International Journal of Computer Applications (0975 - 8887)
Volume 187 - No.103, May 2026

De-Drive: A Zero-Knowledge Hybrid Decentralized Storage Architecture Leveraging IPFS, Ethereum, and Client-Side AES Cryptography

Hamsadhwaj M.
Computer Science and Engineering
Jain University, Bengaluru
Karnataka, India

Nagendra Prasad K.G.
Computer Science and Engineering
Jain University, Bengaluru
Karnataka, India

Harsh K.
Computer Science and Engineering
Jain University, Bengaluru
Karnataka, India

ABSTRACT

The rapid digitalization of global infrastructure has amplified the vulnerabilities inherent in centralized cloud storage systems, where single points of failure, administrative access abuses, and external cyberattacks routinely compromise sensitive data. Traditional cloud architectures rely on centralized trust models that frequently succumb to data breaches and censorship. To address these critical flaws, this paper proposes De-Drive, a zero-knowledge, hybrid decentralized storage application (DApp) that seamlessly integrates Web3 architecture with robust cryptographic protocols. De-Drive leverages a hybrid storage model: heavy file payloads are stored off-chain on the decentralized InterPlanetary File System (IPFS) via Pinata nodes, while the immutable reference links—Content Identifiers (CIDs)—are permanently logged on the Ethereum blockchain (Sepolia Testnet) utilizing a custom, highly gas-optimized Solidity smart contract. To solve the inherent privacy flaws of public IPFS networks, De-Drive implements strict client-side Advanced Encryption Standard (AES) cryptography. Files are converted to Base64 strings and encrypted locally within the React frontend utilizing a user-defined symmetric key prior to network transmission. This architecture ensures absolute zero-knowledge storage; neither the IPFS nodes hosting the data nor the blockchain observers auditing the ledger can decipher the underlying content without the specific decryption key. The proposed architecture successfully resolves the blockchain storage trilemma by delivering a decentralized, immutable, and strictly private data vault, demonstrating significant improvements in cost-efficiency and security over both traditional cloud services and purely on-chain storage alternatives.

General Terms

Decentralized Storage, Blockchain, Cryptography, Web3, Distributed Systems, Data Privacy, Smart Contracts

Keywords

IPFS, Ethereum, AES-256, Zero-Knowledge Storage, DApp, Solidity, Hybrid Architecture, Client-Side Encryption, Pinata, MetaMask, ethers.js, Gas Optimization, Web3

1. INTRODUCTION

The architecture of the internet is undergoing a foundational paradigm shift from the centralized models of Web2 to the decentralized, peer-to-peer frameworks of Web3. For over two decades, data storage has been dominated by an oligopoly of centralized cloud service providers, primarily Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. While these platforms have delivered unprecedented convenience, massive economies of scale, and highly optimized user experiences, they mandate a fundamental and increasingly dangerous concession: the total surrender of data ownership and user privacy. In the Web2 paradigm, users must trust centralized entities to secure their data against external threats, manage access controls ethically, and resist unwarranted censorship. This trust model is failing.

The reliance on centralized servers has precipitated a continuous escalation in data breaches, establishing a systemic security crisis. Recent cyber threat intelligence reports indicate that cloud security incidents are increasingly severe, driven by misconfigurations, compromised identities, and sophisticated ransomware attacks. The 2024–2025 period witnessed massive disruptions, such as the Snowflake data breach and widespread infrastructure outages, underscoring the extreme fragility of centralized systems. Threat actors increasingly target supply chains and exploit vulnerabilities in Identity and Access Management (IAM) configurations, with stolen credentials currently ranking among the most common initial infection vectors. Furthermore, the M-Trends 2025 report highlights that 55% of threat groups are financially motivated, actively seeking to exfiltrate and extort sensitive data stored in centralized repositories (15).

