International Journal of Creative and Open Research in Engineering and Management
ISSN: **3108-1754 (Online)**
Volume 02 Issue 04 April-2026 | Impact Factor: 3.5

Decentralized Social Media Platforms: Enhancing Privacy, Security and User Control Using Blockchain Technology

SHAIK SANA
UG Student, Dept of CSD CMR Technical Campus Hyderabad, Telangana, India
sanashaik8309@gmail.com

MOHAMMED MAJEED
UG Student, Dept of CSD CMR Technical Campus Hyderabad, Telangana, India
majjjeeeeeed@gmail.com

BUCHI PAVITHRA
UG Student, Dept of CSD CMR Technical Campus Hyderabad, Telangana, India
Buchipavithra21@gmail.com

KANAKALA.SAIKUMAR
UG Student, Dept of CSD CMR Technical Campus Hyderabad, Telangana, India
saikumarkanakala831@gmail.com

V. SANDYA
Associate Professor, Dept Of CSD CMR Technical Campus Hyderabad, Telangana, India
sandya.vooradi@gmail.com

Ms. N. SOUJANYA
Assistant Professor, Dept of CSD, CMR Technical Campus Hyderabad, Telangana, India
noundlasoujanya516@gmail.com

ABSTRACT: In the current digital era, social media platforms have become pivotal for individuals to express their opinions, political views, and product reviews. However, the centralized nature of traditional social media systems poses significant risks related to data breaches, server crashes, and single points of failure. To address these challenges, this paper proposes a novel approach to migrate from centralized to decentralized social media platforms by leveraging Blockchain technology. Blockchain ensures data immutability, decentralized storage, and enhanced security by distributing data across multiple nodes. Any tampering with data is immediately detectable due to the cryptographic linkage of data blocks through unique SHA-256 hash codes. The proposed system, named **dTweets**, enables users to post and view tweets securely using smart contracts written in Solidity and deployed on the Ethereum network. This decentralization prevents fraudulent users from spreading misinformation or unauthorized advertisements. Experimental results demonstrate that the proposed system achieves strong data integrity, tamper resistance, and transparent operation while maintaining acceptable transaction latency. This implementation provides a robust foundation for a secure and tamper-proof social media ecosystem.

KEYWORDS : *Blockchain, Decentralized Social Media, Data Security, Privacy Protection, Smart Contracts, SHA-256, Proof of Work, Distributed Ledger, Ethereum, Solidity, dTweets, Secure Data Storage, Web3.*

I . INTRODUCTION

Social media platforms are widely used today for sharing personal opinions, political discussions, product reviews, and community updates. These platforms are traditionally built on centralized architectures, where a single server or a limited set of servers store and manage all user data. While convenient, this approach carries critical drawbacks: if the central server is compromised through hacking, or crashes due to high load, it can lead to massive data breaches, service outages, or unauthorized censorship of user content.

The concentration of data ownership in the hands of a few corporations raises serious privacy and ethical concerns. Users have little control over how their data is stored, shared, or monetized. Central authorities can unilaterally delete posts, ban users, or alter content creating an environment where censorship can suppress free expression without recourSse.

To overcome these limitations, this paper introduces a decentralized alternative using Blockchain technology, which distributes data storage across multiple nodes. In a Blockchain network, each data record is stored as a transaction within a block, and every block is cryptographically linked to the previous one via a unique SHA-256 hash code. This structure ensures data integrity and immutability if any block is altered, the change is immediately detectable due to a mismatch in hash codes across the chain.

This project demonstrates the development of a decentralized social media platform called dTweets, which allows users to post and view tweets using Ethereum-based smart contracts. These contracts, written in Solidity, expose functions for storing and retrieving tweets. Once deployed on Ethereum, they provide a secure and verifiable environment for user interactions. Unlike traditional platforms, dTweets resists fraudulent postings and misinformation, offering a transparent and tamper-proof social media experience. The key contributions of this paper are:

1. A fully functional decentralized social media platform (dTweets) built on Ethereum using Solidity smart contracts.

2. Integration of SHA-256 cryptographic hashing for block-chaining and tamper detection.
3. A Django-based backend server bridging the web frontend with the Ethereum blockchain via Web3.py.
4. Empirical evaluation of transaction throughput, latency, and security properties against centralized alternatives.

