Unveiling Algorand Storage Peculiarities

Nawaz Abdullah Malla[1,*,†], Alessandro Marcelletti[1,†], Andrea Morichetta[1,†] and Francesco Tiezzi[2,†]

[1]University of Camerino, Camerino, Italy
[2]Università degli Studi di Firenze, Firenze, Italy

Abstract
Blockchain technology has undergone rapid evolution since the advent of Bitcoin, giving rise to decentralized systems and digital currencies. Recently, Algorand blockchain has made significant progress in addressing scalability, security, and decentralization challenges inherent in traditional blockchain architectures. This paper explores how Algorand's storage mechanisms work and their interplay with the novel features supported by Algorand smart contracts. Our experiments focus on the behavior of different storage types—Local, Global, and Box storage—analyzing aspects such as user access, data visibility, and the effects of update and deletion operations. The findings showcase Algorand's adaptability, scalability, flexibility, and security, making it a compelling choice for developers and researchers. This research contributes to understanding Algorand's storage dynamics, providing insights into its impact on the broader blockchain landscape.

Keywords
Blockchain, Algorand, Smart Contracts, Storage, Dapp.

1. Introduction

Blockchain technology has progressed very rapidly since the advent of Bitcoin [1], paving the way for decentralized systems and digital currencies. Ethereum [2] and Hyperledger [3] allow for personalized rules and transaction formats through smart contracts governed by cryptographic conditions. Current blockchain platforms commonly encounter challenges in scalability, decentralization, security, and governance. Scalability issues lead to congestion and slow transaction processing. Decentralization concerns arise from the concentration of mining power. Security vulnerabilities pose risks while governance processes can be slow and inefficient.

One of the recent advancements in this landscape is Algorand, a groundbreaking blockchain platform developed to address the scalability, security, and decentralization challenges inherent in traditional blockchain architectures [4]. Algorand offers innovative solutions to address the pervasive challenges of current blockchain platforms. Algorand employs the Pure Proof of Stake (PPoS) consensus mechanism, ensuring fast and efficient block propagation, overcoming the scalability bottlenecks associated with other consensus mechanisms [4]. Algorand's PPoS not only achieves consensus in a decentralized manner but also enhances the platform's reliability by minimizing the risk of forks. Summing up, Algorand demonstrates the possibility of achieving at the same time decentralization, security, and scalability, thereby paving the way for more effective blockchain-based solutions.

Another innovative aspect of Algorand concerns its storage architecture, which presents a paradigm shift from conventional blockchain systems. Differently from traditional blockchains where smart contracts are often burdensome due to storage constraints, Algorand's approach allows for efficient data management, enabling more complex and feature-rich applications to be deployed on the network. This adaptability extends to data updation and deletion, crucial aspects of blockchain storage that have posed challenges for earlier systems. The advantages of Algorand's storage system are multifaceted and significantly impact the overall efficiency and user experience within the blockchain ecosystem. Firstly,

DLT2024: 6th Distributed Ledger Technologies Workshop, May, 14-15 2024 - Turin, Italy
*Corresponding author.
†These authors contributed equally.
nawaz.malla@unicam.it (N. A. Malla); alessand.marcelletti@unicam.it (A. Marcelletti); andrea.morichetta@unicam.it (A. Morichetta); francesco.tiezzi@unifi.it (F. Tiezzi)

the hierarchical storage model, combined with PPoS consensus, facilitates high throughput and low latency, ensuring that transactions are processed swiftly [4]. This is particularly critical for applications demanding real-time responsiveness, such as financial transactions and IoT use cases. Secondly, the stateful smart contract architecture enables developers to build decentralized applications with intricate logic and dynamic data requirements. This flexibility extends to data updation and deletion, allowing for a more dynamic and responsive blockchain environment.

However, understanding the usage of different storage mechanisms in Algorand presents challenges due to the complexity and variety of options available. Furthermore, the lack of standardized guidelines or best practices for storage usage adds another layer of difficulty, compelling developers to rely on iterative experimentation to identify optimal storage strategies. Overall, the multifaceted nature of Algorand's storage mechanisms necessitates a deep understanding of blockchain principles and careful consideration of various factors to effectively leverage these storage options in decentralized applications. For these reasons, in this work, we aim to provide a technical overview of Algorand's storage mechanisms and smart contract dynamics. Firstly, we provide insights into the complexities of data manipulation within Algorand storage, shedding light on how data is managed and modified within the Algorand storage. Secondly, we examine the behavior exhibited by different types of storage within Algorand and, lastly, we analyze the effects of upgrading the smart contracts post-deployment. To support these analyses, we perform several experiments to observe the behavior of storage in different scenarios, reporting and discussing the related outcomes.