In highly regulated sectors such as healthcare and finance, the consequences of these vulnerabilities are catastrophic. The transition to digital record-keeping has exposed significant weaknesses; for example, incidents of hacking and unauthorized network access in healthcare have surged dramatically, highlighting the sheer inadequacy of centralized databases to protect sensitive patient information from advanced persistent threats (12). These statistical trends unequivocally demonstrate that centralized cloud storage is not merely an architectural bottleneck, but a severe operational liability.

Web3 marks a significant evolution in internet architecture by shifting control from centralized authorities to decentralized frameworks powered by blockchain technology. By decentralizing control and utilizing cryptographic proofs, Web3 empowers users to reclaim data sovereignty. The foundational role of Layer-1 blockchain infrastructure introduces trustless execution, immutability, and transparency. However, applying blockchain technology directly to data storage introduces profound scalability and cost challenges, necessitating hybrid architectural models to achieve practical utility.

The objective of this research is to design, implement, and analyze a decentralized storage framework—De-Drive—that mitigates the vulnerabilities of centralized systems while overcoming the limitations of contemporary decentralized networks. The primary goal is to establish a truly private, user-owned storage vault that operates on a strict zero-knowledge basis. This paper details the engineering of De-Drive, which:

[leftmargin=*, itemsep=1pt, parsep=0pt]Eliminates single points of failure by distributing file storage across the IPFS peer-to-peer network; Guarantees data integrity by logging file references permanently on the Ethereum blockchain; Ensures confidentiality through a robust client-side AES-256 encryption layer; and Optimizes Ethereum smart contracts to minimize gas consumption, making the hybrid architecture economically viable for everyday enterprise and consumer use.

2. LITERATURE REVIEW AND RELATED WORK

2.1 Centralized Cloud Infrastructure vs. Decentralized Networks

Traditional cloud storage platforms operate on client-server architectures where data is stored in massive, centrally managed data centers. These systems utilize location-based addressing (e.g., URLs) to retrieve data. The fundamental flaw in this architecture is the single point of failure; if the server hosting the URL fails, if the DNS routing is hijacked, or if the domain is censored by a governing authority, the data becomes entirely inaccessible.

In stark contrast, decentralized networks utilize content-based addressing. The InterPlanetary File System (IPFS), proposed by Juan Benet in 2014, is a peer-to-peer distributed file system that connects all computing devices with the same system of files (8). Instead of asking a central server *where* a file is located, IPFS asks the distributed network for the specific cryptographic hash (Content Identifier, or CID) of *what* the file is. This high-throughput, content-addressed block storage model forms a generalized Merkle Directed Acyclic Graph (DAG). Because the CID is derived directly from the file's binary content, the data is inherently tamper-proof; any alteration to the file alters its hash, immediately alerting the network to the modification.

Concurrently, the development of Ethereum, introduced by Vitalik Buterin in 2013, provided a foundational layer for decentralized state execution (10). Ethereum functions as a blockchain with a built-in Turing-complete programming language, enabling the deployment of smart contracts. These smart contracts act as cryptographic mechanisms that execute arbitrary rules for ownership, transaction formats, and state transition functions. While Ethereum provides an unalterable ledger for truth, it is entirely unsuited for bulk data storage due to the exorbitant cost of modifying the global state.

2.2 The Evolution of Hybrid Blockchain Storage

Recent academic literature extensively explores the integration of IPFS and blockchain to create functional hybrid architectures (26). Blockchain networks inherently struggle with data storage due to extreme redundancy requirements; every full node in the network must store a complete copy of the ledger to maintain consensus. Consequently, storing large-scale raw data directly on-chain results in exponential state bloat and is computationally and economically prohibitive.

The hybrid model—frequently analyzed in recent studies as IPFS-BC—addresses this by storing heavy data payloads off-chain in the IPFS network, while the blockchain only retains the unique CID hash and associated access control metadata. This approach reduces blockchain storage space consumption by approximately 75% compared to native on-chain storage, significantly decreasing data redundancy while maintaining absolute mathematical proof of the data's existence and integrity. Such architectures have been successfully proposed and evaluated for academic record verification, electronic healthcare records (EHR), and industrial IoT data management, demonstrating substantial improvements in scalability, operational costs, and retrieval efficiency (6).

