2026 IEEE/ACM International Symposium on Quality of Service (IWQoS) | 979-8-3195-1703-6/26/$31.00 ©2026 IEEE | DOI: 10.1109/IWQoS70441.2026.11661029

Desi: Revisiting Signature Verification in Blockchain-based Storage System

Ziwei Li*, Songsong Xu†, Tong Li‡‖, Xiaoliang Wang¶*, Bo Wu§,
Yangfei Guo *, Shenglin Jiang*, and Ke Xu*†
Zhongguancun Laboratory*, Tsinghua University†, Renmin University of China‡, Tencent§,
Capital Normal University¶, State Key Laboratory of Internet Architecture‖

***Abstract*—Blockchain-based storage systems are fundamentally limited in write throughput due to the computational overhead of signature verification during consensus. To overcome this bottleneck, we propose Desi, a novel framework that optimizes write transaction processing. In Desi, storage nodes remove individual signatures from incoming write requests and cache the simplified transactions. Once a batch reaches a predefined target condition, a Merkle tree is constructed over the batch, and the root hash is signed to form a root transaction that participates in consensus. To guarantee timely and accurate query results, storage nodes can combine cached write information with on-chain data, preventing delayed writes from affecting reads. To defend against potential malicious behavior by storage nodes, Desi incorporates a challenge-and-response mechanism, enabling verifiable accountability and user compensation. Experimental results demonstrate that Desi achieves 2.8–3.8x higher write throughput compared to the baseline mechanism, while maintaining security and verifiability.**

***Index Terms*—Blockchain-based Storage, Signature Verification, Incentive and Punishment**

I. Introduction

Blockchain was originally proposed as a distributed and decentralized public ledger, in which each node stores the ledger locally and achieves the ledger consistency based on the Byzantine fault tolerant(BFT) consensus. In recent years, blockchain technology has developed rapidly and expanded to identity management [1], [2], data management [3], [4], network security [5], [6] and collaborative storage [7], [8]. These applications demand a substantially larger volume of stored data than traditional cryptocurrency scenarios, motivating extensive research on blockchain-based storage solutions.

The main idea of blockchain-based storage is to combine the blockchain with the authenticated data structure (ADS) [9]. Instead of storing all data on-chain, the blockchain maintains lightweight metadata such as hashes and indices to ensure the authenticity and verifiability of off-chain data, thereby significantly reducing storage overhead. When a user queries data, storage nodes can aggregate the ADS stored on-chain to construct integrity proofs, enabling efficient verification without requiring each read operation to undergo consensus. Consequently, this approach greatly improves read performance compared to traditional blockchain systems.

In recent years, prior research has focused on optimizing read efficiency [10], [11], the handling of write transactions has received limited attention. Each write transaction typically involves three steps: signature verification, order transactions, and execution transactions. Among these, signature verification is a major performance bottleneck. For example, a computer with 16GB of memory and the Apple M1 Pro chip (8-core CPU) can update Merkle Patricia Tree (MPT) up to one million times per second, yet can only verify approximately $10,000$ 256-bit ECDSA signatures per second. Existing BFT protocols, in contrast, can order several hundreds of thousands of transactions per second, highlighting a clear mismatch between consensus throughput and verification capacity.

To address this bottleneck, we propose Desi, a novel framework that optimizes the processing of write transactions. In Desi, storage nodes receive write requests from clients and remove individual signatures, caching the simplified transactions. Once a batch reaches a certain size or a configurable time interval elapses (e.g., 10 seconds), the system constructs a Merkle tree over the batch, and the root hash is signed to form a root transaction that participates in consensus. This approach reduces the number of signature verifications by a factor of m, where m is the number of transactions per root. To prevent delayed writes from affecting data queries, the system allows storage nodes to generate up-to-date query results using cached write information combined with on-chain data.

