IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 8, March 2026
Impact Factor: 8.2

A Novel Blockchain-Driven Approach for Decentralized Cloud Storage

Ch. Prudhvi[1], S. Vohita[2], K. Sreeja[3] , Ch. Sirisha[4]
UG Scholars, Department of Computer Science & Engineering[1-3]
Assistant Professor, Department of Computer Science & Engineering[4]
CMR Technical Campus, Hyderabad, India

Abstract: *The exponential growth of digital data has intensified reliance on cloud storage, yet conventional centralized architectures remain persistently vulnerable to unauthorized access, data tampering, and privacy violations. This paper presents a novel blockchain-driven approach for decentralized cloud storage that addresses these concerns through a multi-layered security mechanism. The proposed system fragments a user's file into multiple independent blocks, encrypts each block using the AES-256 algorithm with PBKDF2-derived keys, and distributes them across distinct nodes of the InterPlanetary File System (IPFS). The corresponding IPFS hash addresses are then recorded on an Ethereum-based blockchain through a Solidity smart contract, ensuring immutability and tamper-resistance of the entire storage index. During file retrieval, the system queries the blockchain to collect all block hash addresses, fetches encrypted blocks from IPFS, reassembles them in the correct sequence, and delivers the decrypted output to the user. The implementation is built using Python, Django, Web3.py, and the Truffle/Ganache development environment, and has been functionally verified across all core user-facing modules.*

Keywords*:* Blockchain, Decentralized Storage, IPFS, AES Encryption, Smart Contracts, Ethereum, Data Security, Python, Django

I. INTRODUCTION

The digital universe is expanding at an unprecedented rate. According to Forbes, approximately 2.5 quintillion bytes of data are generated every single day, and over 90 percent of the world's stored data has been produced within the last two years alone [1]. As individuals and organizations generate more data than ever before, cloud storage has emerged as the dominant solution for storing, managing, and accessing information remotely.

However, the prevailing model of cloud storage today is fundamentally centralized. A small number of technology companies, operating large-scale data centers, control the overwhelming majority of stored cloud data. While this arrangement offers genuine convenience, it introduces a range of serious security and privacy risks. In a centralized system, an adversary who successfully compromises the server can gain unrestricted access to all stored data, since there is no architectural separation that limits the blast radius of a single breach.

Beyond direct security threats, centralized cloud storage raises legitimate concerns about data privacy. User data is routinely accessed by service providers for analytics, advertising, and other commercial purposes, frequently without meaningful consent from the data owner. Furthermore, centralized providers typically offer storage on a plan-based pricing model, compelling users to pay for entire storage tiers even when their actual usage falls well below the allocated quota. This inflexibility translates into direct financial inefficiency.

Scalability is a further challenge. As data volumes grow, scaling a centralized infrastructure demands substantial capital investment in additional hardware, networking capacity, and operational staff. This creates barriers to entry and concentrates the benefits of cloud infrastructure among a small number of well-resourced entities.

Blockchain technology offers a fundamentally different architectural foundation. Operating as a distributed, peer-to-peer ledger, a blockchain maintains a shared and continuously verified record of transactions across all participating nodes. Once data is recorded, it becomes computationally infeasible to alter without consensus from the network, providing a strong guarantee of immutability. When combined with IPFS — a peer-to-peer distributed file system that uses content-addressed identifiers to locate and retrieve data — blockchain creates the basis for a genuinely decentralized storage architecture in which no single party controls either the data or the index that describes it.

This paper proposes and implements a system that brings these technologies together in a practical, working prototype. Files uploaded by users are split into blocks, encrypted with AES-256 using a PBKDF2-derived key, stored across IPFS nodes, and indexed immutably on the Ethereum blockchain through a Solidity smart contract. The remainder of this paper is organized as follows: Section II surveys related work; Section III analyzes the existing system and presents the proposed system; Section IV describes the system design; Section V details the implementation; Section VI presents results and discussion; and Section VII concludes the paper.

II. LITERATURE SURVEY

A significant body of research has examined the security challenges of cloud storage and the potential of blockchain and distributed file systems to address them. The works summarized below collectively establish the technical foundation upon which the proposed system is built.

