


CRYPTOCURRENCIES  
&  
DECENTRALIZED LEDGERS

RESEARCH PROJECT - FALL 2022

## Verkle Trees & Stateless Ethereum

*Pranav Jangir (pj2251@nyu.edu)*  
*Animesh Ramesh (ar8006@nyu.edu)*  
*Arun Patro (akp7833@nyu.edu)*

Implementation Code at: 

## Abstract

*Blockchain technology requires efficient and secure data structures for storing and validating data. Stateless Ethereum, a new approach to Ethereum blockchain architecture, aims to improve scalability and security by enabling the use of stateless client nodes that do not store the entire blockchain history. Verkle Trees, a variant of Merkle Trees using Vector Commitment schemes, can improve upon the efficiency and security of traditional Merkle Trees and are suitable for use in stateless Ethereum nodes. In this research project, we present the implementation and comparison of Verkle Trees using various Vector Commitment schemes in C++ and evaluate their performance using simulation data from Ethereum mainnet. Our aim is to assess the efficiency of Verkle Trees and provide insights into their suitability for use in stateless Ethereum nodes. We present insights on what tree width is optimal, what the tradeoffs are in increasing width and which Vector commitments are best suited for use in stateless ethereum. Based on experiments we conclude that increasing width beyond a point increases proof time considerably but lesser reductions in proof size. We also discuss properties of Vector Commitments that are favourable for stateless ethereum and which VC schemes satisfy them.*

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Verkle Trees</b>	<b>3</b>
2.1	Comparison with Merkle Trees . . . . .	3
2.2	Vector Commitments (VCs) . . . . .	3
2.3	Summary of KZG and BalanceProofs . . . . .	4
2.4	Summary IPA style VCs . . . . .	4
2.5	Proof aggregation for polynomial commitments . . . . .	5
<b>3</b>	<b>Implementation and Evaluation</b>	<b>6</b>
3.1	Description of the simulation data used in the evaluation . . . . .	6
3.2	Implementation details of Verkle Tree . . . . .	6
3.3	Results of the evaluation . . . . .	6
<b>4</b>	<b>Discussion and Conclusion</b>	<b>10</b>
4.1	Summary of the main findings . . . . .	10
4.2	State trees and LevelDB . . . . .	11
4.3	Choice of Vector Commitment . . . . .	11

# 1 Introduction

**Stateless Ethereum** is a proposed implementation of the Ethereum blockchain that uses lightweight, efficient data structures called stateless clients to securely verify transactions rather than storing the entire state of the network.

**Verkle trees** are a type of data structure that can be used in stateless Ethereum to improve scalability, security, and efficiency by storing only a small number of hashes to represent the state of the network. Thus allowing for easy verification and making it more difficult for an attacker to modify the state without detection. Overall, using Verkle trees in Ethereum can help make the system stateless by enhancing efficiency, scalability, and security, particularly for devices with limited resources, such as smartphones and IoT edge devices.

## 2 Verkle Trees

### 2.1 Comparison with Merkle Trees

Verkle trees are similar to Merkle trees in structure except that they use efficient vector commitment schemes to store hashes. In Merkle trees, the hash of a node is a combination of the hashes of its children, therefore when sending proof that a certain key exists in the structure hashes of its siblings (node's parent's other children) need to be sent as well which blows up the proof size. Verkle trees improve upon this at the cost of added complexity (hence higher verification and proof construction time) using Vector Commitments.

### 2.2 Vector Commitments (VCs)

Vector commitments are protocols that lets us commit to a vector  $V$  of size  $n$ . Prover can send the commitment  $C(V)$  to a verifier along with a value  $y$  claiming that the value  $y = V[i]$ . A prover, with the help of a proof  $\pi(i)$  and  $C(V)$  which was sent earlier, can verify if indeed  $y = V[i]$ .

Succinct or efficient vector commitments are protocols where  $C(V)$  and  $\pi(i)$  are small. Ideally both single elements of a group or field. Note that by this definition, Merkle trees are also VC schemes with  $\pi(i) = O(\log n)$ .

Apart from succinctness, [1] defines that a VC is aggregatable if multiple  $\pi(i)$  can be combined into a single proof for a vector, and a VC is maintainable if many  $\pi(i)$  can be updated quickly when the underlying vector changes its values on certain points. We say many proofs are updated quickly if they can be updated faster than trivially updating each of them.

Our goal when constructing Merkle proofs is that the prover should be able to convince the verifier that the  $i^{th}$  child of a parent node has the same hash as the hash value that the verifier holds. This is exactly what we want to achieve by using VCs.

**KZG style VCs** [2] are VC schemes that are easily aggregatable but not maintainable and can be slightly modified into **BalanceProofs**[1] so that it becomes maintainable as well at the cost of increased subsequent proof generation time. KZG uses elliptic curve pairing and requires a bilinear pairing for verification. They are optimal in proof size but require a trusted setup generation phase between the prover and the verifier.