The removal of individual signatures introduces a potential vulnerability: a storage node could forge a transaction to participate in consensus and manipulate data. To mitigate this, Desi introduces a lightweight malicious behavior detection mechanism, where clients assign incremental sequence numbers to transactions. Any attempt by a storage node to forge a transaction is immediately detectable, as subsequent client transactions will fail to participate in consensus normally. Further, a public blockchain platform provides comprehensive punishment mechanisms, including fund deposits, subscription services, challenges, responses, and compensation claims. Malicious nodes that fail to respond appropriately are penalized, with penalties automatically transferred to affected users.

This work was supported in part by the National Science Foundation for Distinguished Young Scholars of China under No. 62425201, the National Natural Science Foundation of China (No 62572473, No. 62441230), the funding from the State Key Laboratory of Internet Architecture (No.HLW2025ZD17), the Scientific Research Innovation Capability Support Project for Young Faculty (No. SRICSPYF-ZY2025001) and in part by the Zhongguancun Laboratory. Ziwei Li and Songsong Xu contributed equally. Xiaoliang Wang is the corresponding author (wangxiaoliang@cnu.edu.cn).

979-8-3195-1703-6/26/$31.00 © 2026 IEEE

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:13:45 UTC from IEEE Xplore. Restrictions apply.

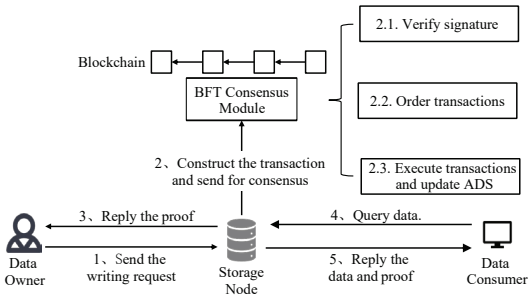


Fig. 1: Blockchain-based storage system.

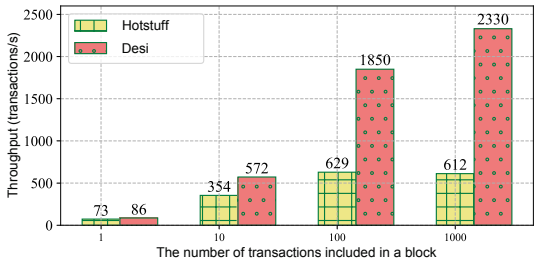


Fig. 2: The throughput improvement when removing all signatures in transactions.

In summary, Desi is a write transaction optimization framework for blockchain-based storage systems that significantly reduces signature verification overhead by aggregating multiple write requests into a single root transaction, thereby substantially improving write throughput without affecting query correctness or freshness. To ensure security and accountability, it introduces a lightweight mechanism to detect and penalize malicious storage nodes, enabling users to submit verifiable challenges to a third-party blockchain platform for compensation, which guarantees incentive compatibility with negligible overhead. We implement a full prototype and conduct extensive evaluations, showing that Desi achieves 2.8–3.8× higher throughput than the baseline (i.e., HotStuff [12]), demonstrating the practicality and effectiveness of our approach.

The remainder of this paper is organized as follows. §II introduces background and motivation of Desi. We first describe the framework of Desi in §III, and then elaborate on the design of Desi's two sub-schemes, root-based signature compression scheme in §III-C and lightweight accountability scheme in §III-D, respectively. §IV give the evaluation results of Desi. §V shows the related work and §VI concludes this paper.

II. Background and Motivation

A. Background

Classic Byzantine fault-tolerant consensus. Unlike the blockchain-derived consensus protocols that rely on mining, such as Proof of Work (PoW) used by Bitcoin, the classic BFT protocol is more environmentally friendly since it relies on the Quorum mechanism [12]. Classic BFT consensus aims to make a set of possibly faulty distributed replicas execute the commands in the same order, maintaining a consistent state. When designing such a protocol, two properties should be ensured. (1) *Safety.* All correct nodes commit and execute commands in a consistent total order. (2) *Liveness.* All correct nodes eventually commit and execute newly proposed commands. With the properties of safety and liveness, BFT consensus can ensure the blockchain integrity when the number of byzantine nodes is less than $1/3$.