The rest of the paper is organized as follows. Section 2 reviews relevant related works. Section 3 includes a general introduction to the Algorand blockchain and related concepts. The experiments and related analysis are discussed in Section 4. Finally, Section 5 concludes the paper by addressing directions for future work.

2. Related work

The research community is investigating Algorand's potential to address challenges encountered by other blockchain platforms [5, 6, 7, 8]. The Algorand blockchain has demonstrated its capabilities in solving the blockchain trilemma, which attracted the research community's attention towards this platform.

Many articles have been published on smart contracts and the security aspects of Algorand. Authors have analyzed the Algorand smart contract semantics and discovered different types of vulnerabilities. Sun et al. have proposed the *panda* framework [9] to detect the vulnerabilities automatically. Bartoletti et al. developed a formal model to establish the fundamental properties of the Algorand blockchain through rigorous proofs [10]. Alturki presented an encompassing model of consensus protocol and outlined the formal proof of its asynchronous safety [11].

Benhamouda et al. analyzed Algorand's protocol using a unique network model crafted to precisely address the concerns of protocol designers, particularly emphasizing recovery from intermittent network disruptions [12]. D'Onfro critically examines the feasibility of smart contracts in consumer protection laws, highlighting coding challenges for compliance, industry resistance, and concerns regarding post-execution modifications [13]. Abbasi et al. have identified the drawbacks of the current Algorand transaction fee model established a competitive market for Algorand block space and derived the optimal transaction fee and block size [14]. Šljukić et al. have introduced a model for conducting Algorand blockchain auctions to lease municipal-owned real estate properties [15].

While numerous studies have paid attention to aspects like the security of smart contracts, performance, and transactional costs within Algorand as witnessed by the above discussion, the complexity and diversity of available options in Algorand storage present challenges that have yet to be comprehensively addressed. Different from the above-mentioned works, our objective focused on examining the storage mechanisms within the Algorand ecosystem, intending to gain a comprehensive understanding and make valuable contributions to this aspect.

Table 1
Algorand Storage: Capacities, Operations, and Funding Accounts

Storage	Size	Read	Write	Delete	Funding Account
Local Storage	2kb	local_Get	local_Put	local_Del	user account
Global Storage	8kb	global_Get	global_Put	globalDel	creator account
Box Storage	32kb	box_Extract, box_Get	box_Put, box_replace	box_Del	application account

3. The Algorand blockchain and its storage mechanisms

In this section, we present the main concepts behind the Algorand blockchain, with a focus on its storage capabilities, and the delete and update features of smart contracts. Algorand has its native currency called Algo. To participate in consensus, build an application, pay a transaction fee, or store data on the blockchain, one must possess the Algos in his account to fulfill the minimum balance requirement [16].

Algorand supports two kinds of smart contracts: stateless (called smart signatures) and stateful. The stateless smart contract [17] is used for signing the transactions. Moreover, it provides a way to delegate the signature authority to another account. The stateful smart contract refers to an application that resides on the blockchain with a particular application ID. The Algorand blockchain stores the on-chain data in accounts by following the account-based model [18]. The data such as the local state of joined applications, Algo balances, and asset balances are stored on-chain. The account desiring to use local state needs to sign and submit the *applicationcall* transaction of type *OptIn*. The local state is visible only when an account opts into the application.

The Algorand storage is categorized into three types: Local storage, Global storage, and Box storage. Algorand blockchain uses different types of storage to have improved performance. The local storage is best suited for storing user-specific data, global storage for shared data, and box storage for application-specific data. It is possible to use only box storage to have the same model as Ethereum. Boxes and global storage are associated with the Algorand application, while local storage with every account address that opts into the application. Moreover, the storage is not free on the Algorand blockchain. The accounts responsible for funding that storage are responsible for the minimum balance requirement. We report in Table 1 information and features of the three types of storage, which we discuss in more detail below.