**IPA style VCs** are VC schemes that use inner products as commitments and recursively reduce the size of the proof to be sent. IPA style VC schemes have the benefit that they are simpler and do not require trusted setup generation or pairing operations but are more expensive in terms of proof size and verification time as we discuss below.

## 2.3 Summary of KZG and BalanceProofs

We implemented a simpler version of aSVC [3] as mentioned in [4]. We also implemented the proof aggregation ideas present in aSVCs.

KZG are polynomial based vector commitments as they model the vector as a polynomial as follows:

Suppose we have a vector  $V$  of size  $n$ . Then we can see the  $i^{th}$  entry of the vector as evaluation of a polynomial  $p(x)$  at  $w^i$  where  $w$  is  $n^{th}$  root of unity. A secret elliptic curve group element  $s$  is used along with its powers necessary to compute the polynomial at  $s$ . Commitment to a polynomial  $p(x)$  is then defined as  $p(s)$  which is a group element. The secret values need to be generated in a trusted manner, which can be done efficiently using techniques like [5]. The commitment  $p(s)$  can then be used to open the polynomial at any random point  $r$ .

If we want to prove that  $V[i] = y$  then consider the polynomial  $q(x) = \frac{p(x)-y}{x-w^i}$ . Clearly, if  $p(w^i) = y$  then,  $q(x)$  is a polynomial and  $q(x)(x - w^i) = p(x) - y$ . Therefore, if we set the proof  $\pi(i) = q(s)$ , then in order to prove  $p(w^i) = y$ , it is enough to verify that  $e(\pi(i), (s - w^i)) = e(p(s) - y, 1)$  where  $e$  is the bilinear pairing operation.

**aSVCs** are similar to vanilla KZG with the added observation that if we take the domain of the polynomials as roots of unity, then proofs can be aggregated and modified efficiently using lagrange interpolation and Barycentric formulas [6]. aSVC also defines an update key  $upd_i$ , using which we can easily update the proof  $\pi(i)$  in constant time if the value of  $V[i]$  changes. However, if the polynomial changed values at many  $w^i$  then each proof needs to be updated individually, hence the scheme is aggregatable but not maintainable.

**BalanceProofs**[1] modify the above scheme by lazily updating the proofs when needed or when the total number of updates exceeds a specific threshold. To improve it further, they divide the whole vector into many small sub-vectors and introduce another vector that just contains the commitment values of all the subvectors. Proof updates are then naively done for the subvectors but lazily for the vector that contains commitments for the sub-vectors. By doing this they are able to reduce the proof update time from  $O(n)$  to  $O(\sqrt{n} \log n)$  at the cost of increasing updated proof query time from  $O(1)$  to  $O(\sqrt{n})$ . In the discussion section we discuss why improving the updation time is important for stateless ethereum.

We discuss shortly how many proofs can be aggregated into a single proof.

## 2.4 Summary IPA style VCs

Similar to KZG, we look at vector as a polynomial. We can see  $p(x)$  as

$$\begin{aligned} p(x) &= (a_0, a_1, \dots, a_n) \cdot (x^0, x^1, \dots, x^n) \\ p(x) &= \vec{a} \cdot \vec{x} \end{aligned}$$

If we have group elements

$$\vec{g} = (g_0, g_1, \dots, g_n)$$

Then, commitment is defined as

$$C(p) = \vec{a} \cdot \vec{g}$$

IPA relies on proving to the verifier that  $X$  follows

$$X = \vec{a} \cdot \vec{g} + \vec{b} \cdot \vec{h} + (\vec{a} \cdot \vec{b})q$$

where  $X$  is sent by the prover to the verifier and  $\vec{g}$ ,  $\vec{h}$  and  $q$  are elliptic curve group elements and are known to both the prover and the verifier.  $X$  is said to follow the "Inner product property." If we take  $\vec{b} = (z^0, z^1, \dots, z^n)$  then the prover can prove  $p(z) = y$  iff

$$X = \vec{a} \cdot \vec{g} + \vec{b} \cdot \vec{h} + yq$$

follows the Inner product property. The Inner product property can be proved naively by sending all the values of  $\vec{a}$  but that would make the proof size  $O(n)$ . We can instead reduce the problem to proving a Inner product property statement similar to the one above but of size  $O(n/2)$  by doing one interaction between the prover and verifier. Therefore with  $O(\log n)$  such interacts, the problem reduces to proving a IP property with  $n = 1$ . The interactivensness can be removed by using the Fiat–Shamir heuristic. Note that the proof size is now  $O(\log n)$ . For details refer [7].