Blockchain-based storage. Blockchain-based storage is to combine the blockchain with an ADS. Blockchain only maintains and stores the ADS, while the real data with high storage overhead is stored locally by the storage nodes. To explain it more clearly, we take HotStuff as an example. As shown in Fig 1, the storage nodes can process the query transactions locally and provide proof based on the ADS, thus the throughput of the blockchain will not restrict the reading throughput of the entire storage system. However, all writing transactions from data owners will be broadcast by storage nodes for consensus, by which all storage nodes will get a consistent ADS. Therefore, the throughput of the blockchain will restrict the writing throughput of the entire storage system.

B. Motivation

Signature verification is the bottleneck of blockchain-based systems. In blockchain-based storage systems, each write transaction typically involves three stages: i) Verifying signature, ii) Ordering transactions, and iii) executing transactions and updating ADS. While prior work has extensively optimized the latter two stages to improve write throughput, the first and fundamental bottleneck signature verification has largely been overlooked. During consensus, all storage nodes are required to verify the signatures of write transactions, which imposes substantial computational overhead and severely constrains system throughput. In practice, even when modern BFT protocols can order hundreds of thousands of transactions per second, the overall write throughput remains bounded by the much lower signature verification capacity of individual nodes. Hence, we seek to redesign the write path by decoupling signature verification from per-transaction processing and minimizing redundant verification work across nodes. This motivates the design of Desi.

Design rationale of Desi. The design rationale of Desi is inspired by Trust-Control Theory from social science, which characterizes the fundamental trade-off between fully trusting and fully distrustful societies. A fully trusting system enables highly efficient operations but is inherently vulnerable to misbehavior, while a fully distrustful system relies on strict monitoring and verification to prevent violations, at the cost of significantly reduced efficiency. Desi adopts a principled balance between these two extremes by combining trust, in the form of batch verification over aggregated write requests, with control, in the form of lightweight challenge-based detection and penalty mechanisms. This design achieves high throughput and accountable security, overcoming the performance constraints caused by strong per-transaction verification in existing blockchain-based off-chain storage systems.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:13:45 UTC from IEEE Xplore. Restrictions apply.

Potential benefit. Our motivation is further supported by empirical observations. Figure 2 reports the performance of the HotStuff with ten nodes deployed on a single machine. The results show that system throughput improves substantially when signature verification is removed from the critical path of each transaction. Moreover, this performance gain becomes increasingly pronounced as the number of transactions per block grows, indicating that signature verification, rather than transaction ordering, is the dominant bottleneck in high-throughput settings. These results corroborate the necessity of redesigning the write path to amortize cryptographic verification costs. Motivated by this, Desi adopts a batch-oriented verification design that aggregates multiple write requests into a verifiable root transaction, substantially reducing verification overhead while preserving security and accountability.

III. Desi Overview

A. The Desi Framework

As shown in Fig 3, Desi involves four types of participants: data owners, storage nodes, data consumers, and a public blockchain with smart contract support. Independent storage service providers deploy storage nodes. They jointly maintain a storage blockchain using a BFT consensus protocol and employ an MPT as the ADS to ensure the consistency and verifiability of the storage state.

Data owners submit write requests to storage nodes to update their data. Storage nodes package these write requests into write transactions and participate in consensus, ensuring that all storage nodes consistently update their local MPTs. Data consumers retrieve data from storage nodes and verify the integrity and freshness of query results using the proofs generated from the ADS. The public blockchain, equipped with smart contracts, is leveraged to implement incentive and punishment mechanisms. When malicious behaviors are detected, users can submit challenges to smart contracts and claim compensation accordingly.

Desi is built upon two tightly coupled mechanisms: Root-based Signature Compression (RSC) and Lightweight Accountability (LA). RSC is responsible for reducing cryptographic verification overhead in the write path. It aggregates multiple signed write requests into a batch, removes individual client signatures, and constructs a Merkle tree over the batch. The storage node signs only the Merkle root to generate a root transaction that participates in consensus. Other storage nodes verify a single root signature instead of verifying signatures for all individual write requests, thereby amortizing the cost of signature verification across a batch of transactions.