2.3 The Privacy Gap: The Loophole in Public Decentralized Networks

While IPFS excels at permanence, immutability, and decentralization, it fundamentally fails at privacy. IPFS is an inherently public network; any user possessing a file's CID can query the Distributed Hash Table (DHT) and retrieve the file from the hosting peers (22). Extensive research into the privacy limitations of IPFS reveals that complete anonymity is mathematically and architecturally impossible within the network, and sensitive content cannot be natively offered privately without exposing the provider's IP address and the payload's plaintext contents (22).

A significant research gap exists concerning standardized content encryption mechanisms natively embedded within IPFS (11). While IPFS employs transport encryption (SecIO or TLS) to secure data transmission between nodes, it entirely lacks built-in content encryption, leaving stored data completely vulnerable to unauthorized access the moment it is retrieved (11). Relying solely on a smart contract and IPFS is insufficient for applications requiring confidentiality (12). If an entity uploads an unencrypted medical record to IPFS and stores the CID on Ethereum, the blockchain publicly broadcasts the CID to the world, essentially providing global, permanent access to the sensitive file.

To bridge this critical loophole, recent studies emphasize the absolute necessity of client-side encryption applied before data enters the decentralized network (12). By encrypting data at the source (source-side or client-side encryption), users maintain sole custody of the decryption keys (25). This ensures that even if a malicious actor, network observer, or node operator intercepts the CID from the public blockchain ledger, the underlying IPFS payload remains an unintelligible cryptographic cipher (25). De-Drive is explicitly engineered to resolve this vulnerability by embedding a rigorous AES encryption protocol directly into the client application, ensuring a zero-knowledge ecosystem where trust in the storage provider is entirely eliminated.

3. SYSTEM ARCHITECTURE AND OPERATIONAL FLOW

The De-Drive architecture is designed to orchestrate a seamless interaction between the user, a cryptographic frontend interface, the IPFS off-chain storage network, and the Ethereum on-chain state machine. The system ensures that data is entirely obfuscated before it leaves the user's localized hardware environment, effectively decoupling the act of data storage from the act of data ownership and visibility.

3.1 Architectural Flowchart

The lifecycle of a file within the De-Drive ecosystem follows a highly specific, bifurcated pathway. This architecture separates the heavy lifting of physical data hosting from the cryptographic record-keeping required for ownership provenance.

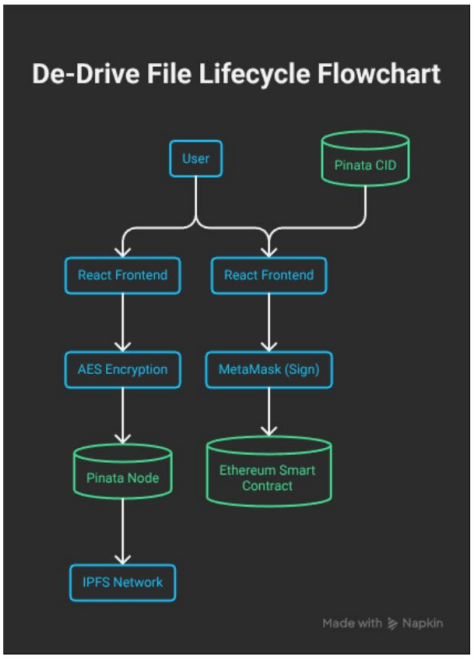

Fig. 1. De-Drive File Lifecycle Flowchart illustrating the dual-path hybrid architecture (Path A: Off-Chain Data Flow; Path B: On-Chain State Flow).