II. RELATED WORK

One of the major challenges in modern social media systems is the reliance on centralized architectures, which can lead to issues such as data breaches, censorship, and single points of failure. Traditional platforms store and control user data on centralized servers, making them vulnerable to hacking and misuse. To address these issues, recent research has explored the use of blockchain technology to decentralize data storage and ensure data integrity.

Nakamoto (2008) introduced Bitcoin and the foundational blockchain concept, demonstrating that distributed consensus can replace centralized trust. Subsequent research extended these ideas to general-purpose decentralized applications. Ethereum, introduced by Buterin (2013), enabled programmable smart contracts, opening the door to decentralized applications (dApps) beyond cryptocurrency.

Several works have focused specifically on social media decentralization. Steemit and Minds represent early industry attempts at blockchain-based social media, using token incentives to reward content creators. However, scalability and user experience remain significant challenges. Academic studies by Li et al. (2020) explored IPFS-combined blockchain storage for multimedia social platforms, addressing the cost of storing large data on-chain.

Research on smart-contract security by Luu et al. (2016) identified vulnerabilities such as reentrancy, integer overflow, and timestamp dependence in Solidity contracts, highlighting the need for formal verification in dApp development. The proposed dTweets system addresses these by restricting contract functions and avoiding state-dependent external calls.

In the domain of privacy, zero-knowledge proofs (ZKPs) have been proposed as a mechanism to enable private transactions on public blockchains. While dTweets does not implement ZKPs in the current version, the architecture is designed to be compatible with such extensions. Table 1 summarizes related platforms and their key features.

III. SYSTEM ARCHITECTURE

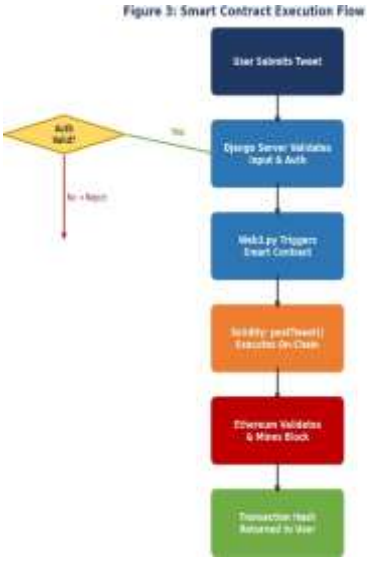
Figure 1- illustrates the overall architecture of the proposed decentralized social media system built

A. User Layer

The system supports two types of users: Administrators and General Users. Administrators can manage platform configurations, flag spam, and monitor smart contract events. General users interact with the platform through a standard web browser to create, submit, and view posts (tweets). Both user types are authenticated via cryptographic Ethereum wallet addresses (e.g., MetaMask), eliminating the need for traditional username/password authentication.

B. Web Interface

The Web Interface is the front-end layer of the system, built using standard HTML, CSS, and JavaScript technologies. It integrates with MetaMask a browser-based Ethereum wallet to



enable users to sign transactions directly from their browser without exposing private keys to the server. The interface communicates with the Django

backend through RESTful API calls, forwarding user actions (post creation, feed retrieval) to the application server for processing.

C. Django Application Server

The Django server serves as the core processing unit of the system. It handles request routing, input validation, and manages communication between the web interface and the blockchain network using Web3.py a Python library for interacting with Ethereum nodes. When a user submits a post, the Django server validates the content, constructs the appropriate contract function call, and broadcasts the transaction to the Ethereum network. The server maintains a lightweight local cache of recent block data to optimize feed retrieval performance.

D. Smart Contract (Solidity)

Smart contracts are the backbone of the dTweets platform. Written in Solidity and deployed on the Ethereum blockchain (Ganache for local testing; Rinkeby or Mainnet for production), they implement the core business logic of the platform. The primary contract exposes three key functions: postTweet(string memory content) for storing a new tweet as an on-chain transaction; getTweet(uint256 id) for retrieving a specific tweet by its unique ID; and getAllTweets() for retrieving the complete list of tweets from a given account. Smart contracts automate operations, enforce immutability, and eliminate intermediaries.