LA is responsible for ensuring security and accountability under compressed verification. Since individual signatures are removed before consensus, LA provides a lightweight mechanism to detect forgery, tampering, and omission of write requests by malicious storage nodes. It further enforces accountability by allowing users to submit verifiable challenges to smart contracts on a third-party public blockchain, which triggers penalties and compensation, thereby incentivizing storage nodes to behave honestly.

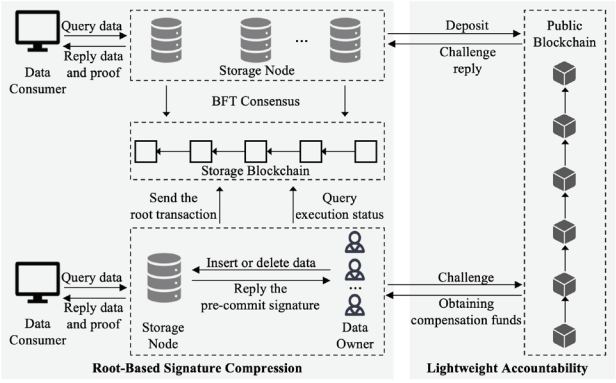


Fig. 3: The framework of Desi.

This paper focuses on optimizing the processing for transactions involving write operations, thus only considering a simple key-value storage application. It should be noted that our scheme can be extended to any other storage applications that require high writing throughput.

B. Threat Model and Assumptions

We assume either the data owner, storage node, or data consumer could be malicious and may modify the data to their benefit. The storage nodes are incentivized to reply to any query access or update request honestly to continuously gain profit because all data owners and query clients are customers of the storage nodes. The storage nodes may be malicious to modify the data and results to their benefit but are incentivized to provide replies for all their customers, or the customers will select another storage node as the service provider. Furthermore, in the consensus for maintaining the storage blockchain, we assume that less than $1/3$ of the storage nodes could be malicious (i.e., express Byzantine behavior), which is a requirement to reach a consensus for BFT protocols with the presence of Byzantine nodes.

Since Desi commits designed write transactions to the blockchain in a batched manner, storage nodes may conduct the following classes of malicious behaviors:

- **Transaction fabrication:** uploading forged write transactions to the blockchain when the data owner did not issue any update.
- **Transaction tampering:** modifying the content of legitimate write requests before commitment.

These attacks threaten the integrity and completeness of the write history, and directly target the performance–security trade-off that motivates Desi: removing per-transaction signature verification from the critical path improves throughput but weakens immediate verifiability.

We assume standard cryptographic primitives are secure. A Public Key Infrastructure (PKI) is assumed to exist for distributing authentication keys, and all nodes know each other's public keys. We do not consider side-channel attacks or compromises of cryptographic primitives.

C. Design of Root-Based Signature Compression

The RSC mechanism is a core component of Desi, enabling efficient write processing in the storage system. It facilitates

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:13:45 UTC from IEEE Xplore. Restrictions apply.

the batching and compression of transaction signatures, reducing verification overhead while preserving data integrity. The mechanism governs interactions among data owners, storage nodes, and data consumers.

In Desi, storage nodes execute BFT consensus and maintain the storage blockchain, which records both the transactions and the ADS required to prove the integrity of all queried data. Each data object is represented as a key-value pair $(key, data)$, where the key uniquely identifies the object. Data consumers query requests are processed locally by storage nodes, leveraging cached and on-chain ADS to provide up-to-date integrity proofs. Write requests from data owners, in contrast, are encapsulated as transactions and submitted for consensus.

In the following, we provide an overview of the interaction protocols between data owners, storage nodes, and data consumers, highlighting how RSC mechanism enables high-throughput transaction processing.