3.2 Explanation of the Dual-Path Methodology

The bifurcation of Path A (Payload) and Path B (Provenance) is the foundational cornerstone of the hybrid Web3 architecture.
Path A (Off-Chain Data Flow): The user selects a file via the React frontend and inputs a secret symmetric passphrase. Before any network requests are initiated, the frontend utilizes the local processing power of the user's browser to convert the file into a Base64 string and apply the AES-256 encryption algorithm. The resulting ciphertext is packaged into a binary Blob format. This encrypted Blob is then transmitted via an API call to a Pinata node. Pinata acts as an IPFS pinning service, ensuring the data is broadcast to the global DHT and persistently hosted, rather than relying on the user's browser node to stay online. The IPFS network hashes the encrypted payload and returns a unique CID (e.g., a `Qm...` hash string), which represents the immutable, cryptographic address of the encrypted file. Because the file was encrypted locally, the IPFS network is completely blind to the actual contents of the payload.
Path B (On-Chain State Flow): Upon receiving the CID from the Pinata gateway, the frontend application shifts from data hosting to state management. It triggers the user's browser wallet (e.g., MetaMask) via the `ethers.js` library. The user is prompted to cryptographically sign a transaction authorizing the logging of the CID to the Ethereum Sepolia Testnet. The `IPFSRegistry.sol` smart contract receives this transaction, validates the data to prevent redundant uploads, and permanently logs the CID against the user's Ethereum wallet address in the contract's state mapping.
Storing data directly on the Ethereum Virtual Machine (EVM) requires updating the state across tens of thousands of global nodes, an operation that incurs exorbitant gas fees strictly proportional to the data size (18). By offloading the raw data payload to IPFS (Path A), the system leverages a network optimized specifically for high-throughput, unstructured file hosting. Ethereum (Path B) is utilized strictly as a decentralized index and registry. It acts as an unbreakable cryptographic notary that mathematically proves a user uploaded a specific file at a specific time, without the blockchain ever bearing the weight of the file itself. This division of labor achieves the scalability and speed of traditional cloud storage while retaining the censorship-resistance and immutability of a decentralized consensus mechanism (26).

4. IMPLEMENTATION DETAILS

The engineering of De-Drive requires intricate coordination across three distinct technical domains: smart contract optimization within the Solidity environment, client-side cryptographic processing in JavaScript, and a responsive frontend bridge utilizing modern web frameworks to manage the decentralized state.

4.1 Smart Contract Engineering and Gas Optimization

The core logic of the blockchain layer is contained within the `IPFSRegistry.sol` smart contract, deployed on the Ethereum Sepolia Testnet. Because every individual operation on the EVM requires the expenditure of computational "gas" (paid in Ether), extreme engineering care was taken to minimize execution costs for the end user.

4.1.1 Calldata vs. Memory Data Locations. In Solidity, the choice of data location for reference types drastically impacts the gas consumption of a transaction (24). For array and string parameters passed to functions, developers traditionally default to using `memory`. Declaring a variable in `memory` forces the EVM to create a temporary, mutable copy of the variable during execution (24). This memory expansion and copying overhead becomes highly expensive for large arrays or strings, as each element copied incurs additional sequential gas costs (19).
To achieve optimal efficiency, De-Drive strictly utilizes the `calldata` keyword for external function parameters that only need to be read (19). Calldata is a non-modifiable, non-persistent, and read-only area where function arguments are stored during external calls (24). Accessing data directly from `calldata` completely bypasses the need for memory storage allocation and copying operations, resulting in substantial gas savings (1).

4.1.2 The keccak256 Hashing for Storage Efficiency. To prevent users from wastefully spending gas uploading the exact same CID

multiple times, the contract implements a strict redundancy check mechanism. However, comparing strings directly in Solidity is notoriously gas-intensive because it requires element-by-element iteration and dynamic memory evaluation (3).
Instead, De-Drive utilizes the `keccak256` cryptographic hash function built into the EVM. By mapping the `keccak256` hash of the CID to a boolean value within a nested Solidity mapping, the contract achieves an $O(1)$ look-up time to verify if a file has already been uploaded by the user (5). While theoretical concerns occasionally arise regarding hash collisions in mappings, the mathematical probability of two different CID strings producing the exact same 256-bit `keccak256` output is infinitesimally small, making it an entirely secure, standard, and highly efficient method for managing decentralized state arrays (5).

