Cauchyproofs: Batch-Updatable Vector Commitment with Easy Aggregation and Application to Stateless Blockchains

Zhongtang Luo*, Yanxue Jia†, Alejandra Victoria Ospina Gracia‡, Aniket Kate§
*Purdue University, luo401@purdue.edu
†Purdue University, jia168@purdue.edu
‡No Affiliation[1], aleospinagracia@hotmail.com
§Purdue University / Supra Research, aniket@purdue.edu

***Abstract*—Stateless blockchain designs have emerged to address the challenge of growing blockchain size by utilizing succinct global states. Previous works have developed vector commitments that support proof updates and aggregation to be used as such states. However, maintaining proofs for multiple users still demands significant computational resources, particularly in updating proofs with every transaction. This paper introduces Cauchyproofs, a batch-updatable vector commitment enabling proof-serving nodes to efficiently update proofs in quasi-linear time relative to the number of users and transactions, utilizing an optimized KZG scheme to achieve complexity $O\left((|\vec{\alpha}| + |\vec{\beta}|)\log^2(|\vec{\alpha}| + |\vec{\beta}|)\right)$, compared to previous $O\left(|\vec{\alpha}| \cdot |\vec{\beta}|\right)$ approaches. This advancement reduces the computational burden on proof-serving nodes, allowing for efficient proof maintenance across large user groups. We demonstrate that our approach is approximately five times faster than the naive approach at the Ethereum-level block size. Additionally, we present a novel matrix representation for KZG proofs utilizing Cauchy matrices, enabling faster all-proof computations with reduced elliptic curve operations. Finally, we propose an algorithm for history proof query, supporting retrospective proof generation with high efficiency. Our contributions substantially enhance the scalability and practicality of proof-serving nodes in stateless blockchain frameworks.**

1. Introduction

Traditional blockchain protocols such as Bitcoin and Ethereum require every fully-functional running node to maintain a state consisting of all accounts and all transactions ever generated since the inception of that blockchain. The size of the state grows as the blockchain runs, reaching over 85 million elements in Bitcoin and over 200 million accounts with a 35 GB of state in Ethereum [1].

Such enormous state sizes create difficulty in the decentralization of the network, as the storage and computational requirements for running a full node become increasingly expensive. To address this issue, *stateless blockchains* [1] have been proposed to build a cryptocurrency with a succinct global state. The design paradigm of stateless blockchains represents a shift of responsibility. Every running node only maintains a succinct digest of the state, and the user has to provide the necessary proofs to the node to showcase the validity of their transactions, usually by demonstrating the inclusion of their account and balance in the state. The node then verifies the proof and processes the transaction accordingly.

The succinct global state digest is usually achieved by a vector commitment [2], [3], [4], [5], which allows a vector of values to be committed by a short digest. For instance, previous works have explored representing the state of smart contracts [6] or account balances [2] by a vector and employ a vector commitment to implement a succinct global state digest. In this design paradigm, to verify the validity of every transaction in a block, the node has to verify the proof of every account involved in the transaction. Therefore, all the proofs have to be included in the block. To reduce block size, the concept of *aggregatable vector commitment* [2] has been proposed, which allows the aggregation of multiple proofs into a single proof of constant size, reducing the size of the block.

Ideally, the proof of every user should not change often if that user makes no transactions, even if the state is being updated with other transactions. However, Christ et al. [1] has shown that it is impossible to build a stateless blockchain without frequent proof update almost linear to the number of transactions, thus necessitating what prior works call as a *proof-serving node* [2], [3], [7] that maintains and serves proofs of a specific user group. Figure 1 illustrates this scenario.

Previous works on aggregatable vector commitments have demonstrated that it is possible to update one proof with one change in the vector in constant time [2], [5]. With this technique, if the proof-serving node computes through an update of $|\vec{\beta}|$ transactions for each of the $|\vec{\alpha}|$ users, naively it will take at least $O\left(|\vec{\alpha}| \cdot |\vec{\beta}|\right)$ time complexity, since for each user, the node has to access every transaction at least once. Naturally, such complexity blows up quadratically as users and transactions increase, This raises a natural question: can we do better?

In this paper, we motivate the idea of a *batch-updatable* vector commitment, where the proof-serving node can update $|\vec{\alpha}|$ proofs with $|\vec{\beta}|$ modifications in time quasi-linear to $\left(|\vec{\alpha}| + |\vec{\beta}|\right)$. Specifically, we revised the previous KZG-

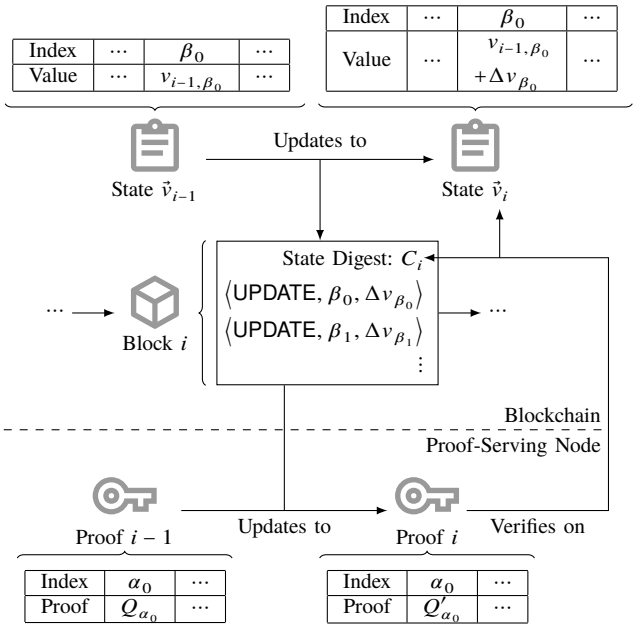

Figure 1: In this stateless blockchain example, let us assume that the blockchain state is a vector $\vec{v}$ of length n. We denote the global state after the i-th block $\vec{v}_i$ and the succinct global state digest C_i. The i-th block contains this digest C_i and a list of updates $\left\langle \mathsf{UPDATE}, \beta_j, \Delta v_{\beta_j} \right\rangle$ representing the changes in the vector from $\vec{v}_{i-1}$ to $\vec{v}_i$. Our proof-serving node maintains and serves proofs of a specific group of users, represented by the vector $\vec{\alpha}$ where $0 \le \alpha_j \le n$ that contains a list of indices for $\vec{v}$. It is responsible for providing proofs of the type Q_{α_j}, which proves the value of v_{i,α_j} given the digest C_i. Given that updating proofs frequently is inevitable, the proof-serving node has to update these proofs after every block gets published to ensure that the verification goes through on the latest digest.

based schemes [2] and developed an algorithm to achieve said update in $O\left(\left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right)\log^2\left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right)\right)$, irrelevant to the length of the vector n, which may consist of many more other users' data. This result helps to motivate proof-serving nodes, as one node maintaining a group of users becomes cheaper than each user maintaining their proof individually, which would cost at least $O\left(|\vec{\alpha}| \cdot |\vec{\beta}|\right)$ for each user to access every transaction. With Etheruem's current block size of 20,000 transactions, we expect our algorithm to enable proof-serving nodes to be roughly 5 times faster than the naive algorithm.

As a side result, the method we developed allows us to represent KZG proofs as matrix multiplications with Cauchy matrices. The result allows us to compute all proofs in a KZG vector commitment with $2n\log n + (n\log n)$ elliptic multiplications, an improvement from the previous $3n\log n + o(n\log n)$ result [8] that cares about the numbers of elliptic curve multiplications. It can save around 10^6 multiplications under Ethereum's use case study [8]. The improvement comes from the fact that the previous result's algorithm first computes the KZG polynomial coefficients given the vector, a step which we discover that we can skip. We note that applying the all-proof computation algorithm to the batch update scenario naively causes the update complexity to be quasi-linear to the total vector length n, which may be significantly larger than the user and update size $|\vec{\alpha}|$ and $\left|\vec{\beta}\right|$. Thus, batch update remains an interesting scenario that this work aims to explore.

As a final observation, we note that by divide and conquer, our algorithm can also be modified to support a *history proof query* that outputs proofs at any point in time throughout the blockchain lifetime. The history query is useful in a scenario where a user wants to verify the state of the blockchain at a certain point in time, which can be useful for auditing purposes.

1.1. Contribution

As our first contribution, we present a way to represent KZG proofs as matrix multiplications with Cauchy matrices. The formulation allows several optimizations in the computation of KZG proofs.