A. Cloud Storage Security

Diao [2] examined data security policies for cloud storage environments and identified that the structural characteristics of centralized cloud systems inherently expose user data to leakage and unauthorized modification. The study proposed a set of security policies derived from analysis of existing academic research, and highlighted the growing frequency of targeted attacks on cloud infrastructure. This work underscores the fundamental inadequacy of relying solely on perimeter-based defenses when the storage architecture itself constitutes a single point of failure.

Lee et al. [3] implemented AES encryption on the Heroku cloud platform and evaluated its effectiveness as a practical data security mechanism. Their performance analysis confirmed that AES is suitable for real-world cloud deployments, while noting that encryption latency increases with file size. This observation directly informs the block-level encryption strategy adopted in the proposed system: splitting files into smaller chunks limits the per-operation encryption overhead and allows the system to remain responsive even for moderately large files.

B. Blockchain as a Trust Infrastructure

Nakamoto [4] introduced Bitcoin and demonstrated that a distributed peer-to-peer network can establish trust and transaction finality without a central intermediary. The core innovation — chaining cryptographic hashes of transaction records to make history tamper-evident — provides the conceptual basis for using blockchain as an immutable index of storage transactions.

Zyskind and Nathan [5] extended this concept to personal data management, proposing a blockchain-based access control system in which users own and govern their own data. Their protocol treats the blockchain as an automated access-control manager that handles instructions to store, query, and share data without requiring trust in any third party. This vision aligns closely with the privacy goals of the proposed system.

C. Decentralized Storage Frameworks

Ruj et al. [8] proposed BlockStore, a decentralized storage framework that uses blockchain to track underutilized storage contributed by network participants. BlockStore employs smart contracts to handle payment and penalty logic, and uses proofs of storage to verify that host nodes are not tampering with stored data. While BlockStore focuses primarily on the economic incentive layer, it confirms the technical viability of blockchain-managed distributed storage at scale.

Benet [9] described IPFS as a peer-to-peer distributed file system that uses content-addressed identifiers derived from the cryptographic hash of each piece of content. Because every identifier is computed from the content itself, IPFS

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 8, March 2026
Impact Factor: 8.2

inherently detects any modification to stored data. The system has no single point of failure and nodes do not need to trust one another, making it well-suited as the storage substrate for a decentralized system.
Li et al. [10] proposed Meta-Key, a data-sharing protocol for blockchain-based decentralized storage that uses proxy re-encryption to enable secure sharing of encrypted data without exposing decryption keys. Their security analysis demonstrated resistance to collusion attacks, addressing a practical concern in multi-user decentralized environments.
Table I summarizes the key contributions of these related works and how they relate to the proposed system.

TABLE I Comparison of Related Works

Reference	Core Focus	Key Technology	Relevance to Proposed Work
Diao [2]	Cloud storage security policies	Cloud computing	Establishes need for stronger storage security
Lee et al. [3]	AES encryption on cloud	AES, Heroku PaaS	Validates AES for practical cloud encryption
Nakamoto [4]	Peer-to-peer trust without intermediary	Blockchain, PoW	Foundation for immutable transaction ledger
Zyskind & Nathan [5]	User-controlled personal data	Blockchain, Access Control	Motivates on-chain access management
Ruj et al. [8]	Decentralized storage with incentives	Blockchain, Smart Contracts	Confirms scalability of blockchain storage index
Benet [9]	Content-addressed distributed file system	IPFS, DHT	Core storage layer used in proposed system
Li et al. [10]	Secure data sharing on decentralized storage	Proxy Re-encryption, Blockchain	Addresses multi-user key management

III. SYSTEM ANALYSIS

A. Existing System and Its Limitations

Existing approaches to secure cloud storage primarily rely on access control mechanisms imposed over centralized storage backends. A representative class of these systems constructs a multiuser access control model in which data owners define policies governing who may read or modify stored datasets. The cloud provider enforces these policies using cryptographic key management to prevent unauthorized access.
While this approach provides a useful layer of control, it suffers from a fundamental architectural weakness: the storage medium itself remains centralized. If the cloud provider is compromised — through an external attack, an insider threat, or legal compulsion — the access control layer offers no protection, since it is the provider who enforces it. The confidentiality and integrity of stored data ultimately depend on trusting a single organization. Additionally, because the storage index and the data coexist on infrastructure controlled by one party, there is no independent mechanism to verify that data has not been silently modified after upload. Accuracy and integrity guarantees are therefore weak.