4.2 Client-Side Cryptography Mechanics

To satisfy the strict zero-knowledge requirement, data must be encrypted client-side using a proven symmetric key algorithm before it interacts with any network protocols. De-Drive utilizes the `crypto-js` library to implement the Advanced Encryption Standard (AES) with a 256-bit key size (16). Established by the US National Institute of Standards and Technology (NIST) in 2001 (FIPS 197), AES is a substitution-permutation network cipher that remains computationally secure against all known practical brute-force and differential cryptanalysis attacks (16).

4.2.1 The Encryption Pipeline. The transformation of a raw user file into a mathematically secure IPFS payload involves several distinct and sequential operations executed entirely within the browser's DOM:

[leftmargin=*, itemsep=2pt, parsep=0pt]**File Reading and Base64 Conversion:** The native JavaScript `FileReader` API reads the user's file and converts the binary data into a Base64 string. Base64 is essential as it represents arbitrary binary data in a safe ASCII string format, ensuring compatibility with the AES encryption functions which expect string or WordArray inputs (7). **AES-256 Symmetric Encryption:** The Base64 string is passed to `CryptoJS.AES.encrypt()` along with the user's secret passphrase. The library utilizes a key derivation function to stretch the human-readable password into a robust 256-bit key, generates a random Initialization Vector (IV) to ensure semantic security (preventing identical files from producing identical ciphertext), and outputs the encrypted result (7). **Blob Formatting:** Uploading raw Base64 strings directly to IPFS can result in malformed data retrieval and encoding errors. Therefore, De-Drive extracts the ciphertext and converts it into a typed array (`Uint8Array`), which is then instantiated as a JavaScript Blob object representing raw, immutable data (25). **Pinata Transmission:** This encrypted `.enc` Blob is appended to a `FormData` object and transmitted to the Pinata IPFS nodes via an authenticated REST API call (25). Because the encryption occurs locally within the browser memory, neither Pinata, the Internet Service Provider, nor the IPFS swarm can intercept or read the raw file.

4.2.2 The Decryption Pipeline. The retrieval process exactly reverses this pipeline. The encrypted `.enc` Blob is fetched from the IPFS gateway via its public CID. The Blob is converted back into text format using the `FileReader`. `CryptoJS.AES.decrypt()` processes the cipher string using the user-provided passphrase (7). If the key is correct, the function removes the padding and returns the original Base64 string, which is then dynamically reconstructed into a downloadable file link injected into the browser's DOM (25). If an incorrect key is provided, the AES algorithm fails to produce valid PKCS#7 padding, the operation mathematically aborts, and the data remains secure, preserving zero-knowledge integrity.

4.3 The Frontend Bridge and UI Design

The user interface serves as the critical bridge connecting the severe complexities of Web3 and cryptography to the end user. De-Drive is built using React.js and Vite, providing a fast, component-driven architecture. Blockchain interaction is facilitated by the `ethers.js` library, which acts as an RPC (Remote Procedure Call) wrapper, seamlessly communicating with the MetaMask browser extension to manage account states, inject providers, and handle transaction signing flows.
To align with modern Web3 aesthetic standards and communicate a sense of advanced security, De-Drive employs Glassmorphism design principles (23). Glassmorphism is a visual design style characterized by translucent interfaces mimicking frosted glass, which creates visual hierarchy and depth against complex, vibrant backgrounds (17).
The CSS implementation relies heavily on the `backdrop-filter: blur()` property combined with semi-transparent background colors (e.g., `rgba(255, 255, 255, 0.1)`) and subtle border accents to establish layering (21). Careful attention was paid to contrast and accessibility; dark text was prioritized over light glass panels, and structural elements were layered using exact z-index values to ensure the interactive elements (upload zones, retrieve buttons) maintained prominence (23). This design ethos ensures that the highly technical underlying processes—IPFS pinning, EVM state modification, and AES cryptography—are encapsulated within an intuitive, visually frictionless dashboard.