Our second contribution motivates the concept of a *batch-updatable* vector commitment, where we can apply update $|\vec{\alpha}|$ proofs with $\left|\vec{\beta}\right|$ modifications efficiently. Our new algorithm allows us to update $|\vec{\alpha}|$ KZG proofs with $\left|\vec{\beta}\right|$ modifications in time complexity quasi-linear to $\left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right)$. Based on the notation we developed, we apply extra matrix multiplication and polynomial evaluation techniques to reduce the computation cost from $O\left(|\vec{\alpha}| \cdot |\vec{\beta}|\right)$ to $O\left(\left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right)\log^2\left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right)\right)$. We implemented and benchmarked our solution, finding it about five times faster than the naive approach at the Ethereum-level block size.

For our third contribution, we reexamine the fast KZG all-proof computation first theorized by Feist et al. [9]. We discover that putting the computation under the lens of vector commitment without detouring to polynomial coefficients allows us to reduce the number of elliptic multiplications from $3n\log n + o(n\log n)$ to $2n\log n + o(n\log n)$. We analyzed this improvement under Ethereum's use case scenario [8], and found that we can save millions of elliptic curve multiplication operations.

Our last contribution discusses the scenario where a history-proof query is required. We similarly give an algorithm that allows the computation of a group of $|\vec{\alpha}|$ proofs at any point in time throughout $\left|\vec{\beta}\right|$ updates in time complexity quasi-linear to $\left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right)$.

1.2. Related Work

Algebraic vs. Hash-Based Vector Commitments. Parallel to solutions that leverage algebraic properties such as KZG-based vector commitments, hash-based solutions, most notably the Merkle tree [10], implement similar functionalities with an interesting trade-off. For the Merkle-based solution, updates can be done dynamically without the need to

batch them together, but aggregation is not possible without SNARK-related techniques. Moreover, with Merkle trees, updating the proofs requires knowing the full vector data, whereas for algebraic solutions this is not necessary. We highlight a few key differences in Table 1.

Other Algebraic Vector Commitments. Various other cryptographic primitives have been employed to develop aggregatable vector commitments, including hidden order groups [4], [5], [12] and lattices [13]. To our knowledge, no batch-update solutions have been proposed for these constructions. We speculate that a unified framework could be developed to support batch updates for these constructions and leave it for future work.

Advancement on Elliptic Curve-Based Solutions. Various other works have considered implementing vector commitments using elliptic curves with different focuses. For instance, a multilinear extension of KZG commitment has been proposed under the computation verification setting [14]. Another work [15] considered the problem of aggregating proofs of different vector commitments together.

Batch and Dynamic Updates. Although various works have investigated proof aggregation under the name of 'batching' [4], [16], [17], to our knowledge, our work is the first paper that systematically studies the batch update problem and proposes efficient solutions.

On the other hand, the problem of dynamically updating proofs has been studied by various works. A square-root update algorithm based on dividing the commitment into several segments has been proposed that trades much increased proof size with performance [11]. Hyperproofs [3] was the first aggregatable solution to support dynamic updates with a logarithmic complexity. However, it uses a Merkle-like structure and SNARKs to achieve aggregation. We showcase their complexity under the batch-update setting in Table 1.

2. Solution Overview

Our goal is to structure proof computations such that the update is easy. Towards that end, we observe that our results are based on our observation that KZG proofs can be broken down as matrix multiplications with Cauchy matrices. Therefore, by computing two such matrix multiplications, we can evaluate all relevant KZG proofs simultaneously. We then use this observation to optimize the computation of KZG proofs in several scenarios. We give a breakdown of our results in Figure 2.

2.1. Notation

We use arrows to describe vectors. For instance,

$$\vec{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1}).$$

When $\vec{\alpha}$ is a vector, we use $|\vec{\alpha}|$ to denote its size and α_i to denote its i-th element. Meanwhile, when $\vec{\alpha}$ and $\vec{\beta}$ are vectors, we use $\vec{\alpha} \circ \vec{\beta}$ to denote their element-wise product.

We use additive notation for group operations and square brackets $[x]_i$ to denote the group element $g^x \in G_i$ with respect to some generator g. We may omit i when the underlying group is clear. Typically, we use capital letters (e.g. C) to denote group elements. We use arrows with brackets (e.g. $[\vec{\alpha}]$) to denote the vector of group elements corresponding to the vector scalar, indexed by $[\alpha_i]$. We use $p(x)$ to represent a polynomial and $p(s)$ to represent the polynomial evaluated at s. To make matrix multiplication easier to represent, we also refer to vectors by their column representations.

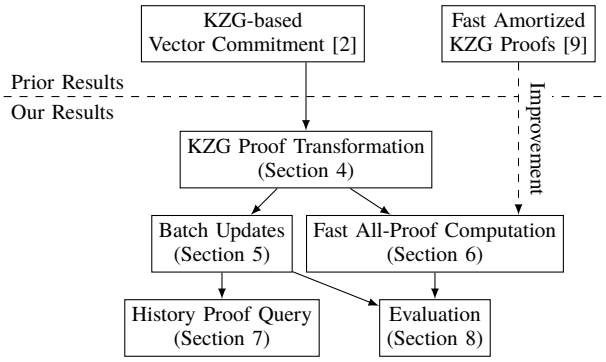

Figure 2: A view of our results. Our results are based on our observation of transforming KZG proofs as matrix multiplications with Cauchy matrices. We used the scheme developed by Tomescu et al. [2]. We also show that our results improve the fast amortized KZG proofs by Feist et al. [9].

Benefits of Additive Notation. Using the arguably less common additive notation for group operations allows us to represent linear transformations of group elements as matrices. For instance, consider some vector

$$\left[\vec{b}\right] = ([b_0], [b_1]).$$

We can leverage the benefit of the notation to represent an example of its linear transformation as

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} [b_0] \\ [b_1] \end{pmatrix} = \begin{pmatrix} [b_0] + 2[b_1] \\ 3[b_0] + 4[b_1] \end{pmatrix}.$$

We can then evaluate this linear transformation as long as we know $\left[\vec{b}\right]$, though not necessarily its scalar vector $\vec{b}$ directly.

As we will see later, we also use various techniques on matrix multiplication optimization to accelerate the evaluation we do in our algorithms. Such techniques are easier to represent in the matrix form, and can look unwieldy when using the more common multiplicative notation (e.g. (g^{b_0}, g^{b_1})).

2.2. KZG Proof Transformation

We start by outlining the construction of the KZG commitment [18], a polynomial commitment scheme that allows a succinct commitment of any polynomial up to degree $(n-1)$ and a succinct proof of its evaluation at any point, and show how it can be converted to a vector commitment scheme. We then give our result, a transformation that allows

TABLE 1: Comparison between various vector-commitment solutions. We picked the ones that focus on update complexity and the ones that motivated our work. We include the proof size, complexity of batch updates, whether it is possible to update proofs dynamically, whether aggregation is possible without SNARKs, and whether it is possible to update proofs without the full vector data. For BalanceProofs, we used their final two-layer construction appearing in their macrobenchmark.

	Proof Size	Batch Update	Dynamic Update	Aggregation Requirement	Update w/o Full Vector
Merkle [10]	1	$\left(\lvert\vec{\alpha}\rvert + \lvert\vec{\beta}\rvert\right) \log n$	Yes	SNARK	No
BalanceProofs [11]	$\sqrt{n}$	$\left(\lvert\vec{\alpha}\rvert + \lvert\vec{\beta}\rvert\right)^{5/4} \log\left(\lvert\vec{\alpha}\rvert + \lvert\vec{\beta}\rvert\right)$	Yes	-	Yes
Hyperproofs [3]	1	$\left(\lvert\vec{\alpha}\rvert + \lvert\vec{\beta}\rvert\right) \log n$	Yes	SNARK	No
aSVC [2]	1	$\lvert\vec{\alpha}\rvert \cdot \lvert\vec{\beta}\rvert$	No	-	Yes
Our Work	1	$\left(\lvert\vec{\alpha}\rvert + \lvert\vec{\beta}\rvert\right) \log^2\left(\lvert\vec{\alpha}\rvert + \lvert\vec{\beta}\rvert\right)$	No	-	Yes

us to represent KZG proofs as multiplications with Cauchy matrices.

Construction of the KZG Commitment. In the KZG commitment, a pairing-friendly elliptic curve (G_1, G_2) with $e : G_1 \times G_2 \to G_T$ is used. The commitment scheme relies on a trusted setup involving a random trapdoor scalar s not known to any party and a public parameter of

$$\left([1]_1, [s]_1, \left[s^2\right]_1, \ldots, \left[s^{n-1}\right]_1\right)$$

and

$$\left([1]_2, [s]_2, \left[s^2\right]_2, \ldots, \left[s^{n-1}\right]_2\right)$$

is published.

Suppose we are committing to some polynomial $p(x)$. The KZG commitment

$$C = [p(s)]_1$$

is the trapdoor scalar s evaluated on the polynomial $p(x)$. Such evaluation is possible by taking the coefficients of the polynomial $p(x)$ and multiplying them with the corresponding powers of s in the public parameter.