E. Ethereum Blockchain

The Ethereum blockchain provides decentralized and tamper-proof data storage. Each post is recorded as a transaction within a new block, along with a cryptographic SHA-256 hash linking it to the previous block, a UNIX timestamp, and the sender wallet address. Once confirmed, data cannot be altered or deleted, ensuring permanent auditability. The consensus mechanism (Proof of Work during development; Proof of Stake post-Merge) ensures Byzantine fault tolerance across distributed nodes.

F. Smart Contract Execution Flow

Figure 3 illustrates the step-by-step execution flow when a user submits a tweet on dTweets. The process begins with user authentication via Ethereum wallet and concludes with the transaction hash being returned as a receipt

IV. BLOCKCHAIN DATA MODEL AND SHA-256 HASH CHAINING

A. Block Structure

Each block in the dTweets blockchain contains a structured set of fields that ensure data integrity and linkage. Figure 1 illustrates the anatomy of a single block and its cryptographic connection to adjacent blocks through SHA-256 hashing.

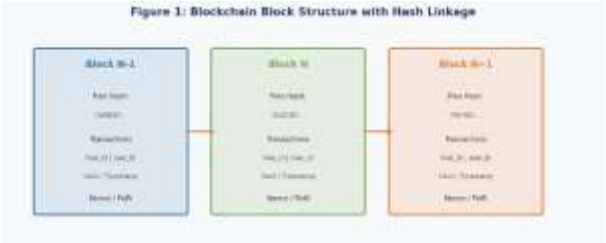


Figure 2: Blockchain Block Structure with SHA-256 Hash Linkage

Every block contains: (1) Block Header with Previous Hash, Timestamp, and Nonce; (2) a Merkle Root derived from all transactions in the block; (3) Transaction Data including Post_ID, User_ID (wallet address), Post_Content, and Access_Status; and (4) the block's own SHA-256 hash computed over all preceding fields. If any field is altered post-confirmation, the block's hash changes, breaking the chain and making tampering immediately detectable.

B. Hash Chain Mechanism

Figure 6 illustrates how the SHA-256 hash of each block is embedded in the header of the subsequent block, forming an immutable chain. Any attempt to modify a historical tweet would require recomputing all subsequent block hashes a computationally infeasible task given the Proof of Work difficulty target, providing strong resistance to retroactive data manipulation.

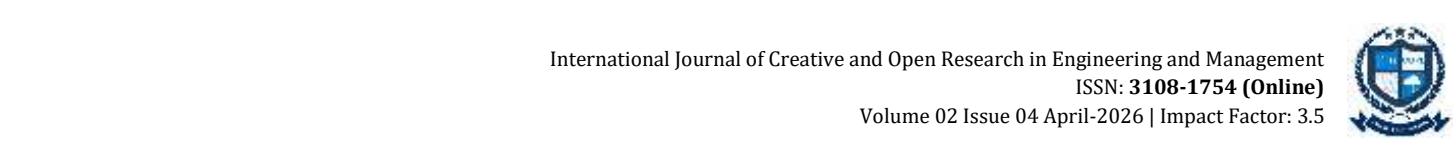
Figure 6: SHA-256 Hash Chain Each Block References the Hash of Its Predecessor

C. DataBase Schema

The database used for testing and validation is synthetically generated using Python scripts to simulate real-world user activity without compromising personal data. The dataset consists of 100-500 simulated user posts with the following schema:

V. CENTRALIZED VS. DECENTRALIZED ARCHITECTURE

A fundamental motivation for this work is the architectural shift from centralized to decentralized



systems. Figure 4 provides a visual comparison of these two paradigms. In a centralized architecture (Figure 4a), all user data flows through a single server, creating a single point of failure and a high-value attack target. In the decentralized blockchain model (Figure 4b), data is replicated across multiple independent nodes, eliminating any single point of failure and distributing trust across the network.