The storage blockchain. Storage blockchain is maintained by all storage nodes, who run the BFT consensus and maintain a consistent ledger. The ledger records the status of all write transactions and provides access and verification for data owners. Besides, the ledger also records the root of MPT, which is stored by all storage nodes to provide integrity proof for query data.

Every storage node locally stores an MPT tree, which records the hash of data, the data owner ID, and the sequence number of the transaction that recently performed update operations on the data. When executing a transaction from the data owner id_o to update a $(key, H(data))$ pair, the storage node will get the sequence number seq in the transaction, construct the $68bytes\ value$ by combining id_o, $H(data)$, and seq to a byte array, i.e., $(H(data)||id_o||seq)$, and finally using $(key, value)$ to update the data MPT. When have finished the execution of all transactions in the block, storage nodes need to store the transaction status and the related MKT locally.

Every storage node locally maintains the information of all data owners, including the 32 bytes identification (id), the latest transaction sequence number ($ltseq$), the public key (pk), and the id of the authorization storage node (i.e., aid) which can send simplified writing transactions. The data owner can purchase the storage service from storage nodes and join the system. When participating in the storage system, the data owner can send a participation request to any storage node (usually the node from which the data owner purchases the storage service) to join the storage system.

Data writing transaction protocol. Once a data owner has joined the system, they can send a writing transaction (WT) to the storage nodes for consensus. Additionally, a data owner can submit an authorization transaction (AT) to authorize storage nodes to send a simplified transaction (ST) for consensus. Each data owner will locally store a counter (denoted as cnt), which is used to set the sequence number of all transactions. When joining the storage system, the data owner will initialize cnt as 1. Whenever a data owner sends a new WT or AT, it will set the sequence number to be local cnt and set cnt to be $cnt + 1$. When the ST satisfies the predefined criteria, storage nodes can construct the root transaction (RT), and each RT aggregates rts STs.

Data Querying Transaction Protocol. When querying data, the data consumer will send a query request to the storage node based on the keyword key of the data. After receiving the request, the storage node first verifies the signature of the transaction. After verification, the key is encoded as a Key in the MPT tree and queried. If the data exists, a Merkel proof is constructed based on the MPT tree. Then, the data hash value $H(data)$ and the identity list of the storage node $nodeList$ are returned to the user.

D. Design of Lightweight Accountability

The lightweight accountability scheme is introduced to deter storage nodes from forging simplified transactions. The mechanism is constructed based on a public chain that supports digital currency and smart contracts, e.g., Ethereum. When joining the system, storage nodes need to pledge funds to a public blockchain to build basic trust, and a fund-management contract is constructed to manage all the mortgaged funds. Clients can query the public blockchain for mortgage balance and historical malicious behavior of all storage nodes. When the data owner believes that the storage node has forged a data modification transaction, it can pose a challenge to a certain data change. When the storage cannot send a valid ACK message to the contract within a certain period, corresponding financial penalties will be imposed. The following introduces the details of the lightweight accountability scheme.

The User Account. In Desi system, we store users' essential information in both the storage chain and smart contracts to ensure the smooth functioning of this system. The contract utilizes the UserInfo structure to store user information, which is then stored on the chain through the global mapping id2userinfo $<$ bytes $\rightarrow$ UserInfo $>$.

Deposit and Revoke. The contract offers deposit and revokes services to storage nodes, allowing them to secure their funds before utilizing the storage system. In cases where a user node engages in malicious behavior during subsequent transaction storage, a portion of the mortgaged funds can serve as compensation to the victim, thus penalizing the perpetrator.

Subscribe Service. When both parties involved in the transaction are registered in the contract's UserInfo, they become eligible to access the system and utilize its services for storing their transactions. To facilitate this functionality, we primarily utilize the Subscribe interface in the contract, while employing the SubInfo structure to record and store all relevant transaction information accurately.