The proof of a single point $y = p(x)$ is given by

$$Q = \left[\frac{p(s) - y}{s - x}\right]_1.$$

This is again, possible to evaluate by first performing a polynomial long division with s as the variable and then multiplying the coefficients of the result with the corresponding powers of s in the public parameter.

To convert the KZG commitment to a vector commitment that commits to vector $\vec{v}$, a list of x values is chosen, and an interpolation polynomial $p(x)$ is constructed such that $p(x_i) = v_i$ for all $0 \le i < n$. Typically, we pick $x_i = \omega^i$ for some root of unity ω such that $\omega^n = 1$ for ease of computation. We gave a toy example of such construction in Figure 3.

With this manner, the KZG commitment C and proof Q_i can be written as

$$C = \sum_{i=0}^{n-1} v_i \left[\lambda_i(s)\right]_1,$$
$$Q_i = \left[\frac{C - v_i}{s - \omega^i}\right]_1$$

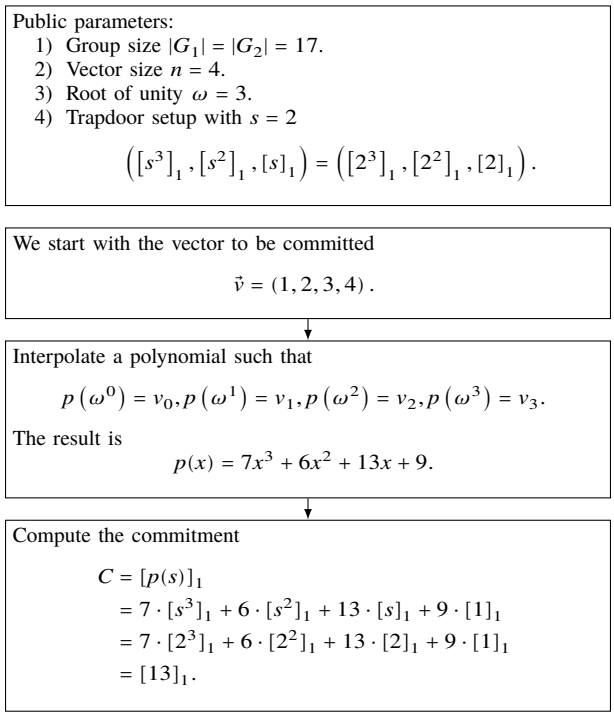

Figure 3: A toy example of a KZG commitment used as a vector commitment. The vector is transformed into a polynomial first, and then the polynomial is evaluated at the trapdoor scalar to get the commitment.

where $\lambda_i(s)$ is the Lagrange polynomial defined as

$$\lambda_i(s) = \prod_{j \ne i} \frac{s - \omega^j}{\omega^i - \omega^j}.$$

We leave a formal description of this functionality to Section 3.4.

Proof Transformation. Our result allows us to represent this type of KZG proof as matrix multiplications with Cauchy matrices. To begin, we construct the matrix

$$M_{i,j} = \begin{cases} 0 & (i = j) \\ \left(\omega^i - \omega^j\right)^{-1} & (i \ne j) \end{cases}.$$

Then, the commitment and the proof of the i-th element can be computed as

$$C = \sum_{i=0}^{n-1} v_i \left[l_i\right]_1,$$
$$Q_i = v_i w_i \left[l_i'(s)\right]_1 + a_i \left[l_i(s)\right]_1 - \left[b_i\right]_1$$

where $\vec{a}$ and $\left[\vec{b}\right]_1$ are the result vectors of the matrix multiplication

$$\vec{a} = M\left(\vec{v} \circ \vec{w}\right),$$
$$\left[\vec{b}\right]_1 = M\left(\vec{v} \circ \vec{w} \circ \left[\vec{l}(s)\right]_1\right)$$

and $\vec{w}$, $\left[\vec{l}(s)\right]_1$ and $\left[\vec{l}'(s)\right]_1$ are constant vectors that can be evaluated based on the KZG setup

$$w_i = \frac{\omega^i}{n},$$
$$\left[l_i(s)\right]_1 = \sum_{j=0}^{n-1} \omega^{i(n-1-j)} \left[s^j\right]_1,$$
$$\left[l_i'(s)\right]_1 = \sum_{j=0}^{n-1} (n-1-j)\, \omega^{i(n-2-j)} \left[s^j\right]_1.$$

We leave a formal proof of this result to Section 4.

2.3. Batch Updates

We now consider the batch-updating scenario where $|\vec{\alpha}|$ proofs need to be updated with $|\vec{\beta}|$ modifications. Specifically, we define the vector commitment functionality

$$\mathsf{VC.BatchUpdate}\left(\vec{\alpha}, \vec{Q}, \vec{\beta}, \Delta\vec{v}\right) \to \vec{Q}'$$

where we update the original proofs $\vec{Q}$ indexed at $\vec{\alpha}$ with modifications $\Delta\vec{v}$ indexed at $\vec{\beta}$ to produce a new vector of proofs.

Going back to the KZG commitment, we observe that Q_i is linear with respect to v_i. Therefore, to perform a batch update, it suffices to compute

$$\Delta Q_i = \Delta v_i w_i \left[l_i'(s)\right]_1 + \Delta a_i \left[l_i(s)\right]_1 - \left[\Delta b_i\right]_1$$

where

$$\Delta\vec{a} = M\left(\Delta\vec{v} \circ \vec{w}\right),$$
$$\left[\Delta\vec{b}\right]_1 = M\left(\Delta\vec{v} \circ \vec{w} \circ \left[\vec{l}(s)\right]_1\right)$$

We can then filter out all columns in M where $\Delta\vec{v}$ is zero, since the corresponding $\Delta\vec{a}$ and $\left[\Delta\vec{b}\right]_1$ will be zero. We also only need to keep track of the $|\vec{\alpha}|$ rows that we care about in M. Performing these two optimizations trims M to a $|\vec{\alpha}| \times |\vec{\beta}|$ matrix M' where

$$M'_{i,j} = \begin{cases} 0 & \left(\alpha_i = \beta_j\right) \\ \left(\omega^{\alpha_i} - \omega^{\beta_j}\right)^{-1} & \left(\alpha_i \neq \beta_j\right) \end{cases}.$$

We observe that this specific matrix multiplication can be done in $O\left(\left(|\vec{\alpha}| + |\vec{\beta}|\right) \log^2\left(|\vec{\alpha}| + |\vec{\beta}|\right)\right)$. We leave a description of the algorithm to Section 5.

2.4. Fast All-Proof Computation

In this section we consider the case where we want to compute all proofs in a KZG vector commitment, and give an algorithm that allows us to compute all proofs in $2n \log n + o(n \log n)$ elliptic multiplications. Instead of starting from polynomial coefficients like the previous work [8], [9], we directly compute the proofs by leveraging the matrix multiplication representation we developed in the previous section.

Observe that $M = \mathrm{diag}(\vec{\sigma}) \cdot C$ is the product of a diagonal matrix $\mathrm{diag}(\vec{\sigma})$ and a circulant matrix C, where

$$\vec{\sigma} = \left(\omega^0, \omega^{-1}, \omega^{-2}, \ldots, \omega^{-(n-1)}\right),$$
$$C_{i,j} = \begin{cases} 0 & (i = j) \\ \left(1 - \omega^{i-j}\right)^{-1} & (i \neq j). \end{cases}$$

Since circulant matrices are diagonalizable with the Fourier matrix [19], we can diagonalize C by

$$C = F^* \mathrm{diag}(F\vec{c})\, F$$

where F is the $n \times n$ Fourier matrix and

$$\vec{c} = \left(0, \frac{1}{1-\omega^{n-1}}, \frac{1}{1-\omega^{n-2}}, \ldots, \frac{1}{1-\omega}\right)$$

is the representation of C.

We can precompute this Fourier transformation as

$$\vec{\tau} = F\vec{c}$$
$$= \left(\frac{n-1}{2}, \frac{n-3}{2}, \frac{n-5}{2}, \ldots, \frac{-n+1}{2}\right).$$

We leave a proof of this computation to Section 6. Then

$$\vec{a} = \mathrm{diag}(\vec{\sigma})\, F^* \mathrm{diag}(\vec{\tau})\, \left(F\left(\vec{v} \circ \vec{w}\right)\right),$$
$$\left[\vec{b}\right]_1 = \mathrm{diag}(\vec{\sigma})\, F^* \mathrm{diag}(\vec{\tau})\, \left(F\left[\vec{v} \circ \vec{w} \circ \left[\vec{l}(s)\right]_1\right]\right).$$

We note that we only have to perform DFT and iDFT once on group elements during the computation of $\left[\vec{b}\right]_1$. Therefore, the computation of all proofs can be done in $2n \log n + o(n \log n)$ elliptic curve multiplications (See Section 6).