	IPA	KZG
Assumption	Discrete log	Bilinear group
Trusted setup	No	Yes
Commitment size	1 Group element	1 Group element
Proof size	$2 \log n$ Group elements	1 Group element
Verification	$O(n)$ group operations	1 Pairing

Table 1: Comparison between IPA and KZG. Source:[7]

## 2.5 Proof aggregation for polynomial commitments

In our implementation, instead of sending each proof individually for each state change in a block, we aggregate all the state changes and send an aggregated proof that is of size  $O(1)$  for KZG and  $O(\log n)$  for IPA[3][8][4].

**Main Idea:** If we have to prove multiple polynomial evaluations

$$\begin{aligned} f_0(z_0) &= y_0 \\ &\vdots \\ f_{m-1}(z_{m-1}) &= y_{m-1} \end{aligned}$$

$$z_i \in \{w^0, \dots, w^{n-1}\}$$

Then that is equal to committing to the polynomial  $g(x)$  and then opening it later at a random value.

$$g(x) = r^0 \frac{f_0(x) - y_0}{x - z_0} + r^1 \frac{f_1(x) - y_1}{x - z_1} + \dots + r^{m-1} \frac{f_{m-1}(x) - y_{m-1}}{x - z_{m-1}}$$

where  $r$  is a random value generated using Fiat-Shamir Heuristic. This way we are going to send proof for a single polynomial. Note that we still have to send the commitments to the polynomials  $f_i(x)$  individually.

### 3 Implementation and Evaluation

#### 3.1 Description of the simulation data used in the evaluation

The simulation data used in the evaluation was obtained from the Etherscan API and stored locally for analysis. The data consisted of a sample of blocks and transactions from the Ethereum mainnet blockchain, including information such as the block height, transaction hash, and the addresses involved in each transaction [9]. The sample size of the data varied depending on the specific evaluation being conducted, but generally included a representative subset of the total number of blocks and transactions on the Ethereum blockchain. The data was collected and stored in a structured format, such as a JSON file, for easy analysis and processing. The use of simulation data from the Etherscan API allowed for the evaluation of the Verkle trees in a controlled and reproducible manner.

#### 3.2 Implementation details of Verkle Tree

We implemented a C++ class for Verkle Tree that supports adding keys to the tree, generating proofs, verifying proofs and aggregating proofs using the aggregation technique described above. The tree has adjustable width and has four modes of operation: Verkle tree using KZG based VC, Verkle tree using IPA based VC, and Merkle trees (Patricia-Merkle and Binary). We are mainly using the BLS12-381 curve for elliptic curve operations provided by the c-bls library. We also used the c-kzg library for some utility functions like pippenger linear combination and FFT. c-kzg has a basic implementation for kzg but does not support proof aggregation therefore we only used the utility functions provided by the library. We used GNU-MP library for big-integer arithmetic in C++. Dependencies and build instructions are present in the github repository.

*Note: We have implemented multiproof (Fig. 1) for both the Merkle trees as defined here [10].*

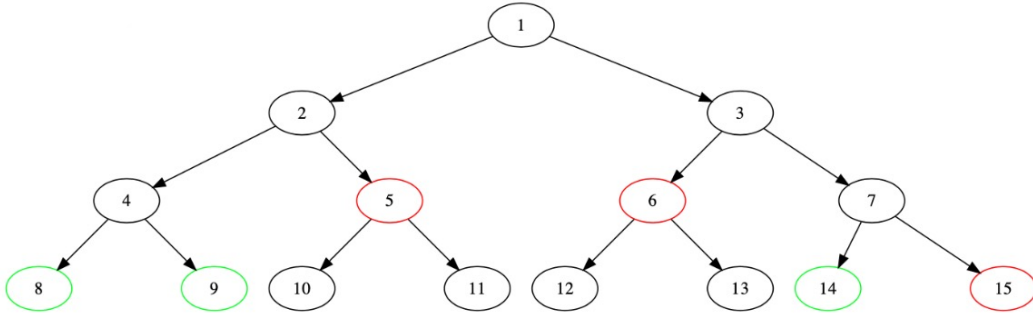


Figure 1: An example of a multiproof for a Merkle tree: the green nodes represent those for which a proof of existence has been queried, and the red nodes are the ones that we need to provide.

#### 3.3 Results of the evaluation

We refer to Patricia-Merkle as PM, Binary-Merkle as BM and KZG-x/IPA-x where  $2^x$  is the width of the Verkle Tree. We show multiple graphs highlighting the differences of the various data structures.

**NOTE:** Below we consider proof to be the full data that a full node has to send to a incoming stateless node. This involves not only the polynomial proof, but also the commitments to required nodes of the tree.