B. Proposed System

The proposed system eliminates the single point of trust by decoupling the storage index from the stored data and distributing both across independent decentralized networks. The core architecture rests on three pillars: AES-256 encryption for confidentiality, IPFS for distributed data storage, and an Ethereum smart contract for immutable index management.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 8, March 2026
Impact Factor: 8.2

When a user uploads a file, the application calculates the appropriate block size based on total file length. Each block is individually encrypted using AES in Counter mode with a 256-bit PBKDF2-derived key. The encrypted block is added to an IPFS node, which returns a unique hash address. All hash addresses, together with the filename, uploader identity, and upload timestamp, are then recorded on the Ethereum blockchain through a smart contract call. For download, the system retrieves all hash addresses from the blockchain, fetches each block from IPFS, decrypts every block, and reassembles them in order for delivery to the user. Fig. 1 illustrates the overall system architecture.

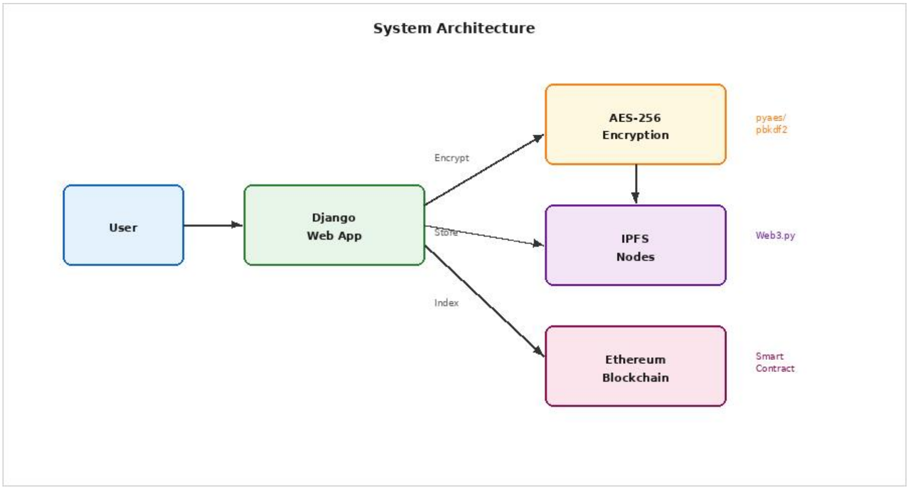

Fig. 1. Overall System Architecture Showing Interaction Between User, Django Web Application, AES Encryption Module, IPFS Storage Layer, and Ethereum Blockchain

C. Advantages of the Proposed System

Because no single node holds the complete file and the storage index exists only on the immutable blockchain, an attacker who gains access to any subset of IPFS nodes cannot reconstruct or tamper with the original file. The system provides confidentiality through encryption, integrity through blockchain immutability, and availability through IPFS distribution. User credentials are also stored on-chain, eliminating the conventional database as a separate attack surface.

D. System Requirements

The hardware and software requirements identified during system analysis are summarized in Tables II and III respectively.

TABLE II Hardware Requirements

Component	**Specification**
Processor	Intel Core i3 or higher
Processor Speed	1.1 GHz minimum
RAM	4 GB minimum
Hard Disk	500 GB minimum
Input Devices	Standard keyboard, two or three button mouse
Display	SVGA monitor

IJARSCT

International Journal of Advanced Research in Science, Communication and Technology

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

ISSN: 2581-9429

Volume 6, Issue 8, March 2026

Impact Factor: 8.2

TABLE III SOFTWARE REQUIREMENTS

Component	Specification
Operating System	Windows 10 or above
Programming Language	Python 3.7
Web Framework	Django
Blockchain Platform	Ethereum (Truffle + Ganache)
Smart Contract Language	Solidity
Distributed Storage	IPFS
Blockchain Interface	Web3.py
Encryption Library	pyaes, pbkdf2
Serialization	pickle

IV. SYSTEM DESIGN

A. Architecture Overview

The system architecture consists of three interconnected layers. The presentation layer is a Django-based web application that handles all user interactions through HTML templates rendered server-side. The business logic layer, implemented in Python, manages file operations, encryption and decryption routines, block size calculation, and communication with external services. The decentralized infrastructure layer comprises two independent networks: the Ethereum blockchain, accessed via Web3.py and a locally deployed Truffle/Ganache environment, and the IPFS network, accessed via the ipfsApi Python client.