2.5. History Proof Query

Similar to bank statement queries in real life, a *history proof query* is a query that outputs proofs at any point in time throughout the blockchain lifetime. We show an algorithm that by leveraging our batch update algorithm, we can compute any version of the proof in time $O\left(|\vec{\alpha}| + |\vec{\beta}|\right) \log^3\left(|\vec{\alpha}| + |\vec{\beta}|\right)$, based on the current version of the proof.

In this algorithm, we treat blockchain transactions as a stream of $\left\langle \mathsf{UPDATE}, \beta_i, \Delta v_{\beta_i} \right\rangle$ updates. This view allows us to treat history computation as interleaving $\left\langle \mathsf{QUERY}, \alpha_j \right\rangle$ commands that output a single proof at the appropriate place in the stream. Since we know that the stream ended with

the current version of the proof, we can reverse the stream, negate all updates and start from the current version of the proof to answer all history queries. Processing history queries in this fashion leaves us with a query stream S that we have to work through. Figure 4 illustrates this process.

With the query stream S and the current vector of proofs $\vec{a}$, we now need to work through it, updating the proofs as we go and outputting the proofs at the appropriate time. Doing this naively requires updating every proof for every update command, which results in a time complexity of $O\left(|\vec{a}| \cdot \left|\vec{\beta}\right|\right)$. However, we observe that we can leverage our batch update algorithm and divide-and-conquer techniques to achieve a time complexity of $O\left(|\vec{a}| + \left|\vec{\beta}\right|\right) \log^3\left(|\vec{a}| + \left|\vec{\beta}\right|\right)$. We now describe this algorithm $\mathsf{VUpdate}\left(S, \vec{a}\right)$.

1) If $|S| = 1$, output the proof if it is a query and return.
2) Let
$$m = \left\lfloor \frac{|S|}{2} \right\rfloor.$$
Split
$$S = (S_0, S_1, \ldots, S_{|S|-1})$$
into two halves
$$S_l = (S_0, S_1, \ldots, S_{m-1})$$
and
$$S_r = (S_m, S_{m+1}, \ldots, S_{|S|-1}).$$
3) Pick out all proofs corresponding to queries in S_l and S_r as vector $\vec{\alpha}_l$ and $\vec{\alpha}_r$, respectively. Pick out all updates in S_l and S_r as vector $(\vec{\beta}_l, \Delta\vec{v}_l)$ and $(\vec{\beta}_r, \Delta\vec{v}_r)$, respectively.
4) Perform batch update of $(\vec{\beta}_l, \Delta\vec{v}_l)$ on $\vec{\alpha}_r$.
5) Recursively do
$$\mathsf{VUpdate}\left(S_l, \vec{\alpha}_l\right)$$
and
$$\mathsf{VUpdate}\left(S_r, \vec{\alpha}_r\right).$$

We give a proof of correctness and time complexity in Section 7.

2.6. Application

Applying the KZG commitment scheme with our batch update algorithm to the stateless blockchain scenario allows us to achieve the best of both worlds. On one hand, prove-serving nodes can enjoy an efficient quasi-linear time complexity for updating proofs. On the other hand, the succinctness of the KZG commitment allows a constant 48 bytes proof (with the BLS12-381 elliptic curve) to be attached to the block to verify *every* transaction in the block. Previously, such feat was only possible with various SNARK-based constructions [3], which needs a proof size of around 50 KB, as well as 100x slower aggregation time and 10x slower verification time compared to KZG-based schemes [11]. Figure 5 gives a comparison of complexity of various vector commitment schemes in the stateless blockchain scenario.

This complexity advantage translates into real-life performance improvements. In our evaluation (Section 8), we show that at block size 20,000 (resembling Ethereum), our algorithm is 5 times faster than the naive KZG approach. Alternatively, if we fix the process time to be one hour, our algorithm can handle around four times more users than the naive KZG approach.

3. Preliminary

Before we dive into the details of our analysis, we take some time to first review a few tricks that we will employ to speed up our algorithms. We also present a formalized definition of the KZG-based vector commitment inspired by the previous work [2], which will be the commitment scheme we work with in this paper.

3.1. Quasi-Linear Polynomial Operations

Given that the KZG commitment scheme operates on polynomials, efficient polynomial operations are essential to reduce computational complexity. While the naive complexity of multiplying two polynomials of degree n is $O\left(n^2\right)$, the application of discrete Fourier transform (DFT) has simplified many operations on polynomials to quasi-linear complexity. Here, we note a few that will be useful in our computation in Table 2.

TABLE 2: Complexity of polynomial operations. Here, we assume the polynomials are of degree n.

Operation	Complexity
Multiply	$O(n \log n)$
Interpolate n points [20]	$O\left(n \log^2 n\right)$
Evaluate on n points [20]	$O\left(n \log^2 n\right)$

Zeroing Polynomial. It is well-known that the coefficients of zeroing polynomial for $\vec{a} = (a_0, a_1, \ldots, a_{n-1})$
$$p(x) = \prod_{i=0}^{n-1} (x - a_i)$$
can be computed in $O\left(n \log^2 n\right)$ by the following divide-and-conquer algorithm [21]:

1) Divide $\vec{a}$ into two halves $\vec{a}_l$ and $\vec{a}_r$.
2) Recursively compute for $\vec{a}_l$ and $\vec{a}_r$, yielding two polynomials of degree $\frac{n}{2}$.
3) Multiply two polynomials together.

3.2. Limits on Prime Field

We observe that many previous papers dealing with polynomial evaluations [2], [22], [23] employ derivatives directly to convey the idea of L'Hospital's rule. In this paper, we sought to use a more intuitive notation. Specifically, we use $\lim_{x \to n} p(x)$ to denote a limit, even though $p(x)$ is defined on a prime field. While the prime field itself is discrete, we can nevertheless consider a natural projection from $\mathbb{Q}[\omega]$ to $\mathbb{Z}/p\mathbb{Z}$, where we replace ω from a root of unity on the complex field to a root of unity on the prime

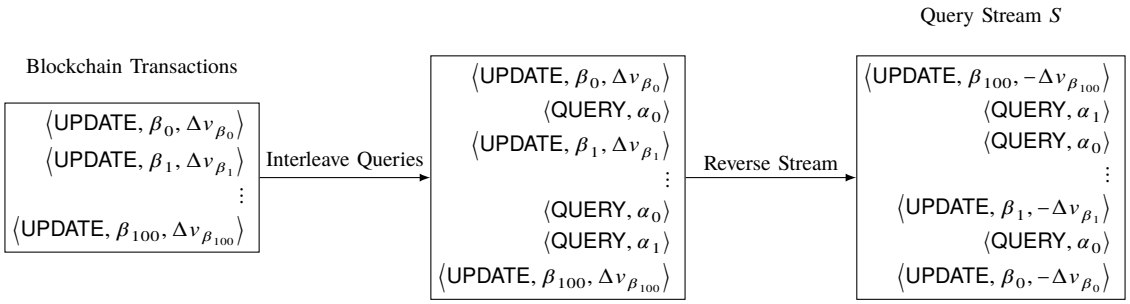

Figure 4: An example of processing history queries into a query stream S. We start with a list of blockchain transactions. We then insert queries into the stream. Finally, we reverse the stream and negate all updates. This process allows us to start with the current version of the proof and process all updates and queries in the sequential order.

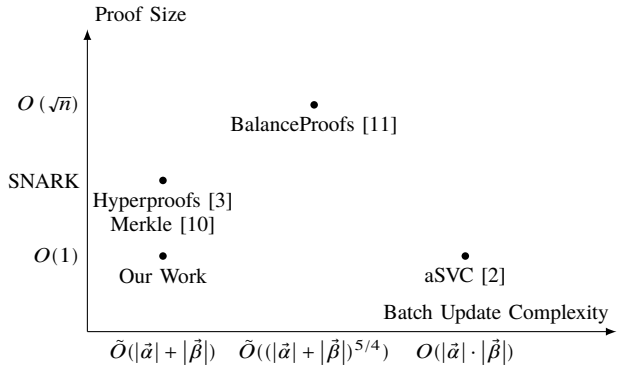

Figure 5: A comparison of various vector commitment schemes employed in stateless blockchain scenarios. $\tilde{O}(\cdot)$ denotes big-O notation with logarithmic factors omitted.

field. It's easy to confirm that both field operations remain homomorphic. This technique allows us to reason cases where both the numerator and the denominator are zeroes with similar techniques as mathematical analysis, such as

$$\lim_{x\to 1}\frac{(x-1)^2}{x-1}=\lim_{x\to 1}(x-1)=0$$

even if the equation is defined on a prime field.