5. RESULTS AND PERFORMANCE ANALYSIS

The ultimate viability of De-Drive as a decentralized alternative to AWS or Google Drive hinges entirely on its economic efficiency and operational performance. The following analysis compares the theoretical cost of storing data directly on the Ethereum blockchain versus utilizing the De-Drive hybrid model, validated by empirical prototype testing.

5.1 EVM Gas Cost Dynamics

Ethereum storage operates as a key-value mapping of slots, each containing exactly 32 bytes. The `SSTORE` opcode, responsible for saving data to the blockchain state, is notoriously the most expensive operation in the EVM. Modifying a storage slot from a zero value to a non-zero value costs 20,000 gas units. Additionally, the transaction data itself incurs a cost. The Ethereum Yellow Paper dictates a baseline fee of 21,000 gas for any standard transaction, plus 16 gas for every non-zero byte of data sent in the transaction payload.

5.2 Comparative Cost Analysis: 2 MB Payload

The viability of decentralized storage for consumer use hinges entirely on its cost-efficiency. To demonstrate the unparalleled advantage of the hybrid model, we analyze the cost of storing a standard high-resolution image file weighing exactly 2 Megabytes (2,097,152 bytes) across two distinct scenarios.
Scenario A: Pure On-Chain Storage (Ethereum EVM). Storing a 2 MB file directly on the EVM requires breaking the raw data into 32-byte words, with each word demanding an individual `SSTORE` operation.

Storage Slots = 2,097,152 bytes $\frac{}{32bytes/slot=65,536slots TotalGas=65,536\times20,000=1,310,720,000gas}$
Assuming conservative mainnet conditions—a gas price of 15 Gwei and an ETH market price of $3,000—the resulting mathematical cost to store a single 2 MB image strictly on-chain is:

$$Cost = 1{,}310{,}720{,}000 \times (15 \times 10^{-9}\ ETH) \times \$3{,}000 \approx \$58{,}982 \quad (1)$$

Pure on-chain storage is thus financially impossible for consumer DApps.

Scenario B: De-Drive Hybrid Model (Empirical Results). In the De-Drive architecture, the heavy 2 MB payload is encrypted locally and stored off-chain on IPFS for fractions of a cent (or entirely free via Pinata's developer tiers). The Ethereum blockchain only receives the immutable Content Identifier (CID) string, which is uniformly 46 bytes in length, regardless of whether the original file was 2 MB or 20 GB.

Live prototype testing on the Sepolia testnet confirmed this efficiency. Because the smart contract only processes a 46-byte string, requiring a maximum of two 32-byte storage slots, empirical testing via MetaMask recorded a flat, consistent network fee of exactly 0.0001 SepoliaETH across multiple uploads.

At a theoretical mainnet equivalent ($3,000/ETH), this translates to a real-world cost of approximately **$0.30 per file**. This represents a **99.99% cost reduction** compared to the pure on-chain storage model.

Table 1. Economic and Performance Comparison (2 MB File, 15 Gwei Gas, $3,000 ETH)

Architectural Metric	**Pure On-Chain Storage (EVM)**	**De-Drive Hybrid Architecture**
Data Stored on EVM State	2,097,152 Bytes	46 Bytes (CID only)
Data Stored Off-Chain	0 Bytes	2,097,152 Bytes (IPFS)
SSTORE Operations Required	65,536 slots	2 slots
Estimated Transaction Cost	≈$58,982.00	0.0001 ETH (≈$0.30)
Privacy Guarantee	None (Public Ledger)	Absolute (Zero-Knowledge AES-256)
Retrieval Speed	Slow (Node Synchronization)	Fast (IPFS Parallel Swarm)

6. CONCLUSION

The persistent and evolving threat landscape surrounding centralized cloud architectures dictates an urgent, industry-wide need for decentralized alternatives that prioritize data sovereignty, immutability, and absolute privacy. The De-Drive platform effectively synthesizes the strengths of three disparate technologies to construct a cohesive, zero-knowledge storage vault that solves the fundamental trilemma of modern data storage.