The deliberate separation of these layers ensures that the security guarantees of the system do not depend on the integrity of any single layer. Even if the web server were compromised, an attacker would find only encrypted block fragments stored across IPFS nodes and a blockchain record of hash addresses — neither of which is useful without the decryption key and knowledge of which blocks belong together.

B. File Upload Process

The complete upload data flow is depicted in Fig. 2. When a user submits a file through the web interface, the Django view reads the raw file bytes and calls the calculateBlock function to determine the block count and individual block size. The function applies the following logic derived directly from the implementation: files of 1000 bytes or more are divided into 10 equal blocks; files between 500 and 999 bytes are divided into 5 blocks; and files below 500 bytes are divided into 3 blocks. A remainder block is appended whenever the file length is not exactly divisible by the computed block size.

Each block is passed to the encrypt function, which applies AES in Counter mode using a 256-bit key derived via PBKDF2. The ciphertext is serialized with Python’s pickle module and submitted to the local IPFS daemon using add_pyobj, which returns a unique hash code for that block. After all blocks are processed, a single transaction record containing the uploader’s username, the filename, the upload date, all block names, and all hash codes is constructed as a delimited string and written to the Ethereum blockchain via a smart contract transact call.

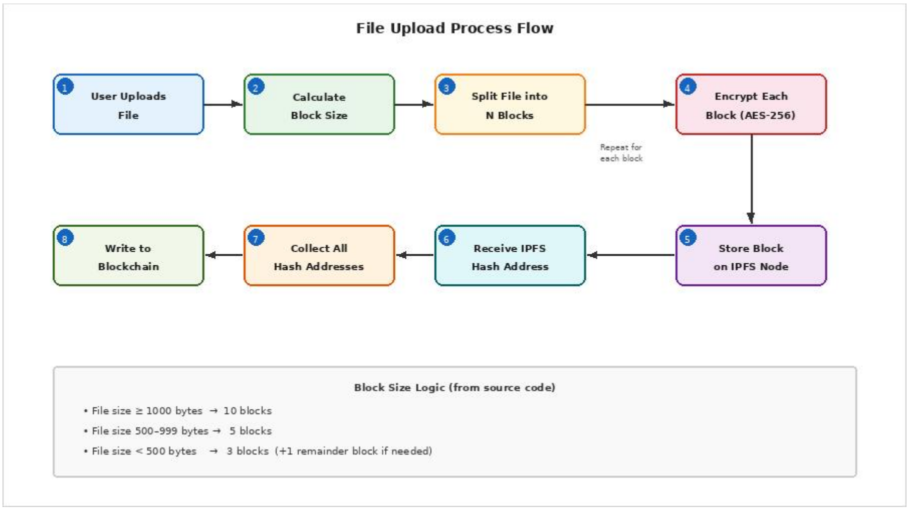
File Upload Process Flow
1 User Uploads File
2 Calculate Block Size
3 Split File into N Blocks
4 Encrypt Each Block (AES-256)
Repeat for each block
5 Store Block on IPFS Node
6 Receive IPFS Hash Address
7 Collect All Hash Addresses
8 Write to Blockchain
Block Size Logic (from source code)
• File size ≥ 1000 bytes → 10 blocks
• File size 500–999 bytes → 5 blocks
• File size < 500 bytes → 3 blocks (+1 remainder block if needed)

Fig. 2. File Upload Process Flow Illustrating Block Fragmentation, AES-256 Encryption, IPFS Storage, and Blockchain Indexing Steps

C. File Download Process

The download process, depicted in Fig. 3, reverses the upload pipeline. The Django view calls readDetails to retrieve all transaction records stored in the smart contract. It scans these records for entries matching both the requesting user's username and the filename. Upon locating the correct record, it extracts the list of block hash codes, iterates through them, calls get_pyobj to retrieve each serialized block from IPFS, deserializes it with pickle, and passes the bytes to the decrypt function. The decrypted bytes are written sequentially to a temporary file, which is then streamed back to the user's browser as an attachment download with the original filename preserved.