3.3. Vector Commitment

Intuitively, a vector commitment [2] allows us to generate a succinct representation of a vector that allows us to verify any element in the vector with a proof. We define the functionality as follows.

1) $\mathsf{VC.KenGen}\left(1^{\lambda},n\right)\rightarrow pp$. Given the vector length n, sample the public parameter pp to be used in the commitment.
2) $\mathsf{VC.Commit}\left(pp,v\right)\rightarrow C$. Given a vector $\vec{v}$, generate its commitment c.
3) $\mathsf{VC.Prove}\left(pp,\vec{v},C,i\right)\rightarrow\pi_i$. Generate a proof π_i for the i-th element v_i.
4) $\mathsf{VC.Verify}\left(pp,C,i,v_i,\pi_i\right)\rightarrow\{0,1\}$. Verify the proof against element v_i and commitment C.

Moreover, recent works have demonstrated that the KZG commitment is aggregatable [2]. Specifically, it also provides the following functionality.

1) $\mathsf{VC.Aggregate}\left(pp,\{(i,\pi_i)\}\right)\rightarrow\pi$. Given a list of proofs $\{(i,\pi_i)\}$, aggregate them into a succinct proof π.
2) $\mathsf{VC.VerifyAggregate}\left(pp,C,\{(i,v_i)\},\pi\right)\rightarrow\{0,1\}$. Given a list of vector values $\{(i,v_i)\}$, verify them against the commitment C with the aggregated proof π.

For a full treatment of the definition of security properties, we refer the readers to the respective work for reference [2].

3.4. KZG-based Vector Commitment

KZG commitment [18] is a common polynomial commitment scheme that uses elliptic curve pairing to achieve $O(1)$ commitment and proof size for polynomials of some predetermined maximum degree $(n-1)$.

KZG commitment requires a trapdoor setup $s\in\mathbb{Z}^{|G_1|}$ not known to any party. The group elements

$$\left([1]_1,[s]_1,\left[s^2\right]_1,\ldots,\left[s^{n-1}\right]_1\right)$$

and

$$\left([1]_2,[s]_2,\left[s^2\right]_2,\ldots,\left[s^{n-1}\right]_2\right)$$

are published as public parameter.

Commitment. For a polynomial $p(x)$ of degree no more than $(n-1)$, its KZG commitment is computed by

$$C=[p(s)]_1,$$

which is computable given the public parameter by expanding $p(x)$ with its coefficients and using the proper powers of s.

Proof. The KZG proof of a single point $y=p(x)$ is given by

$$Q_x=\left[\frac{p(s)-y}{s-x}\right]_1.$$

Verification is carried out by verifying that

$$e\left(Q_x, [s-x]_2\right) \stackrel{?}{=} e\left(C - [y]_1, [1]_2\right).$$

Conversion to Vector Commitment. Typically in schemes such as aSVC [2], a vector $v = (v_0, v_1, \ldots, v_{n-1})$ is transformed to a polynomial by interpolation on a set of points $\left(\omega^i, v_i\right)$ with $0 \le i < n$ and ω being the root of unity such that $\omega^n = 1$.

This interesting construction allows for an efficient setup computation that can be completed in $O(n \log n)$ via a discrete Fourier transform and would otherwise take $O\left(n \log^2 n\right)$ in other generic cases. In our analysis, we assume this aSVC setup. We give the formalization of the KZG-based vector commitment below.

1) $\mathsf{VC.KenGen}\left(1^\lambda, n\right) \to pp$. Sample random

$$s \in \mathbb{Z}^{|G_1|}.$$

Compute the public parameter pp as

$$\left([1]_1, [s]_1, \left[s^2\right]_1, \ldots, \left[s^{n-1}\right]_1\right)$$

and

$$\left([1]_2, [s]_2, \left[s^2\right]_2, \ldots, \left[s^{n-1}\right]_2\right).$$

2) $\mathsf{VC.Commit}\left(pp, v\right) \to C$. Compute

$$C = \sum_{i=0}^{n-1} v_i \left[\lambda_i(s)\right]_1$$

where

$$\lambda_i(s) = \prod_{j=0, j\neq i}^{n-1} \frac{s - \omega^j}{\omega^i - \omega^j}$$

is the Lagrange interpolation polynomial.

3) $\mathsf{VC.Prove}\left(pp, v, C, i\right) \to \pi_i$. Compute

$$\pi_i = \left[\frac{\left(\sum_{j=0}^{n-1} v_j \left[\lambda_j(s)\right]_1\right) - v_i}{s - \omega^i}\right]_1.$$

4) $\mathsf{VC.Verify}\left(pp, C, i, v_i, \pi_i\right) \to \{0,1\}$. Verify that

$$e\left(\pi_i, \left[s - \omega^i\right]_2\right) \stackrel{?}{=} e\left(C - [v_i]_1, [1]_2\right).$$

Multi-Point Proof. The KZG proof of multiple points

$$y_0 = p(x_0), y_1 = p(x_1), \ldots, y_{m-1} = p(x_{m-1})$$

is given by

$$Q_x = \left[\frac{p(s) - \Lambda(s)}{z(s)}\right]_1,$$

where $\Lambda(s)$ is the Lagrange interpolation of the m points and

$$z(s) = \prod_{i=0}^{m-1} (s - x_i)$$

is the zeroing polynomial. Verification is carried out by verifying that

$$e\left(Q_x, [z(s)]_2\right) \stackrel{?}{=} e\left(C - [\Lambda(s)]_1, [1]_2\right).$$

Aggregation. Notice that we can aggregate single-point proofs

$$\left(Q_{i_0}, Q_{i_1}, Q_{i_2}, \ldots, Q_{i_{m-1}}\right)$$

into a multi-point proof by a weighted addition

$$Q = \sum_{j=0}^{m-1} \frac{Q_{i_j}}{\delta_j}$$

where

$$\delta_j = \prod_{k\neq j} \left(\omega^{i_j} - \omega^{i_k}\right)$$

can be computed by noticing that

$$\delta_j = \lim_{x \to \omega^{i_j}} \frac{\prod_{k=0}^{m-1} \left(x - \omega^{i_k}\right)}{x - \omega^{i_j}}.$$

Therefore, denote $g(x)$ as the zeroing polynomial

$$g(x) = \prod_{k=0}^{m-1} \left(x - \omega^{i_k}\right).$$

Then, by L'Hospital

$$\delta_j = \lim_{x \to \omega^{i_j}} g'(x) = g'\left(\omega^{i_j}\right).$$

Therefore, it suffices to find the coefficients of $g(x)$, and then evaluate $g'(x)$ on m different points. Both operations can be done in $O\left(m \log^2 m\right)$.

Tomescu et al. [2] first observed this property in their aSVC work and explained it based on partial fraction decomposition and derivative functions. We offer an explanation that does not require this knowledge in Appendix Appendix A.

We give a formalization of the aggregation functionality below.

1) $\mathsf{VC.Aggregate}\left(pp, \{(i, \pi_i)\}\right) \to \pi$. Compute

$$\pi = \sum_{j=0}^{m-1} \frac{\pi_{i_j}}{\delta_j}$$

where

$$\delta_j = \prod_{k\neq j} \left(\omega^{i_j} - \omega^{i_k}\right).$$

2) $\mathsf{VC.VerifyAggregate}\left(pp, C, \{(i, v_i)\}, \pi\right) \to \{0,1\}$. Verify that

$$e\left(\pi, [z(s)]_2\right) \stackrel{?}{=} e\left(C - [\Lambda(s)]_1, [1]_2\right)$$

where

$$\Lambda(s) = \sum_{j=0}^{m-1} v_{i_j} \lambda_{i_j}(s)$$

$$z(s) = \prod_{j=0}^{m-1} \left(s - \omega^{i_j}\right).$$

Security. KZG commitment is considered to be both hiding and binding, and its security proof relies on various assumptions on the elliptic curve including discrete logarithm (DL), t-polynomial Diffe-Hellman (t-polyDH) and t-strong Diffe-Hellman (t-BSDH). We invite the reader to the original paper for more details.

4. KZG Proof Transformation

Recall in Section 2.2 that we claimed KZG proofs can be represented as multiplications with Cauchy matrices, and that this result helps optimize multiple algorithms down the line. In this section, we give a formal proof of the result.

