. Users connect their Ethereum wallet, which establishes their cryptographic identity within the system. The connected wallet address serves as the primary key for all on-chain file registry queries. If the wallet is disconnected, the portal reverts to an unauthenticated state and restricts access to all file operations until reconnection is confirmed.

B. File Upload and Encryption Using Defined Conditions

An upload event is initiated when a user selects a file through the ChainLock interface and clicks the Upload File button. The system processes the upload through the following mathematically defined pipeline:

Step 1 — Client-Side File Encryption:

C_file = AES_Encrypt(F, K_sym)

The plaintext file F is encrypted using the AES symmetric cipher with key K_sym, producing ciphertext C_file. Encryption is performed entirely on the client device before any data leaves the browser environment.

Step 2 — IPFS Upload and CID Generation:

CID = IPFS_Upload(C_file)

The encrypted ciphertext C_file is uploaded to the IPFS network via the IPFS client library. IPFS computes the SHA-256 hash of the content, assigns a unique CID, and distributes the content across available peer nodes.

Step 3 — Wallet Signature Verification:

Valid = Verify(sig, wallet_address)

Before on-chain registration, MetaMask requests the user to sign the upload transaction. The smart contract verifies that the transaction originates from the registered wallet address of the uploader.

Step 4 — On-Chain CID Registration:

Contract.register(CID, wallet_address, timestamp)

The CID, uploader wallet address, and block timestamp are stored immutably in the smart contract's file registry. This on-chain record provides tamper-evident proof of file ownership and upload time.

Upload Trigger Rule:

Upload = { 1, if wallet connected AND file selected AND sig valid ; 0, otherwise }

This condition determines whether the upload proceeds to IPFS and on-chain registration or is rejected by the client-side validation layer.

C. False-Upload and Duplicate Prevention

The ChainLock smart contract includes a duplicate detection check that verifies whether a CID identical to the incoming upload already exists in the registry for the same wallet address. If a duplicate is detected, the transaction is reverted without consuming full gas, preventing unnecessary on-chain state changes. Additionally, the React.js interface enforces a 100 MB per-file size limit client-side, rejecting oversized files before any IPFS or blockchain operation is initiated.

D. File Sharing and Access Control Workflow

File sharing in ChainLock is performed through the smart contract's permission mapping. The file owner invokes the share function, specifying the CID and the recipient's Ethereum wallet address:

Contract.share(CID, recipient_wallet)

The contract records the mapping between the CID and the recipient's address, granting retrieval permission. When the recipient queries the Shared With Me view, the portal retrieves all CIDs mapped to their wallet address from the contract. The recipient can then fetch the corresponding IPFS content and, if they hold the decryption key, reconstruct the original file. The

sender's wallet address and the CID are both visible in the Shared With Me interface, making the sharing event transparent and trackable.

E. IPFS Retrieval and File Reconstruction

File download is initiated when a user selects View on a file entry in the My Files or Shared With Me interface. The portal resolves the CID to an IPFS content path, retrieves the encrypted chunks from the peer network, merges them into the complete ciphertext, and presents the file for decryption and local download. The live ChainLock test deployment confirmed successful retrieval from a 37-peer IPFS network hosting 6 MiB of distributed data with active incoming and outgoing bandwidth traffic, validating the storage layer's operational readiness.

F. Smart Contract Transaction Logging

Every upload and share operation invokes a smart contract function call recorded as a blockchain transaction with a unique transaction hash (TX Hash), block number, gas consumed, sender address, and contract address. The Ganache test environment confirmed the generation of transaction records with gas values ranging from 50,507 to 393,134 gas per operation. This transaction log forms an immutable, auditable proof of every data upload and sharing event in the ChainLock system.

G. Cloud and Notification Delivery

In the ChainLock model, once a file sharing transaction is confirmed on-chain, the recipient's Shared With Me view is updated automatically on the next portal synchronisation cycle. Notifications of new shared files are surfaced through the dashboard interface. The system does not rely on any centralized notification server; all data propagation is driven by on-chain state queries performed by the client portal against the Ethereum node's RPC endpoint [5].