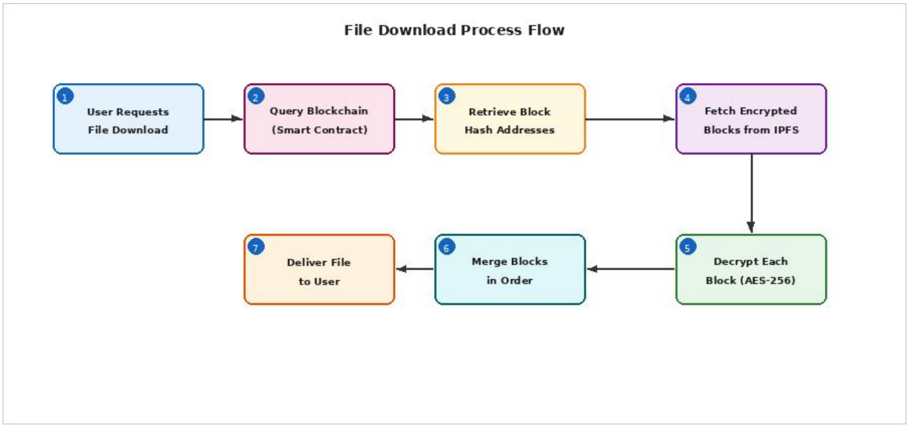
File Download Process Flow
1 User Requests File Download
2 Query Blockchain (Smart Contract)
3 Retrieve Block Hash Addresses
4 Fetch Encrypted Blocks from IPFS
5 Decrypt Each Block (AES-256)
6 Merge Blocks in Order
7 Deliver File to User

Fig. 3. File Download Process Flow Showing Blockchain Query, IPFS Block Retrieval, Decryption, and File Reassembly

D. Block Encryption and IPFS Distribution

Fig. 4 provides a detailed view of the block encryption and distribution process. Each block of the original file is individually encrypted before being uploaded to a separate IPFS node. The IPFS network returns a unique content-addressed hash for every block. These hashes are collected and written together to the Ethereum blockchain as part of a single transaction record. This arrangement means that an attacker who gains access to one or more IPFS nodes can retrieve only encrypted partial blocks — fragments that are both encrypted and incomplete, making reconstruction of the original file computationally infeasible.

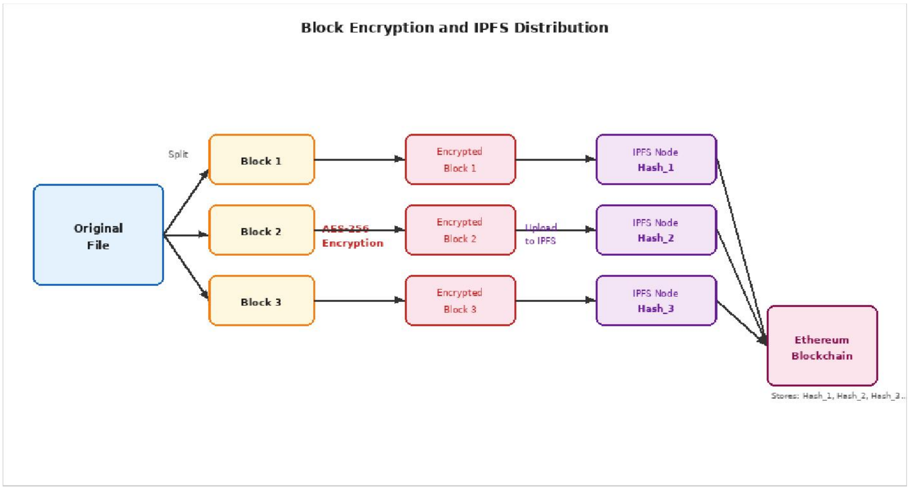

Fig. 4. Block Encryption and IPFS Distribution: Each Block Is Independently Encrypted with AES-256 and Stored at a Distinct IPFS Node; Hash Addresses Are Recorded on the Blockchain

E. Smart Contract Design

The Ethereum smart contract, written in Solidity, serves as the system's immutable registry. It exposes four functions: addClientData stores user registration records; getClientData retrieves all stored user records; addBlockTransaction stores file transaction metadata including all block hash addresses; and getBlockTransaction retrieves all stored transaction records. The contract is deployed on the local Ethereum network using Truffle and assigned a fixed contract address, which is referenced in the Python application to enable consistent communication with the deployed contract instance.

The choice to store user credentials on-chain rather than in a conventional database reflects the system's architectural commitment to decentralization. This approach eliminates the database as a separate attack surface and ensures that user records inherit the same immutability guarantees as file transaction records.