Theorem 4.1. *The KZG commitment and i-th KZG proof can be represented as*

$$\begin{aligned} C &= \sum_{i=0}^{n-1} v_i \left[l_i\right]_1, \\ Q_i &= v_i w_i \left[l_i'(s)\right]_1 + a_i \left[l_i(s)\right]_1 - \left[b_i\right]_1 \end{aligned}$$

where a and $[b]_1$ are the result vectors of the matrix multiplication

$$\begin{aligned} \vec{a} &= M\left(\vec{v} \circ \vec{w}\right), \\ \left[\vec{b}\right]_1 &= M\left(\vec{v} \circ \vec{w} \circ \left[\vec{l}(s)\right]_1\right), \\ M_{ij} &= \begin{cases} 0 & (i = j) \\ \left(\omega^i - \omega^j\right)^{-1} & (i \neq j) \end{cases} \end{aligned}$$

and $\vec{w}$, $\left[\vec{l}(s)\right]_1$ and $\left[\vec{l}'(s)\right]_1$ are constant vectors that can be evaluated based on the KZG setup

$$\begin{aligned} w_i &= \frac{\omega^i}{n}, \\ \left[l_i(s)\right]_1 &= \sum_{j=0}^{n-1} \omega^{i(n-1-j)} \left[s^j\right]_1, \\ \left[l_i'(s)\right]_1 &= \sum_{j=0}^{n-1} (n-1-j)\, \omega^{i(n-2-j)} \left[s^j\right]_1. \end{aligned}$$

Proof. Observe that

$$Q_i = \left[\frac{\sum_{i=0}^{n-1} v_i \lambda_i(s) - v_i}{s - \omega^i}\right]_1$$

Since the Lagrange interpolation term for the i-th point is

$$v_i \lambda_i(s) = \frac{\prod_{j \neq i}\left(s - \omega^j\right)}{\prod_{j \neq i}\left(\omega^i - \omega^j\right)} v_i.$$

We can rewrite it with

$$\begin{aligned} l_i(s) &= \prod_{j \neq i}\left(s - \omega^j\right), \\ w_i &= \frac{1}{\prod_{j \neq i}\left(\omega^i - \omega^j\right)}, \\ \lambda_i(s)\, v_i &= v_i w_i l_i(s). \end{aligned} \tag{1}$$

We first notice that w_i has a closed-form formula that is easy to compute.

Lemma 4.1. *w_i has a closed-form formula as*

$$w_i = \frac{1}{\prod_{j \neq i}\left(\omega^i - \omega^j\right)} = \frac{\omega^i}{n}.$$

Proof. Observe that we can apply the idea of analytic continuation as

$$w_i = \lim_{x \to \omega^i} \frac{x - \omega^i}{\prod_{j=0}^{n-1}\left(x - \omega^j\right)}.$$

Observe that

$$\prod_{j=0}^{n-1}\left(x - \omega^j\right) = x^n - 1$$

since $\left(1, \omega, \omega^2, \ldots, \omega^{n-1}\right)$ are exactly the roots of $(x^n - 1)$.

Therefore, we can perform a long division on

$$\begin{aligned} w_i &= \lim_{x \to \omega^i} \frac{x - \omega^i}{x^n - 1} \\ &= \lim_{x \to \omega^i} \frac{1}{x^{n-1} + \omega^i x^{n-2} + \ldots + \omega^{(n-1)i}} \\ &= \frac{1}{\omega^{i(n-1)} + \omega^i \omega^{i(n-2)} + \ldots + \omega^{(n-1)i}} \\ &= \frac{1}{n\omega^{i(n-1)}} = \frac{\omega^i}{n}. \end{aligned}$$

Then, we have our result. □

Moreover, denote $l_i'(s)$ as the quotient polynomial of $l_i(s)$ over $\left(s - \omega^i\right)$

$$l_i'(s) = \left\lfloor \frac{l_i(s)}{s - \omega^i} \right\rfloor. \tag{2}$$

Then, we will give an easy formula to compute

$$\left(\left[l_i(s)\right]_1, \left[l_i'(s)\right]_1\right)$$

for every $0 \leq i < n$.

Lemma 4.2. *$\left(\left[l_i(s)\right]_1, \left[l_i'(s)\right]_1\right)$ has a closed-form formula as*

$$\begin{aligned} \left[l_i(s)\right]_1 &= \sum_{j=0}^{n-1} \omega^{i(n-1-j)} \left[s^j\right]_1, \\ \left[l_i'(s)\right]_1 &= \sum_{j=0}^{n-1} (n-1-j)\, \omega^{i(n-2-j)} \left[s^j\right]_1. \end{aligned}$$

Proof. We first notice that

$$s^n - 1 = \prod_{i=0}^{n-1}\left(s - \omega^i\right)$$

since $\left(1, \omega, \omega^2, \ldots, \omega^{n-1}\right)$ are exactly the n roots of $(s^n - 1)$.

Therefore, we can expand $l_i(s)$ by a long division

$$\begin{aligned} l_i(s) &= \frac{\prod_{j=0}^{n-1}\left(s - \omega^j\right)}{s - \omega^i} \\ &= s^{n-1} + \omega^i s^{n-2} + \ldots + \omega^{(n-1)i}. \end{aligned} \tag{3}$$

Similarly, we can expand $l_i'(s)$ by a long division

$$\begin{aligned} l_i'(s) &= \left\lfloor \frac{l_i(s)}{s - \omega^i} \right\rfloor \\ &= s^{n-2} + 2\omega^i s^{n-3} + \ldots + (n-1)\, \omega^{(n-2)i}. \end{aligned} \tag{4}$$

Then, we have our result. □

For the commitment C, we observe that the commitment can be expanded based on Equation (1)

$$C = \left[\sum_{i=0}^{n-1} \lambda_i(s)\, v_i\right]_1 = \sum_{i=0}^{n-1} v_i w_i \left[l_i(s)\right]_1.$$

Now, we begin to assemble the proofs. Observe that we can expand the i-th proof Q_i by the i-th Lagrange term and all the rest of the terms

$$\begin{aligned} Q_i &= \left[\frac{C - v_i}{s - \omega^i}\right]_1 \\ &= \left[\frac{\lambda_i(s)\, v_i + \left(\sum_{j\neq i} \lambda_j(s)\, v_j\right) - v_i}{s - \omega^i}\right]_1 \\ &= \left[\frac{\lambda_i(s)\, v_i - v_i}{s - \omega^i} + \sum_{j\neq i} \frac{\lambda_j(s)\, v_j}{s - \omega^i}\right]_1. \end{aligned}$$

For the first term, we can rewrite it with our notation of the Lagrange term and the quotient polynomial from Equation (1) and Equation (2)

$$\begin{aligned} \frac{\lambda_i(s)\, v_i - v_i}{s - \omega^i} &= v_i \cdot \frac{\lambda_i(s) - 1}{s - \omega^i} \\ &= v_i \cdot \frac{w_i l_i(s) - 1}{s - \omega^i} \\ &= v_i w_i \cdot \left\lfloor \frac{l_i(s)}{s - \omega^i} \right\rfloor \\ &= v_i w_i l_i'(s). \end{aligned}$$

For the second term, observe that based on the definition of Equation (1)

$$\begin{aligned} l_i(s) - l_j(s) &= \frac{s^n - 1}{s - \omega^i} - \frac{s^n - 1}{s - \omega^j} \\ &= \frac{(s^n - 1)\left(\omega^i - \omega^j\right)}{(s - \omega^i)(s - \omega^j)} \\ &= \left(\omega^i - \omega^j\right) \cdot \frac{l_j(s)}{s - \omega^i}. \end{aligned}$$

We can hence substitute the term with $l_i(s) - l_j(s)$ and remove s from the denominator as

$$\begin{aligned} \sum_{j\neq i} \frac{\lambda_j(s)\, v_j}{s - \omega^i} &= \sum_{j\neq i} \frac{w_j l_j(s)\, v_j}{s - \omega^i} \\ &= \sum_{j\neq i} v_j w_j \cdot \frac{l_j(s)}{s - \omega^i} \\ &= \sum_{j\neq i} v_j w_j \cdot \frac{l_i(s) - l_j(s)}{\omega^i - \omega^j} \\ &= \left(\sum_{j\neq i} \frac{v_j w_j}{\omega^i - \omega^j}\right) l_i(s) - \sum_{j\neq i} \frac{v_j w_j l_j(s)}{\omega^i - \omega^j}. \end{aligned}$$

Putting these together gives us an expression of Q_i

$$\begin{aligned} Q_i = v_i w_i \left[l_i'(s)\right]_1 &+ \left(\sum_{j\neq i} \frac{v_j w_j}{\omega^i - \omega^j}\right)\left[l_i(s)\right]_1 \\ &- \sum_{j\neq i} \frac{v_j w_j \left[l_j(s)\right]_1}{\omega^i - \omega^j}. \end{aligned}$$

In order to simplify this expression, let us denote $\left(\vec{a}, \left[\vec{b}\right]_1\right)$ as vectors with

$$\begin{aligned} a_i &= \sum_{j\neq i} \frac{v_j w_j}{\omega^i - \omega^j}, \\ \left[b_i\right]_1 &= \sum_{j\neq i} \frac{v_j w_j \left[l_j(s)\right]_1}{\omega^i - \omega^j}. \end{aligned}$$