Local Storage. The local storage is allocated when an account opts into the application. Data is stored in key/value pairs. It is allocated at most 2kb of memory, which can store a maximum of 16 key-value pairs. The allocation of storage is determined at contract creation and cannot be modified later. The account that opts into the application is responsible for the minimum balance requirement.

Local storage can be read by an application call which has the application-id in its foreign app array and the account in its foreign account array. These foreign app arrays and foreign account arrays are used to specify external applications and accounts to a smart contract that should have access to enhance efficiency. By including app-ids in the foreign app array, a smart contract gains automatic access to the addresses of these specified applications.

Global Storage. Global storage stores data on-chain in the creator account. The creator account is responsible for funding the global storage and must possess a minimum balance to allocate global storage to the smart contract. The data is stored in key/value pairs. It has a maximum memory of 8kb, which can store a maximum of 64 key/value pairs [19]. The amount of global storage is assigned at the time of contract creation and cannot be mutated. The contract deployer address is responsible for funding the global storage.

The application call with a designated app-id can access and retrieve the global state of that application, while a global state can be written only by its application. The content of the global state can be deleted by the application, but the storage is not deallocated. Global storage is deleted only when the associated application is deleted.

Box Storage. The application can allocate boxes using the create box opcode specifying the size and name of the box[1]. An application can have as many boxes as it needs. The application account, rather than the creator, is responsible for funding the storage. The size of the box ranges from 0 to 32kb. The data in box storage is unstructured. Boxes are written only by the application that created them and are referenced by the box name and app-id. Every box must have a unique name [20]. The application is the only authority that can delete its boxes, with boxes persisting even upon the application's deletion. The minimum balance associated with boxes can be released by deleting the boxes before deleting the application.

4. Experimentation

In this section, we report the experimentation we made on Algorand storages to gain a comprehensive understanding of their functionalities and behavior. Our primary objective was to analyze how local, global, and box storages handle data manipulation, deletion, and update within the Algorand ecosystem. Through the experimentation, we aimed to identify the behavior of different types of storage, ultimately paving the way for more efficient and robust data management strategies within the Algorand network.

Experimental setup. To understand the behavior of different types of storage, we performed experiments on data manipulation in different types of storage and under different conditions. In particular, we report three different cases. The first relates to simple data manipulations, in which we test write, read, and delete operations in local, global, and box storages. Then, we assess data visibility after deletion, trying to read/write data from/to the local, global, and box storages after deleting these. Similarly, the last case refers to data manipulation after an update, testing reading/writing operations from/to local, global, and box storage when a smart contract is updated at runtime.

To test these cases, we developed a scenario with the different types of Algorand storage. As the first step, we create an application that is assigned with an app-id to access it later. On the creation of the app, the storage schema of the app is defined as the amount of data the app can store on-chain. The parameters *global-byte slices*, *global-ints*, *local-byte slices*, and *local-ints* are passed with some initial data to be stored on the chain. The application can be created without these parameters, but data cannot be saved on-chain.

Notice that the smart contract handles different types of transactions and specifies the conditions under which these transactions should be approved. Our contract logic allows all operations for creating, deleting, opting in, closing out, and updating the application, which can be restricted by writing specific logic. Since we are focusing on storage, we approve all kinds of contract transactions.

The experiments were conducted by deploying the smart contracts on the Algorand test network and interacting with them using the command line interface. To test the desired behaviors, we develop smart contracts that have all the functionalities for interacting with storage in different ways. The code was written in PyTeal, a Python-based smart contract language for Algorand. The source code for the experiments with all the executed commands is publicly available online at https://github.com/nawaz-malla/Algorand.

Storage analysis. Here we report the experimentation on the different Algorand storages and discuss relevant aspects. To conduct our experiments, we choose straightforward examples to minimize the complexity of the application, thus isolating the desired behavior.
Local storage. To comprehend the behavior of local storage we performed different operations. We found that the local state can be accessed only after the opt-in transaction by the user. Indeed, local storage provides some restrictions as compared to global storage by limiting access. The local storage can be read, written, and deleted by opting into the application. Upon deleting the smart contract, the local state is affected in such a way that its data persists even after the deletion of the smart contract and allows read, while the write operation is not possible.