Challenge and Ack. When the transaction initiator node detects incorrect or false transaction information stored by the storage node, it can initiate a challenge using the Challenge interface. This challenge will create challenge information within the contract, which will be stored in the ChallengeInfo structure. The party being challenged must respond within the specified time by calling the Ack interface. If there is no response within the specified time, the challenger is entitled

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:13:45 UTC from IEEE Xplore. Restrictions apply.

to receive the funds requested in the transaction (including the transaction amount and penalty funds) using the Declare interface.

Quick Challenge. The challenge mechanism can effectively help data owners verify the correctness of their data in the contract, but only if the transaction has been uploaded to the contract. In the actual design of the signature storage system, the storage node delays uploading each transaction creation to the contract to maximize the throughput of write operations. Instead, upon receiving a write request, it promises the transaction initiator that the transaction will be packaged sequentially and uploaded to the contract along with other transactions. However, there is a risk of malicious storage nodes making false promises and not uploading transactions. To address this issue, we implemented a quick challenge mechanism.

When the contract receives a quick challenge request, it will verify whether the transaction sequence number on the current chain is greater than the transaction sequence number included in the quick challenge request. If this is true and the transaction does not exist on the chain, it means that the data owner's challenge was successful. In this case, the data owner can choose to declare liquidated damages directly without waiting for a response from any node.

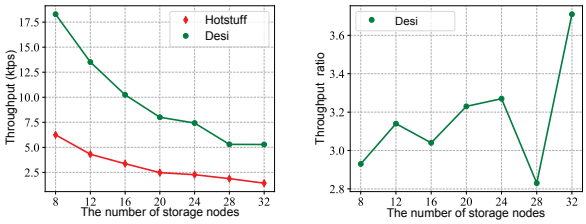
IV. Evaluation

A. Experiment Setup

To evaluate the effectiveness of Desi in enhancing blockchain-based storage systems, we implemented a prototype on four cloud servers. All storage nodes and clients were deployed across these four ECS g7.xlarge instances. Each server is equipped with an Intel Xeon (Cascade Lake) Platinum 8269CY processor with 4vCPUs and a maximum network bandwidth of 100 Mbps. The servers are located in four different geographic regions to emulate a distributed deployment.

We implemented prototypes for both the RSC storage blockchain system and the LA system. The RSC prototype is built on the HotStuff codebase, with substantial modifications to the client and execution modules. In this system, the client acts as the data owner, sending write transaction requests and awaiting final confirmation from storage nodes. If verification succeeds, the execution result is accepted. Otherwise, the client can issue a challenge through a public blockchain smart contract. Storage nodes wait for incoming transactions from clients and construct a root transaction for consensus once the number of cached transactions reaches the preset rts. After consensus, each individual write transaction is executed.

The LA prototype is deployed on the Ethereum public blockchain and comprises three data structures—UserInfo, SubInfo, and ChallengeInfo—as well as six function interfaces: Deposit, Revoke, Subscribe, Challenge, Ack, and Declare. The smart contract was implemented using Remix Solidity 0.8.1 and deployed for experiments on the Sepolia Ethereum testnet.



(a) Throughput (b) Throughput ratio

Fig. 4: The throughput improvement brought by Desi mechanism.