By leveraging IPFS for decentralized, distributed file hosting, the system circumvents the inherent vulnerabilities, censorship risks, and single points of failure found in Web2 client-server architectures. By utilizing Ethereum smart contracts to map cryptographic CIDs directly to user wallets, De-Drive establishes an unbreakable, tamper-proof chain of custody without incurring the financially catastrophic costs associated with native blockchain storage.

Crucially, the implementation of strict client-side AES-256 cryptography completely resolves the primary limitation of public DHT networks like IPFS—the lack of native content privacy. Because the data is transformed into a cryptographic cipher prior to network transmission, the architecture guarantees absolute confidentiality, successfully achieving a zero-knowledge ecosystem where users do not need to trust the physical storage providers.

Through rigorous gas optimization techniques and the strategic separation of payloads from state ledgers, De-Drive presents a

highly performant, economically viable framework that fulfills the promises of Web3 infrastructure.

7. FUTURE SCOPE

While the current iteration of De-Drive successfully establishes a highly secure, single-user decentralized vault, the reliance on a singular symmetric cryptographic key (AES-256) presents distinct structural limitations. If a user loses their secret passphrase, the encrypted IPFS data is permanently irrecoverable due to the mathematical nature of AES. Furthermore, symmetric encryption heavily restricts collaboration; sharing a secure file necessitates securely transferring the master password out-of-band to the recipient, which inherently compromises the zero-knowledge security model and introduces human error.
Future iterations of the De-Drive architecture must transition from isolated, single-user storage to programmable, decentralized access control. This evolution will focus on two key cryptographic integrations to enhance the platform's utility for enterprise applications.

7.1 Integration of the Lit Protocol for Key Management

To solve the key management vulnerabilities of pure self-custody, future development will integrate the Lit Protocol. The Lit Protocol is a decentralized key management network that enables programmable signing and encryption across a distributed threshold network (13). Rather than relying on a single password, Lit utilizes Distributed Key Generation (DKG) where secret cryptographic material is split into shares across independent network nodes; no single server, entity, or hardware enclave ever holds the complete private key (9).
By integrating the Lit SDK, De-Drive could enable users to encrypt their AES keys with conditional logic encoded directly into smart contracts (9). For example, decryption key shares would only be assembled and returned to the client if the user proves ownership of a specific wallet address, holds a specific NFT, or possesses a specific token balance (Token-Gated Access) (9). This allows for decentralized, human-out-of-the-loop key recovery mechanisms, where a user can recover access to their data simply by signing a cryptographic message from a verified wallet, completely removing the reliance on a memorized plaintext password while maintaining zero-knowledge security (14).

7.2 Asymmetric Cryptography (RSA/ECC) for Secure File Sharing

To facilitate secure file sharing between distinct entities (e.g., a patient sharing a medical record with a healthcare provider), the system must adopt an asymmetric encryption framework—such as RSA or Elliptic Curve Cryptography (ECC)—alongside the existing AES infrastructure (4).
In this proposed update, the heavy file payload remains encrypted via the highly efficient AES-256 algorithm, acting as the symmetric data key (20). However, this symmetric key is then itself encrypted using the intended recipient's public key (asymmetric encryption). The encrypted symmetric key and the IPFS CID are securely transferred to the recipient via the blockchain state. Upon retrieval, the recipient utilizes their private key to decrypt the symmetric key, which is subsequently used to decrypt the heavy IPFS payload locally (20). This digital envelope encryption architecture ensures the absolute privacy of health-related data, financial records, or corporate IP within a public decentralized network while enabling seamless, trustless collaboration across the broader Web3 ecosystem (2).