F. UML Diagrams

The system design was modeled using the Unified Modeling Language. The use case diagram identifies five primary interactions between the user and the system: Register, Login, Upload and Encrypt Block File, View File Blocks, and Logout. The sequence diagram captures the message flow between the user, the Django views, the AES encryption module, the IPFS client, and the blockchain smart contract during a complete upload operation. The activity diagram represents the control flow from file selection through block fragmentation, encryption, IPFS storage, and blockchain recording. The deployment diagram illustrates the three runtime environments: the Django web server, the IPFS daemon, and the Ganache blockchain node, each running as a separate local service.

V. IMPLEMENTATION

A. Technology Stack

The backend of the system is implemented in Python 3.7 using the Django web framework. Django was selected for its mature ecosystem, clean request handling, and straightforward template rendering, which keeps the presentation layer clearly separated from the business logic. The Web3.py library provides the Python interface to the Ethereum blockchain, handling connection management, account selection, ABI-based contract loading, and transaction dispatch and receipt waiting.

The Truffle framework manages compilation and deployment of Solidity smart contracts. Ganache provides a local Ethereum blockchain with pre-funded test accounts and a deterministic environment, allowing the complete deployment and interaction cycle to be exercised without connecting to a public network. The IPFS daemon runs locally, and ipfsApi provides the add_pyobj and get_pyobj methods used to store and retrieve serialized block data. Encryption is handled by pyaes, a pure-Python AES implementation, with key derivation performed by the pbkdf2 library.

B. System Modules

The application is organized around five functional modules, each corresponding to a distinct user-facing capability. The registration module accepts user credentials through a web form and writes them to the blockchain via addClientData. The login module authenticates users by calling getClientData, scanning the returned records, and verifying the submitted username and password against stored values.

The upload module handles the complete pipeline: reading the uploaded file, calling calculateBlock, iterating through blocks, encrypting each with AES-CTR, submitting each to IPFS, collecting hash codes, and writing the complete transaction record to the blockchain via addBlockTransaction. The view blocks module calls getBlockTransaction, filters records belonging to the authenticated user, and renders a table showing filenames, block names, and IPFS hash codes. The download module performs the reverse: it reads transaction records, locates the requested file, retrieves and decrypts all blocks from IPFS, and streams the reassembled file to the browser.

C. Key Implementation Details

One notable implementation choice is the adaptive block size calculation. Rather than applying a fixed partition size, the system adjusts the number of blocks to the input file's length: files of 1000 bytes or more become 10 blocks; those between 500 and 999 bytes become 5; and smaller files become 3 blocks. A remainder block captures any bytes left after even division. This strategy ensures that even small files benefit from multi-node distribution while avoiding the overhead of creating too many tiny blocks for large inputs.

The PBKDF2 key derivation function is called with a fixed passphrase and salt to produce a 256-bit AES key. CTR mode is used because it transforms the block cipher into a stream cipher, eliminating the need for padding and making encryption straightforward for arbitrary-length byte sequences. The counter value is fixed across all operations to ensure that decryption always yields the original plaintext given the correct key. In a production deployment, unique per-user keys derived from user-specific secrets would replace the fixed passphrase.

Transaction records stored on the blockchain are formatted as hash-delimited strings: username#filename#date#block_names#hash_codes. Each field is separated by '#' and individual values within fields are space-separated. This simple format is sufficient for the prototype scope, though a production system would benefit from structured Solidity data types to improve query efficiency and reduce gas costs.

D. Development Environment Setup

The system requires three concurrently running services. First, the Ganache blockchain is started by executing runBlockchain.bat, which initializes a local Ethereum network with default accounts and private keys. The smart contract is then deployed using Truffle's migrate command, which compiles the Solidity source and deploys it to the running Ganache instance, producing a contract address that is copied into the Python configuration. Second, the IPFS daemon is started via Start_IPFS.bat, initializing a local IPFS node on port 5001. Third, the Django development server is launched via run.bat, making the application accessible at http://127.0.0.1:8000.

VI. TESTING

A. Testing Strategy

The system was validated through a structured testing process covering module testing, integration testing, and acceptance testing. Module testing was performed on each functional unit in isolation to verify correct behavior independently of other components. Integration testing verified that the modules interact correctly when connected, particularly the handoff between the Django views, the AES encryption functions, the IPFS client calls, and the blockchain transaction functions. Acceptance testing confirmed that the complete system satisfies the original requirements as defined during analysis.