Observe that the expression of $\left(\vec{a}, \left[\vec{b}\right]_1\right)$ can be written as computation in a matrix form

$$\begin{aligned} \vec{a} &= M\left(\vec{v} \circ \vec{w}\right), \\ \left[\vec{b}\right]_1 &= M\left(\vec{v} \circ \vec{w} \circ \left[\vec{l}(s)\right]_1\right) \end{aligned}$$

where

$$\begin{aligned} M &= \begin{pmatrix} 0 & \frac{1}{\omega^0 - \omega^1} & \frac{1}{\omega^0 - \omega^2} & \cdots & \frac{1}{\omega^0 - \omega^{n-1}} \\ \frac{1}{\omega^1 - \omega^0} & 0 & \frac{1}{\omega^1 - \omega^2} & \cdots & \frac{1}{\omega^1 - \omega^{n-1}} \\ \vdots & & & & \\ \frac{1}{\omega^{n-1} - \omega^0} & \frac{1}{\omega^{n-1} - \omega^1} & \frac{1}{\omega^{n-1} - \omega^2} & \cdots & 0 \end{pmatrix} \\ &= \begin{cases} 0 & (i = j) \\ \left(\omega^i - \omega^j\right)^{-1} & (i \neq j) \end{cases}. \end{aligned}$$

This gives us the desired expression

$$Q_i = v_i w_i \left[l_i'(s)\right]_1 + a_i \left[l_i(s)\right]_1 - \left[b_i\right]_1$$

which finishes the proof. □

Pre-computing the Constants. Computing w_i is trivial in $O(n)$ by Lemma 4.1. However, computing $\left(\left[l_i(s)\right]_1, \left[l_i'(s)\right]_1\right)$ by the expansion from Equation (3) and Equation (4) is $O\left(n^2\right)$, which is not efficient. Here we give a simple algorithm that computes $\left(\left[l_i(s)\right]_1, \left[l_i'(s)\right]_1\right)$ in $O(n \log n)$.

Theorem 4.2. *There exists an $O(n \log n)$ algorithm that computes $\left(\left[l_i(s)\right]_1, \left[l_i'(s)\right]_1\right)$ for every $0 \leq i < n$.*

Proof. Based on our expansion from Equation (3) and Equation (4), we observe that computing $\left(\left[l_i(s)\right]_1, \left[l_i'(s)\right]_1\right)$ is the same as computing

$$\begin{aligned} F(x) &= \left[s^{n-1}\right]_1 + \left[s^{n-2}\right]_1 x + \ldots + [1]_1 x^{n-1}, \\ F'(x) &= \left[s^{n-2}\right]_1 + \left[2s^{n-3}\right]_1 x + \ldots + [n-1]_1 x^{n-2} \end{aligned}$$

on $x = \omega^i$, which can then be trivially done by discrete Fourier transform in $O(n \log n)$ for every $0 \leq i < n$. □

5. Batch Update

In Section 2.3, we give a brief description of our batch update algorithm. In this section, we give a formal description of the algorithm that allows us to update $|\vec{\alpha}|$ KZG proofs with $|\vec{\beta}|$ modifications in $O\left(\left(|\vec{\alpha}| + |\vec{\beta}|\right) \log^2\left(|\vec{\alpha}| + |\vec{\beta}|\right)\right)$ time.

Theorem 5.1. *There exists an*

$$O\left(\left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right) \log^2 \left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right)\right)$$

algorithm that updates $|\vec{\alpha}|$ KZG proofs with $\left|\vec{\beta}\right|$ modifications.

Recall that in this scenario, the update can be computed as

$$\Delta Q_i = \Delta v_i w_i \left[l_i'(s)\right]_1 + \Delta a_i \left[l_i(s)\right]_1 - \left[\Delta b_i\right]_1$$

where

$$\Delta \vec{a} = M\left(\Delta \vec{v} \circ \vec{w}\right),$$
$$\left[\Delta \vec{b}\right]_1 = M\left(\Delta \vec{v} \circ \vec{w} \circ \left[\vec{l}(s)\right]_1\right)$$

We can then filter out all columns in M where Δv is zero, since the corresponding Δa and $[\Delta b]_1$ will be zero. We also only need to keep track of the $|\vec{\alpha}|$ columns that we care about in M. Performing these two optimizations trims M to a $|\vec{\alpha}| \times \left|\vec{\beta}\right|$ matrix M' where

$$M'_{i,j} = \begin{cases} 0 & \left(\alpha_i = \beta_j\right) \\ \left(\omega^{\alpha_i} - \omega^{\beta_j}\right)^{-1} & \left(\alpha_i \neq \beta_j\right) \end{cases}.$$

Therefore, it suffices to give an algorithm that computes the matrix multiplication of M' and a vector γ in $O\left(\left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right) \log^2 \left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right)\right)$ time. We give a description of the algorithm below.

Lemma 5.1. *There exists an $O\left(\left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right) \log^2 \left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right)\right)$ algorithm that computes the matrix multiplication of a matrix M' and a vector γ.*

Proof. To isolate the parts in M' where $M'_{i,j} = 0$, we can first permute the rows and columns of $M'_{i,j}$ such that $M'[i,i] = 0$ for $0 \leq i < k$ for some k and M' is non-zero everywhere else. It's easy to show that such permutation is always possible, since the original M' only has $M_{i,j} = 0$ for $\alpha_i = \beta_j$.

We can then divide the matrix into three parts to isolate the zero entries in M':

1) Compute the multiplication with $M_{[0:k],[0:k]}$.
2) Compute the multiplication with $M_{[0:k],[k:|\vec{\beta}|]}$.
3) Compute the multiplication with $M_{[k:|\vec{\alpha}|],[0:|\vec{\beta}|]}$.

Figure 6 demonstrates an illustration of the three cases. Now let us consider the three parts separately.

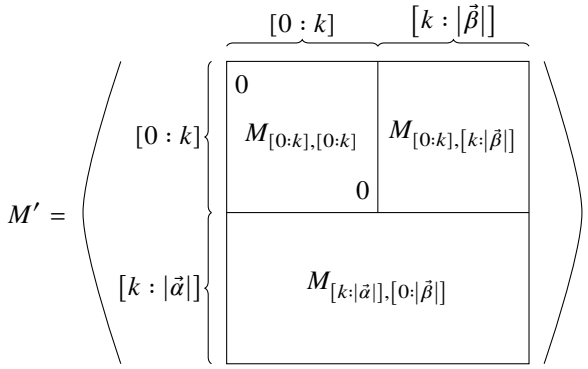

Figure 6: Illustration of the three cases in the matrix multiplication of M'. M' is divided into three parts: $M_{[0:k],[0:k]}$, $M_{[0:k],[k:|\vec{\beta}|]}$, and $M_{[k:|\vec{\alpha}|],[0:|\vec{\beta}|]}$. The main diagonal of $M_{[0:k],[0:k]}$ is zero, and the rest of the matrix is non-zero.

Case 1: $M_{[0:k],[0:k]}$. Observe that the update matrix under this case

$$M'_{i,j} = \begin{cases} 0 & (i = j) \\ \left(\omega^{\alpha_i} - \omega^{\alpha_j}\right)^{-1} & (i \neq j) \end{cases}$$

is exactly the matrix computed in Trummer's problem, with an $O\left(k \log^2 k\right)$ algorithm that solves the matrix multiplication with an arbitrary vector γ given by Gerasoulis et al. [23]. We adapt the algorithm below:

1) Compute the coefficients of the zeroing polynomial

$$g(x) = \prod_{i=0}^{k-1} \left(x - \omega^{\alpha_i}\right).$$

2) Compute the coefficients of the derivative $g'(x)$ and $g''(x)$.
3) Evaluate $g'\left(\omega^{\alpha_i}\right)$ and $g''\left(\omega^{\alpha_i}\right)$ for $i \in [0,k)$.
4) Interpolate the polynomial $h(x)$ that goes through every

$$\left(\omega^{\alpha_i}, \gamma_i g'\left(\omega^{\alpha_i}\right)\right)$$

for $i \in [0,k)$.
5) Compute the coefficients of the derivative $h'(x)$.
6) Evaluate $h'\left(\omega^{\alpha_i}\right)$ for $i \in [0,k)$.
7) Compute the result as

$$r_i = \frac{h'\left(\omega^{\alpha_i}\right) - \frac{1}{2}\gamma_i g''\left(\omega^{\alpha_i}\right)}{g'\left(\omega^{\alpha_i}\right)}.$$