References

[1] Alchemy. 12 solidity gas optimization techniques. https://www.alchemy.com/overviews/solidity-gas-optimization, 2024. Accessed April 5, 2026.

[2] Anonymous. An architecture for managing data privacy in healthcare with blockchain. *PMC*, 2022.

[3] Anonymous. Calculate transaction costs for storing data. Ethereum Stack Exchange, 2024.

[4] Anonymous. Toward blockchain based electronic health record management with fine-grained attribute based encryption and decentralized storage mechanisms. PMC, 2024.

[5] Anonymous. What are the possible security threats when using mappings in Solidity? Ethereum Stack Exchange, 2024.

[6] Anonymous. A blockchain and IPFS hybrid architecture for secure storage and distributed verification of academic documents. *International Journal of Computer Techniques*, 2026.

[7] Davide Barranca. CryptoJS tutorial for dummies. https://www.davidebarranca.com/2012/10/crypto-js-tutorial-cryptography-for-dummies/, 2012. Accessed April 5, 2026.

[8] Juan Benet. IPFS — content addressed, versioned, P2P file system. arXiv:1407.3561, 2014. https://arxiv.org/abs/1407.3561.

[9] BizThon. Lit protocol: A developer's guide to decentralized access control for token-gated content. Global Business Hackathon, Medium, 2024.

[10] Vitalik Buterin. Ethereum white paper: A next generation smart contract and decentralized application platform. https://ethereum.org/whitepaper/, 2013.

[11] DeSci Nodes. The state of privacy in IPFS. https://nodes.desci.com/dpid/287, 2026. Accessed April 5, 2026.

[12] HIPAA Journal. Healthcare data breach statistics. https://www.hipaajournal.com/healthcare-data-breach-statistics/, 2025. Updated for 2026. Accessed April 5, 2026.

[13] Lit Protocol. Lit protocol: Decentralized key management. https://www.litprotocol.com/, 2024. Accessed April 5, 2026.

[14] Lit Protocol. Powering key recovery with lit protocol. Spark by Lit Protocol, 2024.

[15] Mandiant / Google Cloud. M-Trends 2025: Cybersecurity trends and insights. Technical report, Google Cloud, 2025.

[16] National Institute of Standards and Technology. Advanced encryption standard (AES). Technical Report FIPS PUB 197, NIST, 2001.

[17] Nielsen Norman Group. Glassmorphism: Definition and best practices. https://www.nngroup.com/articles/glassmorphism/, 2024. Accessed April 5, 2026.

[18] Pinata. Why storing data on Ethereum costs as much as a penthouse. https://pinata.cloud/blog/why-storing-data-on-ethereum-costs-as-much-as-a-pentho 2026. Accessed April 5, 2026.

[19] RareSkills. The RareSkills book of solidity gas optimization: 80+ tips. https://rareskills.io/post/gas-optimization, 2024. Accessed April 5, 2026.

[20] Request Network. Handle encryption with a Web3 wallet. Request Network SDK Docs, 2024.

[21] Suryank Singh. Glass-morphism UI implementation in React.Js. Medium, 2024.

[22] D. Trautwein et al. Privacy limitations of IPFS and plausible deniability. IPFS Research, 2024.

[23] UX Pilot. 12 glassmorphism UI features, best practices, and examples. `https://uxpilot.ai/blogs/glassmorphism-ui`, 2024. Accessed April 5, 2026.

[24] YBM. Solidity — storage vs memory vs calldata. Coinmonks, Medium, 2024.

[25] Victor Yeo. Encrypt and upload an image file to IPFS. Chaintope BlogChain, Medium, 2026.

[26] H. Zang et al. Research on an on-chain and off-chain collaborative storage model. *Future Internet*, 18(2):92, 2024.