### 3.3.1 Proof Size

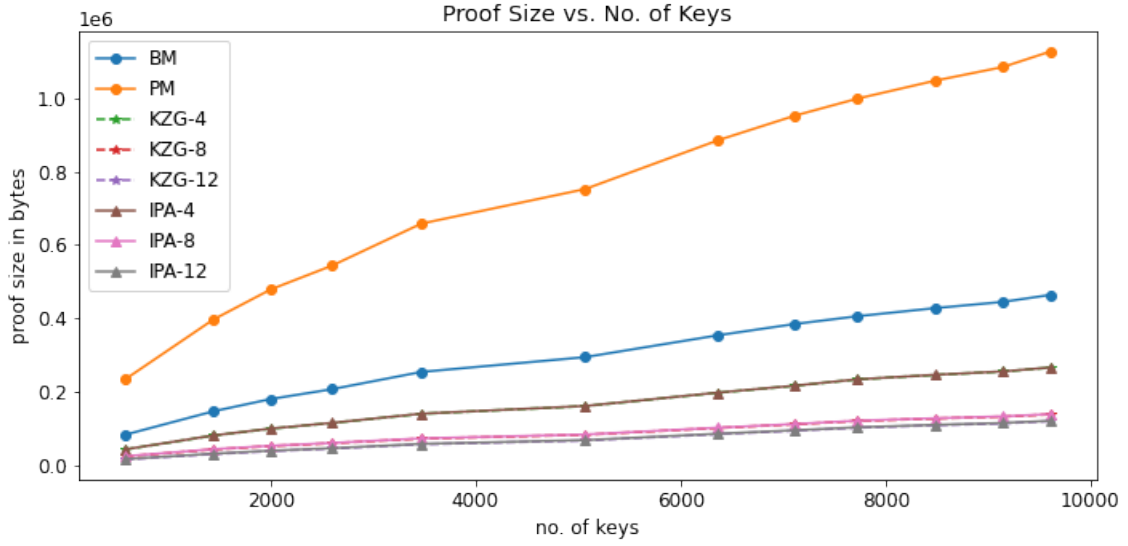


Figure 2: This graph of proof size comparison shows Patricia-Merkle (PM) is the worst, Binary-Merkle (BM) does a bit better but more complex trees are more performant.

**NOTE:** It may look like some curves are missing, but IPA-x and KZG-x had almost the same proof size (Fig. 3). This is because while there is a difference in their polynomial proof size, here we send poly proof and all the node commitments required. The node commitments overshadow the difference in their standalone proof size.

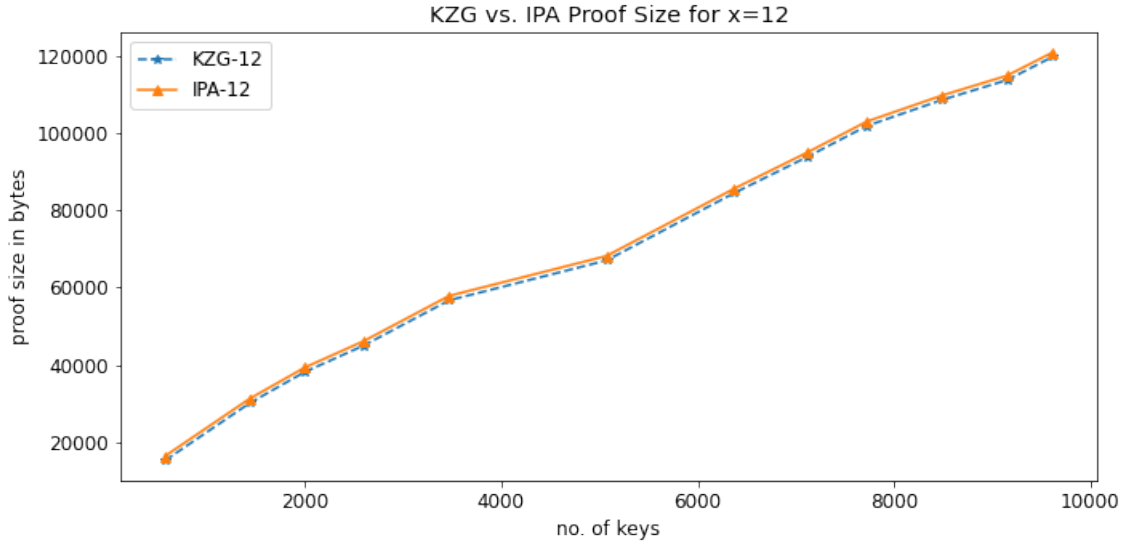


Figure 3: This graph shows that proof size of KZG and IPA are almost equal. For  $x = 4$  or  $x = 8$ , they almost overlap.