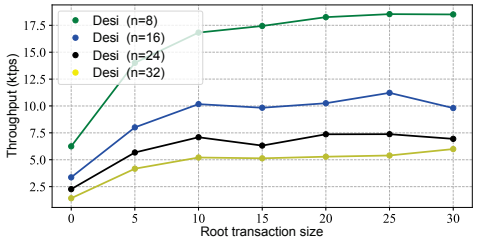


Fig. 5: The effect of rts on throughput

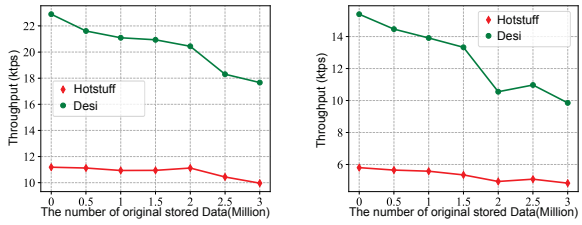
B. Performance Improvement

Throughput improvement. We evaluated both the Desi mechanism and the Hotstuff mechanism under different numbers of storage nodes. At the start of each run, the MPT trees on all nodes were initialized as empty, and each RT contained 20 STs. As shown in Fig. 4, although the overall throughput decreases with the increase in the number of nodes, the Desi mechanism consistently achieves a significant improvement over the Hotstuff mechanism, delivering 2.8–3.8 times higher throughput. While the throughput gain exhibits some fluctuation as the node count increases, the Desi mechanism consistently outperforms the traditional approach across all tested scales.

Throughput under different root transaction size. To test the effect of rts on the throughput, we run the Desi mechanism in 8, 16, 24, and 32 nodes scale separately, and varying the value of rts, getting the results shown in Fig. 5. Due to overheads associated with consensus, MPT tree updates, and network transmission, the system throughput does not increase indefinitely as rts grows. Therefore, the number of rts does not need to be excessively large and can be configured within the range of $[10, 20]$.

Throughput under different storage-scale. To test the effect of the initial storage scale on the throughput, we run the Desi mechanism in 8 and 16 nodes scale separately, and vary the value of the initial storage scale from 0 to 3 million entries, getting the results shown in Fig. 6. In the Desi scheme, the verification cost is no longer a bottleneck, and increasing the initial storage size will lead to an increase in the update cost of the MPT tree, resulting in a slight decrease in overall performance. Even though the initial storage size has a certain impact on system performance, there is still a significant improvement in throughput compared to Hotstuff mechanism.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:13:45 UTC from IEEE Xplore. Restrictions apply.



(a) throughput($n = 8$) (b) throughput($n = 16$)

Fig. 6: Impact of storage scale on throughput

C. Security analysis

This section analyzes how Desi defends against the threat model defined in III-B.

Transaction fabrication. The data owner maintains a local counter, denoted as cnt, to assign sequence numbers to all transactions. Each time the data owner issues a new write transaction (WT) or authorization transaction (AT), the transaction sequence number is set to the current cnt, and cnt is incremented locally. When a Byzantine storage node forges a simplified transaction on behalf of a victim data owner, the latest transaction sequence number recorded on-chain, after incrementing, will no longer match the data owner's local cnt. Consequently, subsequent write transactions from the data owner will be rejected by the storage chain, allowing the data owner to detect the node's malicious behavior.

Transaction tampering. Since the data owner can verify the execution status of transactions, any tampering by storage nodes after the transaction has been committed on-chain can be detected. Specifically, if the on-chain transaction identifier does not match the identifier of the original write transaction sent by the data owner, the malicious activity of the Byzantine node is exposed.

V. Related Work

In blockchain-based storage systems, each write transaction involves signature verification, transaction ordering, and execution with ADS updates. Existing optimizations mainly focus on the ordering and execution stages. Sharding improves throughput by partitioning nodes and distributing transactions across shards, but it sacrifices fault tolerance and security guarantees, while each shard still executes the full consensus pipeline, including signature verification [13]. Pre-execution reduces execution overhead by allowing only a subset of nodes to execute transactions in advance, yet all consensus participants must still perform signature verification [14]. Recent BFT protocols further improve ordering throughput, achieving hundreds of thousands of transactions per second, but these advances mainly optimize transaction ordering and leave signature verification as a major bottleneck [15], [16]. Only a few systems directly reduce signature verification overhead. For example, the Red Belly blockchain allows each transaction to be verified by only a subset of nodes through a leaderless consensus design [17], but this approach is tightly coupled with its specific protocol and is difficult to generalize to other BFT-based storage systems.