B. Test Cases

Table IV presents the test cases executed during acceptance testing, covering all five user-facing modules. In each case, the actual outcome matched the expected outcome, confirming that the system behaves correctly across all primary user workflows.

TABLE IV ACCEPTANCE TEST CASES

ID	Module	Description	Expected Result	Actual Result	Status	Priority
01	Register	Verify user registration saves record to blockchain	Record stored on-chain; user can proceed to login	Record stored; login succeeds	Pass	High
02	Login	Verify authentication with valid credentials	User granted access to dashboard	Dashboard displayed on correct credentials	Pass	High
03	Upload & Encrypt	Verify file is split, encrypted, stored on IPFS, indexed on blockchain	Block table displayed with hash codes; blockchain transaction recorded	11 blocks generated; all hashes stored on-chain	Pass	High
04	View Blocks	Verify block metadata is retrieved and displayed correctly	Table showing block names and IPFS hash addresses	Correct metadata displayed for each uploaded file	Pass	High
05	Download File	Verify file is retrieved from IPFS, decrypted, and downloaded intact	Original file delivered without corruption	File downloaded and content matches original	Pass	High
06	Logout	Verify session is terminated and dashboard is no longer accessible	User redirected to login page	Session cleared; dashboard inaccessible after logout	Pass	High

C. Security Analysis

The security model of the proposed system provides meaningful protection against several classes of attack. An attacker who compromises a subset of IPFS nodes obtains only encrypted individual blocks. Without possessing all block hash addresses — which are stored on the blockchain, not on IPFS — and the correct decryption key, the attacker cannot reconstruct or read the original file. The spatial separation of the storage index from the stored data provides defense in depth that is absent in centralized architectures.

The use of blockchain as the storage index provides a strong integrity guarantee. Any tampering with the blockchain state would require rewriting the entire chain from the point of modification, which is computationally infeasible in a well-populated Ethereum network. AES-256 in CTR mode provides semantic security under standard cryptographic assumptions, and PBKDF2 key derivation adds resistance to brute-force attacks on the passphrase. Table V compares the security properties of the proposed system against the existing centralized approach.

TABLE V SECURITY PROPERTY COMPARISON: CENTRALIZED VS. PROPOSED SYSTEM

Security Property	Centralized System	Proposed System
Data Encryption	Optional / provider-managed	Mandatory AES-256 per block
Storage Distribution	Single data center	Multiple IPFS nodes
Index Integrity	Provider-controlled database	Immutable blockchain ledger
Single Point of Failure	Yes	No
Tamper Detection	Not guaranteed	Cryptographically guaranteed
User Credential Storage	Centralized database	On-chain (blockchain)
Partial Block Attack	Full file exposed on breach	Encrypted fragments only; file unrecoverable

VII. RESULTS AND DISCUSSION

A. Functional Verification

The system was tested against all six core user-facing modules: registration, login, file upload with encrypted block distribution, block viewing, file download, and logout. In each case the system produced the expected outcome. User registration data was successfully written to and retrieved from the Ethereum smart contract, confirming that the on-chain credential storage functions correctly. Login authentication correctly accepted valid credentials and rejected invalid ones by querying the blockchain-stored user records.

File uploads demonstrated the complete pipeline in operation. A sample file uploaded through the web interface was split into eleven blocks, each individually encrypted and stored at a distinct IPFS node. The upload confirmation screen displayed block names, encrypted byte previews, and IPFS hash addresses for all eleven blocks, confirming that fragmentation, encryption, IPFS storage, and blockchain indexing all completed successfully. The transaction was simultaneously visible in the Ganache interface, confirming on-chain persistence.

The download module successfully reassembled and decrypted a previously uploaded file, confirming that the complete upload-store-retrieve-decrypt lifecycle is coherent and that no data is lost or corrupted during block fragmentation and reassembly. The view blocks module correctly retrieved and rendered the hash address metadata for uploaded files, giving users direct visibility into how their data is distributed across the IPFS network.

B. Discussion

The results confirm that the proposed architecture achieves its primary design goals. By separating the storage index from the stored data and distributing both across independent decentralized networks, the system eliminates the single point of trust that characterizes centralized cloud storage. The combination of AES-256 encryption, IPFS distribution, and blockchain immutability provides layered security that is significantly more robust than conventional centralized approaches.