### 3.3.2 Time taken for Proof Generaion

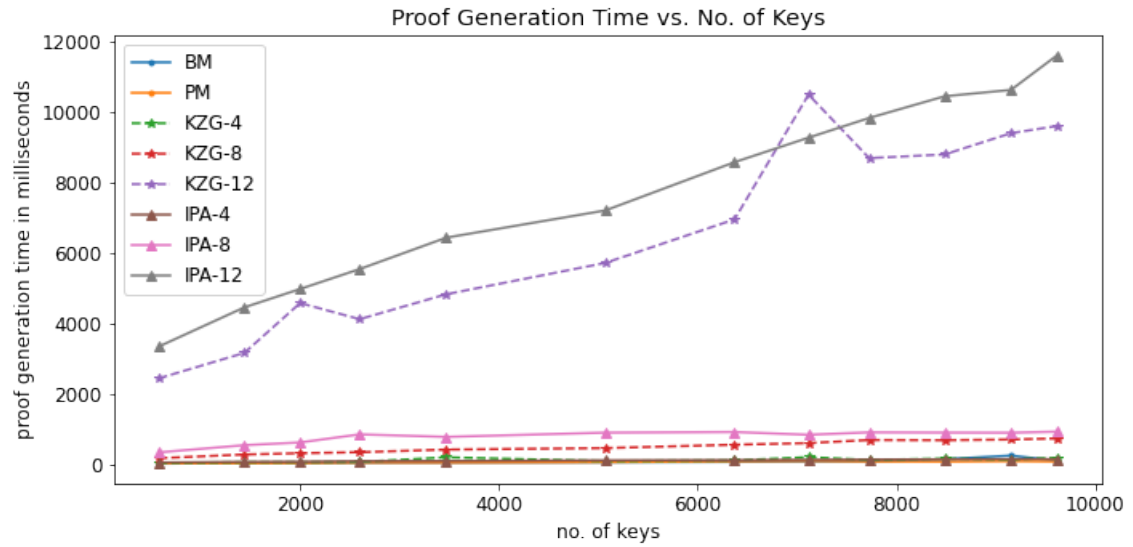


Figure 4: We see that increasing the width of the KZG and IPA increases the proof generation time a lot, especially when we change the width from  $2^8$  to  $2^{12}$

### 3.3.3 Time taken for Proof Verification

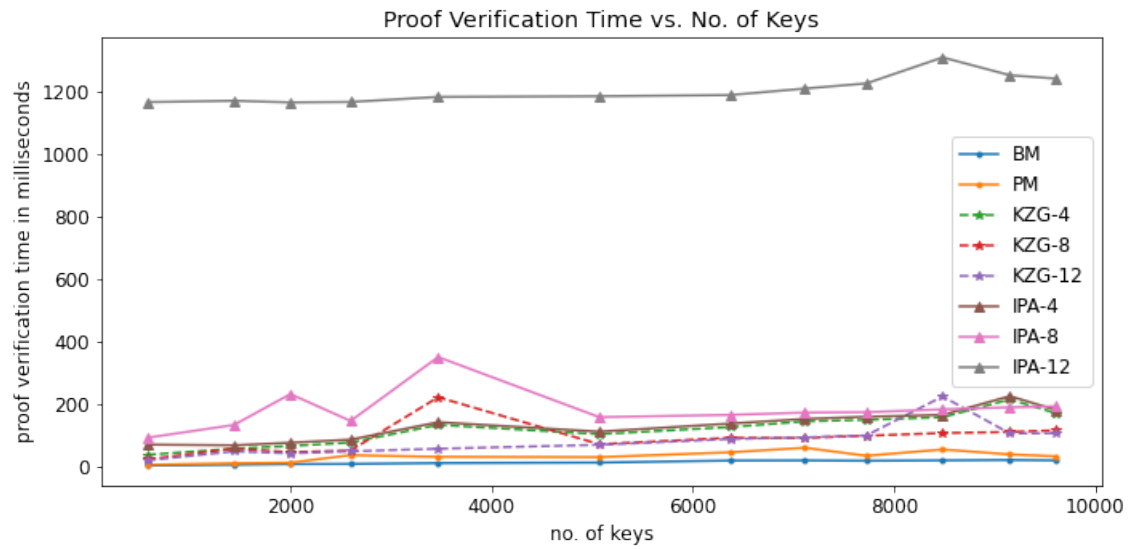


Figure 5: Proof verification time of IPA-12 is very poor compared to the rest.