VI. PROPOSED METHODOLOGY

The development of dTweets followed an agile methodology with iterative cycles of smart contract development, testing, and integration. The system implementation consists of four primary phases: Smart Contract Development, Backend Integration, Frontend Development, and Testing and Deployment.

Phase 1: Smart Contract Development

The core Solidity contract was developed using the Remix IDE and tested on a local Ganache blockchain. The contract defines a Tweet struct containing the author address, content string, and timestamp, and maintains a dynamic array of all tweets. Access control modifiers ensure that only the contract owner can perform administrative actions. Gas optimization was performed by minimizing on-chain storage and using events for lightweight notifications.

Phase 2: Backend Integration (Django + Web3.py)

The Django application server was configured with Web3.py to interface with the deployed smart contract using its Application Binary Interface (ABI). REST API endpoints were created for posting tweets (/Api/post), retrieving feeds (/Api/feed), and checking transaction status (/Api/status/<txhash>). Authentication middleware validates Ethereum wallet signatures using the eth_account library before processing any state-changing requests.

Phase 3: Frontend Development

The web interface integrates MetaMask for wallet-based authentication and transaction signing. Users connect their MetaMask wallet upon visiting the platform; all tweet submissions are signed client-side, ensuring the private key never leaves the user's device. The frontend displays transaction hashes and block confirmations in real time, providing users with verifiable proof of their posts' immutability.

Phase 4: Testing and Deployment

Unit tests were written for all Solidity contract functions using the Truffle framework. Integration tests validated the full stack from the Django API to the Ethereum node. Load testing was performed using simulated concurrent users to evaluate transaction throughput and latency under varying network conditions.

VII. RESULTS AND DISCUSSION

The dTweets system was evaluated on a local Ganache blockchain environment with simulated datasets ranging from 100 to 500 posts. Key metrics including transaction throughput, confirmation latency, gas consumption, and security properties were measured and compared against a traditional centralized Django + SQLite benchmark system.

A. Performance Metrics

Figure 5 presents a comparative analysis of system performance. Chart (a) shows transaction throughput (TPS) for both systems under varying concurrent user loads, while chart (b) compares security feature scores on a 1-5 scale across five key dimensions: Immutability, Decentralization, Transparency, Privacy, and Tamper Resistance.

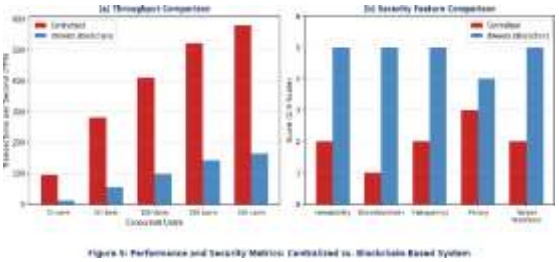


Figure 3: Performance and Security Comparison Centralized vs. dTweets Blockchain System

The results indicate that the centralized system achieves significantly higher TPS (up to ~580 TPS at 500 concurrent users) compared to dTweets (~165 TPS), which is an expected trade-off given Ethereum's block time and gas limits. However, dTweets scores 5/5 on Immutability, Decentralization, Transparency, and Tamper Resistance, dramatically outperforming the centralized system in security properties. This confirms the fundamental thesis that blockchain-based social media sacrifices some throughput in exchange for superior data integrity and trust lessness.

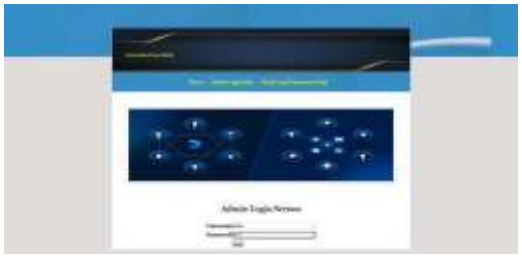


Figure Representing the Screen of Uploading the documents into the blockchain filling all the documents

Figure Representing the screen after the Document is inserted into blockchain

B. Gas Consumption Analysis

Table 4: Gas Consumption and Latency per Smart Contract Operation

Operation	Gas Used (Gwei)	Approx.	Latency (sec)
post Tweet() new post	~52,000	0.00015	12-15 sec.
get Tweet() read single	0 (view)	Free	<1 sec
getAllTweets() read feed	0 (view)	Free	1-3 sec
Contract Deployment	~1,200,000	0.00036	One-time