The prototype implementation operates in a local development environment, which avoids real Ethereum gas costs and IPFS network latency. Transitioning to a public Ethereum network would introduce per-transaction fees that would need to be managed through smart contract optimization. Replacing the fixed PBKDF2 passphrase with per-user keys managed through a secure key management service is the most critical hardening step for a production deployment.

VIII. CONCLUSION

This paper has presented a blockchain-driven approach to decentralized cloud storage that directly addresses the core vulnerabilities of conventional centralized architectures. By fragmenting files into encrypted blocks, distributing those blocks across IPFS nodes, and recording an immutable storage index on an Ethereum smart contract, the proposed system ensures that no single entity holds both the data and the knowledge of where all its pieces reside. Confidentiality is enforced through AES-256 encryption, integrity is guaranteed by blockchain immutability, and availability is supported by the distributed nature of IPFS.

The implementation, built on Python, Django, Web3.py, and the Truffle/Ganache toolchain, demonstrates that this architecture is practically achievable with widely available open-source components. Functional testing and acceptance testing confirmed that all core modules — registration, authentication, upload, block viewing, download, and logout — operate correctly through the complete data lifecycle.

Future work includes deploying the smart contract on a public Ethereum testnet, implementing per-user key management, replacing pickle-based serialization with a platform-neutral format, and optimizing smart contract storage to reduce gas consumption. As decentralized storage solutions mature and transaction costs on blockchain networks continue to decline, architectures of this kind are likely to play an increasingly important role in the future of secure and private data management.

IX. ACKNOWLEDGMENT

The authors would like to thank the faculty members of the Department of Computer Science and Engineering for their guidance and support throughout the development of this project. The authors also acknowledge the open-source communities behind the Ethereum, IPFS, Python, and Django projects, whose tools made this implementation possible.

REFERENCES

[1] B. Marr, "How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read," Forbes, May 2018.

[2] Z. Diao, "Study on Data Security Policy Based on Cloud Storage," in Proc. 2017 IEEE 3rd Int. Conf. Big Data Security on Cloud (BigDataSecurity), IEEE Int. Conf. High Performance and Smart Computing (HPSC), and IEEE Int. Conf. Intelligent Data and Security (IDS), Beijing, China, 2017, pp. 145–149.

[3] B.-H. Lee, E. K. Dewi, and M. F. Wajdi, "Data Security in Cloud Computing Using AES Under HEROKU Cloud," in Proc. 2018 27th Wireless and Optical Communication Conf. (WOCC), Hualien, Taiwan, 2018, pp. 1–5.

[4] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[5] G. Zyskind and O. Nathan, "Decentralizing Privacy: Using Blockchain to Protect Personal Data," in Proc. IEEE Security and Privacy Workshops, San Jose, CA, USA, 2015, pp. 180–184.

[6] C. Cachin, "Architecture of the Hyperledger Blockchain Fabric," in Workshop on Distributed Cryptocurrencies and Consensus Ledgers, Chicago, IL, USA, 2016.

[7] V. Buterin, "A Next-Generation Smart Contract and Decentralized Application Platform," Ethereum White Paper, 2014. [Online]. Available: https://ethereum.org/en/whitepaper/

[8] S. Ruj, M. Stojmenovic, and A. Nayak, "BlockStore: A Secure Decentralized Storage Framework on Blockchain," in Proc. 2018 IEEE 32nd Int. Conf. Advanced Information Networking and Applications (AINA), Krakow, Poland, 2018, pp. 1–8.

[9] J. Benet, "IPFS – Content Addressed, Versioned, P2P File System," arXiv preprint arXiv:1407.3561, 2014.

[10] D. Li et al., "Meta-Key: A Secure Data-Sharing Protocol Under Blockchain-Based Decentralized Storage Architecture," IEEE Networking Letters, vol. 1, no. 1, pp. 30–33, Mar. 2019.

[11] M. Wohrer and U. Zdun, "Smart Contracts: Security Patterns in the Ethereum Ecosystem and Solidity," in Proc. Int. Workshop on Blockchain Oriented Software Engineering (IWBOSE), Campobasso, Italy, 2018, pp. 2–8.

[12] V. Sum, "Security and Privacy Mechanism Using Blockchain," Journal of Ubiquitous Computing and Communication Technologies (UCCT), vol. 1, no. 1, pp. 45–54, 2019.

[13] D. Sivaganesan, "Block Chain Enabled Internet of Things," Journal of Information Technology, vol. 1, no. 1, pp. 1–8, 2019