### 3.3.4 Comparison of Width in Verkle Trees

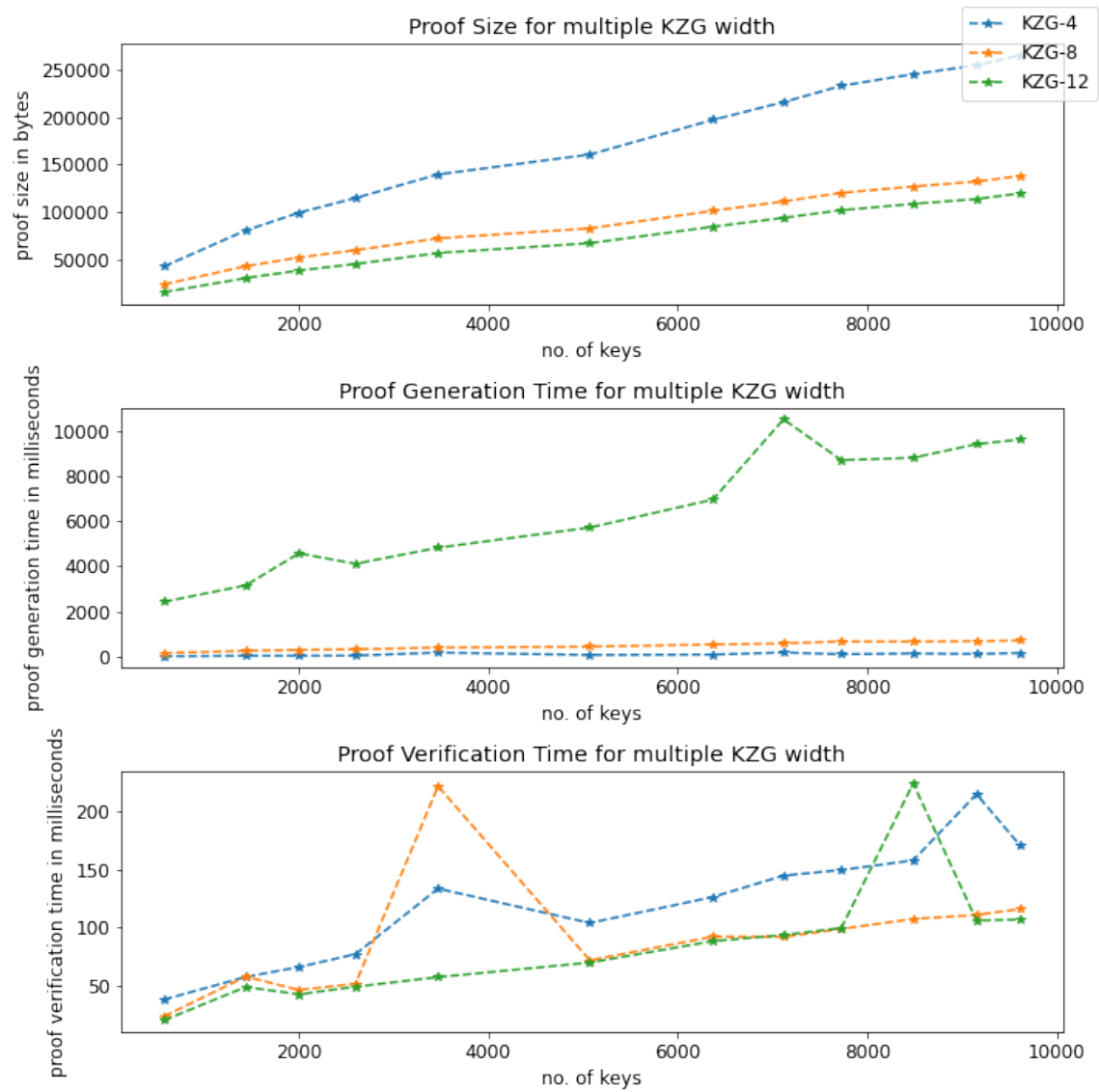


Figure 6: In this figure, we only compare the metrics for KZG VC

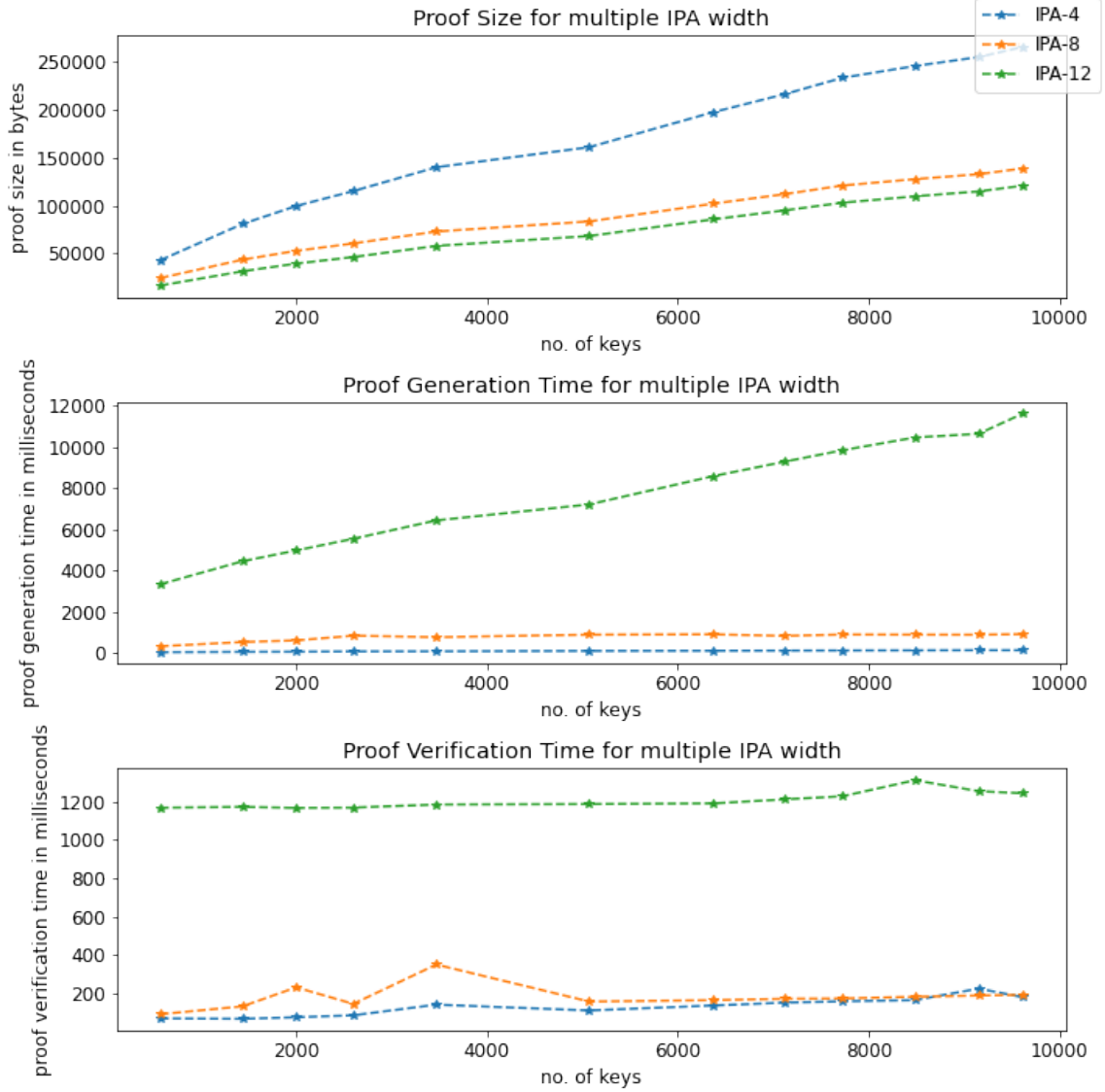


Figure 7: In this figure, we only compare the metrics for IPA VC

## 4 Discussion and Conclusion

### 4.1 Summary of the main findings

Our research shows that Verkle trees are significantly superior to Merkle trees in terms of proof size and space requirements. We have also studied the effect of various parameters on the performance of Verkle trees and found that increasing the width of the tree leads to a reduction in proof size, but also causes an increase in the time required to generate the proof. However, after a certain width, the decrease in proof size becomes less pronounced while the increase in proof generation time becomes more significant. Based on our analysis, a width of  $2^8$  appears to be the optimal choice in our case study. However, in actual practice, the optimal width may vary based on the specific application and the size of the state being used. It was determined that a width of  $2^{10}$  is the optimal choice for Verkle trees in Ethereum. This discrepancy is due to the fact that we have used a subset of the state tree and not its entirety.

We observe from the graphs that:

1. KZG-12 is 9-15x lesser in proof size than PM, and KZG-4 is 4-5x lesser (Fig. 2).

2. KZG-x and IPA-x proof sizes are very comparable as node commitments make up most of the space.