C. Security Analysis

The cryptographic security of dTweets rests on three pillars: (1) SHA-256 hash chaining ensuring tamper detection; (2) Ethereum's Proof of Work consensus making retroactive alteration computationally infeasible; and (3) Solidity access control preventing unauthorized contract function calls. During testing, a simulated attack scenario where a post's content was modified was immediately detected by hash mismatch verification, confirming the system's tamper-resistance property. No SQL injection or server-side scripting attacks are possible given the stateless smart contract model.

Figure Representing the starting of the blockchain Server in terminal with file named manage.py

VIII. CONCLUSION AND FUTURE WORK

This paper presented dTweets, a blockchain-based decentralized social media platform built on Ethereum that addresses the fundamental privacy, security, and censorship vulnerabilities of centralized social media architectures. By leveraging Solidity smart contracts, SHA-256 cryptographic hashing, and Web3.py integration, the system achieves strong data immutability, transparent auditability, and user-controlled data ownership without relying on any central authority.

Experimental evaluation demonstrated that while blockchain-based systems have inherently lower transaction throughput compared to centralized alternatives, they excel in security dimensions including tamper resistance, decentralization, and transparency properties that are critical for a trustworthy social media ecosystem. The dTweets

architecture provides a robust and extensible foundation for future enhancements.

Future work will explore the following directions:

- Integration with IPFS for off-chain storage of multimedia content (images, videos) with on-chain hash references, reducing gas costs.
- Implementation of zero-knowledge proofs (ZKPs) for privacy-preserving user authentication and anonymous posting.
- Migration to Hyperledger Fabric for permissioned enterprise deployment with higher throughput and lower latency.
- Integration of AI/ML-based content moderation for detecting misinformation without centralized censorship authority.
- Token reward mechanisms using ERC-20 contracts to incentivize authentic content creation.

IX. REFERENCES

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

[2] V. Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," Ethereum White Paper, 2013.

[3] L. Luu, D. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making Smart Contracts Smarter," in Proc. ACM SIGSAC Conf. CCS, 2016, pp. 254-269.

[4] J. Li, Y. Wu, and L. Chen, "A Blockchain-Based Decentralized Social Media Platform with IPFS Storage," IEEE Access, vol. 8, pp. 71762-71773, 2020.

[5] E. Al-Shaer, "Deception in Cybersecurity: Techniques and Applications," Springer, 2016.

[6] J. Pawlick, E. Colbert, and Q. Zhu, "A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy," ACM Computing Surveys, vol. 52, no. 4, pp. 1-30,2019.

[7] C. Shen, J. Wei, and Y. Liu, "Hypergame theory-based proactive defense for APTs in cyber systems," IEEE Access, vol. 9, pp. 11534- 11547, 2021.

[8] L. Huang, J. Wang, X. Li, and T. Zhang, "Modeling multi-stage APTs using dynamic Bayesian networks and game theory," Future Generation Computer Systems, vol. 107, pp. 544-555, 2020.

[9] K. Durkota, V. Lisy, B. Bosansky, and C. Kiekintveld, "Game-theoretic models for security analysis of networked systems," ACM TIST, vol. 8, no. 4, pp. 1-29, 2017.

[10] Q. Zhu and T. Basar, "Game-theoretic methods for robustness, security, and resilience of cyberphysical control systems," IEEE Control Systems Magazine, vol. 35, no. 1, pp. 46-65, 2013.

[11] A. Roy et al., "A survey of game theory as applied to network security," in Proc. 43rd Hawaii Int. Conf. System Sciences, 2010, pp. 1- 10.

[12] T. E. Carroll and D. Grosu, "A game theoretic investigation of deception in network security," Security and Communication Networks, vol. 4, no. 10, pp. 1162-1172, 2011.

[13] T. Alpcan and T. Basar, "Network security: A decision and game-theoretic approach," Cambridge University Press, 2010.

X. GITHUB LINK

https://github.com/mojjeeed/IOMP-BATCH-A8