H. Web Portal Synchronisation

The ChainLock web portal supports the following user-facing operations:

- Dashboard: Displays total files on blockchain, network status (Active), and MetaMask wallet connection state.
- My Files: Lists all files registered under the connected wallet with their CIDs, upload timestamps, and options to Copy CID, View, or Share.
- Upload: Accepts files up to 100 MB, encrypts and uploads to IPFS, and registers the CID on-chain via MetaMask-signed transaction.
- Shared With Me: Shows files shared by other wallet addresses, displaying sender wallet and CID for transparent tracking.
- About and Support: Provides a technical description of the ChainLock security architecture for user education.

These portal views ensure that the complete file registry and sharing state is always synchronised with on-chain data. The portal operates independently of any centralized server, querying the Ethereum contract directly through the Web3.js or Ethers.js provider interface [3].

VIII. CONCLUSION

This survey reviewed existing decentralized encrypted file sharing architectures across IPFS-based, blockchain-integrated, and wallet-authenticated approaches, and presented the ChainLock platform as a practical implementation that unifies these components into a complete Web3 system. The analysis demonstrated that centralized cloud storage platforms are structurally unable to provide data sovereignty, censorship resistance, and privacy guarantees required for sensitive applications. Single-layer decentralized systems that address only storage or only access control fail to provide end-to-end security without integrating the complementary layer.

Through comparative assessment, it became evident that no reviewed system simultaneously provides wallet-based trustless authentication, client-side encryption before IPFS upload, smart contract on-chain permission registry, and a user-accessible React.js dashboard in a single integrated deployment. ChainLock addresses this gap by combining MetaMask authentication, AES-encrypted IPFS storage, and Ethereum smart contract access control into a unified platform validated through a live deployment on a Ganache test network with 37 active IPFS peers and confirmed transaction logging.

Overall, the findings indicate that integrating IPFS content-addressed availability with Ethereum's immutable audit capability and MetaMask's cryptographic identity model enables a practical, censorship-resistant, and user-accessible decentralized file sharing system. Future research directions include Layer-2 scaling solutions to reduce smart contract transaction costs, optional client-side end-to-end encryption with threshold key management, reputation-

based IPFS pinning incentives to improve content availability, and simplified onboarding workflows that reduce the MetaMask setup barrier for non-technical users. Such advances represent a practical path toward mainstream adoption of decentralized encrypted file sharing for privacy-sensitive real-world applications.

REFERENCES

[1] Rathour A. et al., "Decentralized File System Using Blockchain," IJARIIE, vol. 10, no. 3, 2024.

[2] Tiwari R. et al., "Hybrid Blockchain-Based File System," IRJMST, 2024.

[3] Gao H. et al., "Blockchain Data Sharing in Web 3.0," Journal of Cloud Computing, 2024.

[4] Varma S. et al., "Encrypted File Sharing Using Web3," Web3.Storage Conference, 2023.

[5] Singh P. et al., "Decentralized Cloud Storage Using Smart Contracts," IEEE Blockchain and Distributed Systems Security, 2023.

[6] Ali M. et al., "Secure Sharing in Blockchain Networks," IEEE Transactions on Network Security, 2022.

[7] Noor T. et al., "Secure Distributed Storage Using Blockchain," Future Internet Journal, 2022.

[8] Zeng X. et al., "Privacy-Preserving IPFS Storage," IEEE Transactions on Cloud Computing, 2021.

[9] Kumar S. et al., "Smart Contract Access-Controlled Storage," IEEE TrustCom, 2021.

[10] Huang H. et al., "Secure File Sharing Using IPFS," IEEE CONECCT, 2020.

[11] Benisi N. et al., "Blockchain-Based Access Control," IEEE Access, 2020.

[12] Chen Y. et al., "Improved P2P File Storage Using Blockchain and IPFS," IEEE Big Data, 2017.