3. Generating proof time is bad for large width KZG, as KZG-12 is 91-222x longer than PM. Whereas KZG-4 is 2-4x longer than PM (Fig. 4).
4. Verification proof time is not too severely affected as KZG-12 is 1.2-2.2x longer than PM and KZG-4 only 2-3x longer than PM (Fig. 5).
5. IPA takes much longer generate proofs also, with IPA-4 taking 1.5-3x longer and IPA-12 taking 150-300x longer than PM (Fig. 4).
6. IPA takes longer to verify than KZG with almost 10-100x longer than PM (Fig. 5).

## 4.2 State trees and LevelDB

Ethereum implementation stores the authenticated Patricia Trie using Google’s levelDB, leading to slow transaction validation due to expensive I/Os [11]

Verkle trees and Patricia Merkle trees are data structures that can use LevelDB as a backend to efficiently store and verify large amounts of data. Verkle trees have the potential to be more efficient in terms of the cost of IO access on LevelDB due to their use of an efficient Vector Commitment scheme, as they reduce the total nodes required for proof creation and verification, thus lowering the overall cost of IO access. In contrast, Patricia Merkle trees need to do more reads and writes.[12][13]

The use of the LSM tree data structure can lead to write amplification, which is the phenomenon of writing more data to disk than is strictly necessary. This can happen because the LSM tree writes new data to the memtable before it is compacted and written to a new level, and it also writes old data to a new level when compaction occurs. This means that the same data may be written multiple times to different levels on disk.