VI. Conclusion

In this paper, we present Desi, a novel high throughput optimization framework for blockchain-based storage systems. Desi restructures the write pipeline by aggregating simplified transactions into root transactions and performing signature verification only on the Merkle root during consensus, thereby substantially reducing the dominant verification overhead of write transactions. Furthermore, a smart contract based incentive and punishment mechanism enforces verifiable accountability against malicious storage nodes. Both theoretical analysis and prototype experiments demonstrate that Desi significantly lowers transaction computational overhead and substantially improves write throughput in practice, while preserving security and verifiability guarantees.

References

[1] J. Chen, S. Yao, Q. Yuan, K. He, S. Ji, and R. Du, "Certchain: Public and efficient certificate audit based on blockchain for tls connections," in *IEEE INFOCOM 2018*. IEEE, 2018, pp. 2060–2068.

[2] M. Shen, H. Liu, L. Zhu, K. Xu, H. Yu, X. Du, and M. Guizani, "Blockchain-assisted secure device authentication for cross-domain industrial iot," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 5, pp. 942–954, 2020.

[3] N. B. Truong, K. Sun, G. M. Lee, and Y. Guo, "Gdpr-compliant personal data management: A blockchain-based solution," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1746–1761, 2019.

[4] Z. Xiong, Y. Zhang, N. C. Luong, D. Niyato, P. Wang, and N. Guizani, "The best of both worlds: A general architecture for data management in blockchain-enabled internet-of-things," *IEEE network*, vol. 34, no. 1, pp. 166–173, 2020.

[5] G. Zhou, X. Guo, Z. Liu, T. Li, Q. Li, and K. Xu, "Trafficformer: an efficient pre-trained model for traffic data," in *IEEE symposium on security and privacy (IEEE SP)*, 2025, pp. 1844–1860.

[6] K. Xu, S. Ling, Q. Li, W. Bo, S. Meng, Z. Zhang, S. Yao, X. Liu, and L. Lin, "Research progress of network security architecture and key technologies based on blockchain," *Telecommunications Science*, vol. 44, no. 1, pp. 55–83, 2021.

[7] Y. Peng, M. Du, F. Li, R. Cheng, and D. Song, "Falcondb: Blockchain-based collaborative database," in *ACM SIGMOD*, 2020, pp. 637–652.

[8] Y. Hu, S. Kumar, and R. A. Popa, "Ghostor: Toward a secure {Data-Sharing} system from decentralized trust," in *NSDI*, 2020, pp. 851–877.

[9] H. Wu, Z. Peng, S. Guo, Y. Yang, and B. Xiao, "Vql: Efficient and verifiable cloud query services for blockchain systems," *IEEE TPDS*, vol. 33, no. 6, pp. 1393–1406, 2021.

[10] C. Zhang, C. Xu, H. Wang, J. Xu, and B. Choi, "Authenticated keyword search in scalable hybrid-storage blockchains," in *ICDE*. IEEE, 2021, pp. 996–1007.

[11] H. Wang, C. Xu, C. Zhang, J. Xu, Z. Peng, and J. Pei, "vchain+: Optimizing verifiable blockchain boolean range queries," in *ICDE*. IEEE, 2022, pp. 1927–1940.

[12] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[13] Y. Zhang, S. Pan, and J. Yu, "Txallo: Dynamic transaction allocation in sharded blockchain systems," in *ICDE*. IEEE, 2023, pp. 721–733.

[14] P. Ruan, D. Loghin, Q.-T. Ta, M. Zhang, G. Chen, and B. C. Ooi, "A transactional perspective on execute-order-validate blockchains," in *ACM SIGMOD*, 2020, pp. 543–557.

[15] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: Dag bft protocols made practical," in *ACM SIGSAC*, 2022, pp. 2705–2718.

[16] S. Liu, W. Xu, C. Shan, X. Yan, T. Xu, B. Wang, L. Fan, F. Deng, Y. Yan, and H. Zhang, "Flexible advancement in asynchronous bft consensus," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 264–280.

[17] T. Crain, C. Natoli, and V. Gramoli, "Red belly: A secure, fair and scalable open blockchain," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 466–483.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:13:45 UTC from IEEE Xplore. Restrictions apply.