[1]To create boxes, there are two different opcodes: box_create and box_put. The former opcode creates an empty box, while the latter opcode not only creates the box but also directly inserts data or value into the created box.

```
 1 @router.method
 2 def writeLocal(name: abi.String):
 3     return App.localPut(Txn.sender(), Bytes(
        "name"), name.get())
 4
 5 @router.method
 6 def readLocal(*, output: abi.String):
 7     return output.set(App.localGet(Txn.
        sender(), Bytes("name")))
 8
 9 @router.method
10 def deleteLocal():
11     return App.localDel(Txn.sender(), Bytes(
        "name"))
```

Listing 1: Data manipulation in local storage

```
 1 @router.method
 2 def createBox(box_name: abi.String):
 3     return Assert(App.box_create(box_name.get(), Int
        (9)))
 4
 5 @router.method
 6 def writeBox(box_name: abi.String, new_name: abi.
        String):
 7     return App.box_put(box_name.get(), new_name.get()
        )
 8
 9 @router.method
10 def deleteBox(box_name: abi.String):
11     return Assert(App.box_delete(box_name.get()))
```

Listing 2: Data manipulation in box storage

However, this comes at the cost of limited visibility to other users within the network. Interestingly, the data remains visible after an update operation and persists even after the deletion of the contract, possibly owing to local storage's intrinsic characteristics. Algorand blockchain uses the opcodes mentioned in Listing 1 to perform different operations on the local storage. The method in lines 1-3 is used to write the data on local storage, lines 5-7 reads the data, and lines 9-11 delete the data from local storage.

Global storage. Similarly, we performed the same experiments on the global storage to manipulate the global state and analyze the comparison between the local and global storage. We observed that during the creation and updating of smart contracts the global storage remains unaffected. Upon deleting the smart contract, this time the data in global storage does not persist, thus no read, and write operations are possible. Furthermore, any account having an app-id can access the global state of that application, which is not possible in local storage. Global storage exists as long as the smart contract exists. While global storage is linked to the application and local storage is linked to accounts, they share a commonality in storing data as key-value pair

Contrasting to local storage, global storage presents a more collaborative environment, with data accessible not only to the initiating user but also to other participants. The visibility of data, both before and after updates, underscores the transparency inherent in the global storage model. Notably, the data becomes inaccessible after the deletion of the smart contract; the local storage and boxes are available even after the deletion of the smart contract if the account has not cleared the state and boxes are not deleted before the deletion of the smart contract.

Box storage. The last considered storage is box one, where we created boxes of different sizes and performed read, write, and deletion operations. We noticed that while creating and updating smart contracts the box storage remains unchanged. However, upon smart contract deletion, we observed that the box state changes wherein the data in the box storage persists post-deletion, permitting only read operations while prohibiting write operations.

The deletion of box storage starts by first clearing all boxes and then deleting the associated smart contract. To create a box, the application account, which is a separate account from the creator must hold the minimum balance required for box creation which is 0.0025 Algos.

Box storage, akin to global storage, facilitates shared visibility among users. Listing 2 presents an example of manipulating the data in box storage. The method in lines 1-3 creates the box, lines 5-7 write the data in the box, and in lines 9-11 deletes the data from box storage. Yet, it distinguishes itself by preserving data even after the deletion of the smart contract.

Discussion. Table 2 summarizes the outcomes of our investigations, describing the behavior of Local, Global, and Box storages concerning user access, visibility, and the persistence of data post-update and post-deletion of the smart contracts. It shows that various storage options display distinct behaviors, forming a foundational understanding of their implications on user access, and data visibility. The table also shows that upgrading smart contracts post-deployment has no effects on storage.

Furthermore, it is clear from the table that deleting smart contracts affects only global storage since the data in local and box storage persists after the deletion of the smart contract. To delete box storage and

Table 2
Results of data manipulation on Algorand storage

Storage Types	On Creation and Updation			After Deleting of App	
	Write	**Read**	**Delete**	**Read after Delete**	**Write after Delete**
Local Storage	yes (optin)	yes (optin)	yes (optin)	yes	No
Global storage	yes	yes	yes	No	No
Box Storage	yes	yes	yes	yes	No

recover the minimum balance, one can initiate the deletion by first clearing all boxes and subsequently deleting the associated smart contract. In the case of local storage, the account can clear its local state from the smart contract to recover its minimum balance by using a close-out transaction. These findings offer a comparative analysis with the current literature, contributing to the broader understanding of Algorand storage mechanisms.