Read amplification, on the other hand, refers to the phenomenon of reading more data from disk than is strictly necessary. This can happen when LevelDB has to search through multiple levels to find the data it is looking for.

Verkle trees may be more efficient regarding the cost of IO access and amplification than Patricia Merkle trees in LevelDB. Specifically, if Verkle trees can perform lookups with fewer reads in LevelDB compared to Patricia Merkle then the cost savings of using Verkle tree is amplified because of the read/write amplification

## 4.3 Choice of Vector Commitment

KZG performs better than IPA but requires a trusted setup phase and stronger security assumptions. As mentioned in BalanceProofs[1], KZG based VCs like aSVC[3] can be compiled to form an maintainable VC that can fast update proofs, but this is not possible with IPA based VCs.

Maintainability of a VC scheme can be useful as there can be hybrid nodes that maintain limited state like commitments and proofs for certain nodes of the tree. Fast updation of proofs and commitments will be a very useful feature for such nodes as it would greatly reduce the bandwidth required to send proofs when node values change.

## References

- [1] W. Wang, A. Ulichney, and C. Papamanthou, “Balanceproofs: Maintainable vector commitments with fast aggregation,” Cryptology ePrint Archive, Paper 2022/864, 2022, <https://eprint.iacr.org/2022/864>. [Online]. Available: <https://eprint.iacr.org/2022/864>
- [2] A. Kate, G. M. Zaverucha, and I. Goldberg, “Constant-size commitments to polynomials and their applications,” in *Advances in Cryptology - ASIACRYPT 2010*, M. Abe, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 177–194.
- [3] A. Tomescu, I. Abraham, V. Buterin, J. Drake, D. Feist, and D. Khovratovich, “Aggregatable subvector commitments for stateless cryptocurrencies,” Cryptology ePrint Archive, Paper 2020/527, 2020, <https://eprint.iacr.org/2020/527>. [Online]. Available: <https://eprint.iacr.org/2020/527>
- [4] D. Feist, “Kzg polynomial commitments,” Blog, 2020, <https://dankradfeist.de/ethereum/2020/06/16/kate-polynomial-commitments.html>. [Online]. Available: <https://dankradfeist.de/ethereum/2020/06/16/kate-polynomial-commitments.html>
- [5] V. Nikolaenko, S. Ragsdale, J. Bonneau, and D. Boneh, “Powers-of-tau to the people: Decentralizing setup ceremonies,” Cryptology ePrint Archive, Paper 2022/1592, 2022, <https://eprint.iacr.org/2022/1592>. [Online]. Available: <https://eprint.iacr.org/2022/1592>
- [6] J.-P. Berrut and L. N. Trefethen, “Barycentric lagrange interpolation,” *SIAM Review*, vol. 46, no. 3, pp. 501–517, 2004. [Online]. Available: <https://doi.org/10.1137/S0036144502417715>
- [7] D. Feist, “Inner product arguments,” Blog, 2021, <https://dankradfeist.de/ethereum/2021/07/27/inner-product-arguments.html>. [Online]. Available: <https://dankradfeist.de/ethereum/2021/07/27/inner-product-arguments.html>
- [8] —, “Pcs multiproofs using random evaluation,” Blog, 2021, <https://dankradfeist.de/ethereum/2021/06/18/pcs-multiproofs.html>. [Online]. Available: <https://dankradfeist.de/ethereum/2021/06/18/pcs-multiproofs.html>
- [9] A. Akhunov, “The shades of statefulness (in ethereum nodes),” 2019. [Online]. Available: <https://medium.com/@akhounov/the-shades-of-statefulness-in-ethereum-nodes-697b0f88cd04>
- [10] “Ethereum consensus specs,” github, 2020, <https://github.com/ethereum/consensus-specs/blob/dev/ssz/merkle-proofs.md#merkle-multiproofs>. [Online]. Available: <https://github.com/ethereum/consensus-specs/blob/dev/ssz/merkle-proofs.md#merkle-multiproofs>
- [11] A. Chepurnoy, C. Papamanthou, S. Srinivasan, and Y. Zhang, “Edrax: A cryptocurrency with stateless transaction validation,” Cryptology ePrint Archive, Paper 2018/968, 2018, <https://eprint.iacr.org/2018/968>. [Online]. Available: <https://eprint.iacr.org/2018/968>
- [12] S. Ponnappalli, A. Shah, A. Tai, S. Banerjee, V. Chidambaram, D. Malkhi, and M. Wei, “Scalable and efficient data authentication for decentralized systems,” 09 2019.
- [13] P. Raju, R. Kadekodi, V. Chidambaram, and I. Abraham, “Pebblesdb: Building key-value stores using fragmented log-structured merge trees,” in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 497–514. [Online]. Available: <https://doi.org/10.1145/3132747.3132765>