5. Conclusion

This work sheds light on Algorand's storage mechanisms and their implications. Its storage models distinguish Algorand from traditional blockchains. Our work focused on the Algorand storage to provide a comprehensive understanding of these storage types and analyze how various storage solutions handle data manipulation, deletion, and updates within the Algorand ecosystem. The experiments performed revealed that local storage is suitable for storing user-specific data, global storage can be used to store data relevant to all users like global variables or states, while box storage is useful for storing large amounts of data on-chain. By providing these different types of storage, Algorand allows developers to optimize their smart contracts based on their specific data storage and access needs, balancing factors such as cost, performance, and data scope. Future work in the exploration of Algorand's storage mechanisms and operations could delve deeper into several promising avenues as examining the security implications and interoperability of Algorand's storage mechanisms with other blockchain platforms, elucidating potential synergies and expanding its applicability across diverse use cases.

Acknowledgement

Malla is a student of the national PhD in Blockchain and DLT (Univ. Camerino, Italy) supported by a PNRR scholarship D.M. 351/2022. This work was partially supported by the project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NextGenerationEU.

References

[1] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Decentralized business review (2008).
[2] W. Nakamoto, V. Buterin, A next generation smart contract & decentralized application platform, Ethereum White Paper,@ inproceedings (2015) 1–36.
[3] U. S. LLC, Hyperledger Blockchain White Papers, 2018. URL: https://www.uniwebb.com/hyperledger-blockchain-white-papers/.
[4] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, N. Zeldovich, Algorand: Scaling byzantine agreements for cryptocurrencies, in: Proceedings of the 26th symposium on operating systems principles, 2017, pp. 51–68.
[5] M. Drijvers, S. Gorbunov, G. Neven, H. Wee, Pixel: Multi-signatures for consensus, in: 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 2093–2110.
[6] D. Leung, A. Suhl, Y. Gilad, N. Zeldovich, Vault: Fast bootstrapping for the algorand cryptocurrency, Cryptology ePrint Archive (2018).

[7] J. Chen, S. Gorbunov, S. Micali, G. Vlachos, Algorand agreement: Super fast and partition resilient byzantine agreement, Cryptology ePrint Archive (2018).

[8] J. Chen, S. Micali, Algorand, arXiv preprint arXiv:1607.01341 (2016).

[9] Z. Sun, X. Luo, Y. Zhang, Panda: Security analysis of algorand smart contracts, in: 32nd USENIX Security Symposium (USENIX Security 23), 2023, pp. 1811–1828.

[10] Bartoletti, M. et al., A formal model of algorand smart contracts, in: FC, Springer, 2021, pp. 93–114.

[11] Alturki, M. et al., Towards a verified model of the algorand consensus protocol in coq, in: FM, Springer, 2020, pp. 362–367.

[12] F. Benhamouda, E. Blum, J. Katz, D. Leung, J. Loss, T. Rabin, Analyzing the real-world security of the algorand blockchain, Cryptology ePrint Archive (2023).

[13] D. D'Onfro, Smart contracts and the illusion of automated enforcement, Wash. UJL & Pol'y 61 (2020) 173.

[14] Abbasi, M. et al., On algorand transaction fees: Challenges and mechanism design, in: ICC, IEEE, 2022, pp. 5403–5408.

[15] Šljukić, M. et al., A model for municipality buildings renting auction on algorand blockchain, in: ICMarkTech, volume 2, Springer, 2023, pp. 207–218.

[16] Algorand, Structure - Algorand Developer Portal, 2023. URL: https://developer.algorand.org/docs/get-details/transactions/.

[17] Algorand, SmartSig details - Algorand Developer Portal, 2023. URL: https://developer.algorand.org/docs/get-details/dapps/smart-contracts/smartsigs/.

[18] Algorand, Contract account algorand developer portal, 2023. URL: https://developer.algorand.org/docs/get-details/dapps/smart-contracts/smartsigs/modes/.

[19] Algorand, Global storage, 2023. URL: https://developer.algorand.org/docs/get-details/dapps/smart-contracts/apps/state/.

[20] Anne Kenyon, Smart Contract Storage: Boxes, 2023. URL: https://developer.algorand.org/articles/smart-contract-storage-boxes/.