Case 2: $M_{[0:k],[k:|\vec{\beta}|]}$ and Case 3: $M_{[k:|\vec{\alpha}|],[0:|\vec{\beta}|]}$. Similarly, for the rest of the two cases, the update matrix is simply a Cauchy matrix

$$M'_{i,j} = \left(\omega^{\alpha_i} - \omega^{\beta_j}\right)^{-1}.$$

We note that A. Gerasoulis has also offered an $O\left(\left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right) \log^2 \left(|\vec{\alpha}| + \left|\vec{\beta}\right|\right)\right)$ algorithm to compute the multiplication of a Cauchy matrix and a vector γ in his work [22]. We recall and adapt the algorithm below:

1) Compute the coefficients of the zeroing polynomial

$$g(x) = \prod_{i=0}^{|\vec{\beta}|-1} \left(x - \omega^{\beta_i}\right).$$

2) Compute the coefficients of the derivative $g'(x)$.
3) Evaluate $g\left(\omega^{\alpha_i}\right)$ for $i \in \left[0, |\vec{\alpha}|\right)$ and $g'\left(\omega^{\beta_i}\right)$ for $i \in \left[0, \left|\vec{\beta}\right|\right)$.
4) Interpolate the polynomial $h(x)$ that goes through every

$$\left(\omega^{\beta_i}, \gamma_i g'\left(\omega^{\beta_i}\right)\right)$$

for $i \in \left[0, \left|\vec{\beta}\right|\right)$.
5) Evaluate $h\left(\omega^{\alpha_i}\right)$ for $i \in \left[0, |\vec{\alpha}|\right)$.

6) Compute the result as

$$r_i = \frac{h(\omega^{\alpha_i})}{g(\omega^{\alpha_i})}.$$

Putting these three cases together finishes the algorithm. □

6. Fast Computation of All Proofs

Computing all KZG proofs fast is first explored by Feist et al. [9] and is considered to use $3n\log n + o(n\log n)$ elliptic curve multiplication operations [8] with overall time complexity $O(n\log n)$. As stated in Section 2.4, in this section, we give a different view of the algorithm that keeps the time complexity and reduces the number of elliptic curve multiplication operations to $2n\log n + o(n\log n)$.

Theorem 6.1. *There exists an $O(n\log n)$ algorithm with $2n\log n + o(n\log n)$ elliptic curve multiplication operations that computes all n KZG proofs for a vector of length n.*

Proof. Recall that

$$M_{i,j} = \begin{cases} 0 & (i=j) \\ \left(\omega^i - \omega^j\right)^{-1} & (i \neq j) \end{cases}.$$

Observe that $M = \operatorname{diag}(\vec{\sigma})\cdot C$ is the product of a diagonal matrix $\operatorname{diag}(\vec{\sigma})$ and a circulant matrix C, where

$$\vec{\sigma} = \left(\omega^0, \omega^{-1}, \omega^{-2}, \dots, \omega^{-(n-1)}\right),$$

$$C = \begin{pmatrix} 0 & \frac{1}{\omega^0-\omega^1} & \frac{1}{\omega^0-\omega^2} & \cdots & \frac{1}{\omega^0-\omega^{n-1}} \\ \frac{1}{\omega^0-\omega^{n-1}} & 0 & \frac{1}{\omega^1-\omega^2} & \cdots & \frac{1}{\omega^1-\omega^{n-1}} \\ \vdots & & & & \\ \frac{1}{\omega^0-\omega^1} & \frac{1}{\omega^{n-1}-\omega^1} & \frac{1}{\omega^{n-1}-\omega^2} & \cdots & 0 \end{pmatrix}.$$

Since circulant matrices are diagonalizable with the Fourier matrix [19], we can diagonalize C by

$$C = F^*\operatorname{diag}(F\vec{c})\,F$$

where F is the $n\times n$ Fourier matrix and

$$\vec{c} = \left(0, \frac{1}{1-\omega^{n-1}}, \frac{1}{1-\omega^{n-2}}, \dots, \frac{1}{1-\omega}\right)$$

is the representation of C.

We can precompute this Fourier transformation as

$$\begin{aligned} \vec{\tau} &= F\vec{c} \\ &= F\cdot\left(0, \frac{1}{1-\omega^{n-1}}, \frac{1}{1-\omega^{n-2}}, \dots, \frac{1}{1-\omega}\right)^T \\ &= \left(\frac{n-1}{2}, \frac{n-3}{2}, \frac{n-5}{2}, \dots, \frac{-n+1}{2}\right)^T \end{aligned}$$

by the following lemma.

Lemma 6.1. *τ_i has a closed-form formula as*

$$\tau_i = \frac{n-2i-1}{2}.$$

We provide a short proof of this lemma in Appendix B.

Then

$$\begin{aligned} \vec{a} &= \operatorname{diag}(\vec{\sigma})\,F^*\operatorname{diag}(\vec{\tau})\,(F(\vec{v}\circ\vec{w})), \\ \left[\vec{b}\right]_1 &= \operatorname{diag}(\vec{\sigma})\,F^*\operatorname{diag}(\vec{\tau})\left(F\left[\vec{v}\circ\vec{w}\circ\left[\vec{l}(s)\right]_1\right]\right). \end{aligned}$$

We note that multiplying by F and F^* can be understood as performing a DFT and iDFT, respectively, which can be done in $O(n\log n)$ time. Also, multiplying by a diagonal matrix can be done in $O(n)$. Therefore, a and $[b]_1$ can be computed in $O(n\log n)$ time complexity, and

$$Q_i = v_i w_i\left[l_i'(s)\right]_1 + a_i\left[l_i(s)\right]_1 - \left[b_i\right]_1$$

can be computed in $O(n)$. □

Elliptic Curve Multiplication Complexity. Observe that we need $\Theta(n\log n)$ elliptic curve scalar multiplications only when performing DFT and iDFT during the computation of $[b]_1$. Since DFT and iDFT use $n\log n$ multiplications every time, we only need $2n\log n + o(n\log n)$ elliptic curve scalar multiplications.

7. History Proof Query

In this section, we finish up the proof of the versioned update algorithm given in Section 2.5.

Theorem 7.1. *There exists an*

$$O\left(\left(|\vec{\alpha}| + |\vec{\beta}|\right)\log^3\left(|\vec{\alpha}| + |\vec{\beta}|\right)\right)$$

algorithm that updates $|\vec{\alpha}|$ KZG proofs with $|\vec{\beta}|$ modifications and outputs any version in-between the modifications.

Proof. Recall the algorithm in Section 2.5 as follows.

1) If $|S| = 1$, output the proof if it is a query and return.
2) Let
$$m = \left\lfloor \frac{|S|}{2} \right\rfloor.$$
Split
$$S = (S_0, S_1, \dots, S_{|S|-1})$$
into two halves
$$S_l = (S_0, S_1, \dots, S_{m-1})$$
and
$$S_r = (S_m, S_{m+1}, \dots, S_{|S|-1}).$$
3) Pick out all proofs corresponding to queries in S_l and S_r as vector $\vec{\alpha}_l$ and $\vec{\alpha}_r$, respectively. Pick out all updates in S_l and S_r as vector $(\vec{\beta}_l, \Delta\vec{v}_l)$ and $(\vec{\beta}_r, \Delta\vec{v}_r)$, respectively.
4) Perform batch update of $(\vec{\beta}_l, \Delta\vec{v}_l)$ on $\vec{\alpha}_r$.
5) Recursively do
$$\mathsf{VUpdate}(S_l, \vec{\alpha}_l)$$
and
$$\mathsf{VUpdate}(S_r, \vec{\alpha}_r).$$

Here we give a short proof on the correctness and time complexity.

Correctness. The correctness of the algorithm comes from the fact that the only way S_l and S_r affect each other is that an update in S_l will affect every query in S_r. The queries in S_l and the updates in S_r do not affect across the halves.

Time Complexity. Observe that the batch update complexity is $O\left(n \log^2 n\right)$. Therefore, we can write the time complexity $T(n)$ as

$$T(n) = 2T\left(\frac{n}{2}\right) + O\left(n \log^2 n\right).$$

We can then apply the master theorem to obtain that

$$T(n) = O\left(n \log^3 n\right).$$

□

8. Evaluation

We focus on evaluating our algorithm's main application to stateless blockchain and improvement from previous solutions. Towards that end, we evaluated our batch update algorithm and all-proof computation algorithm and compared with previous solutions under realistic scenarios.

8.1. Batch Updates

We implemented our batch update algorithm in Rust. We used the BLS12-381 curve provided by the `arkworks` library. An anonymous version of our code is available at https://anonymous.4open.science/r/buvc-rs-9B74. For comparison, we also implemented a naive algorithm that updates each proof individually, using the algorithm provided by aSVC [2].

We performed our benchmark on a local server with an Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz and 32 GB of RAM. We used the `criterion` library to measure the time taken for each algorithm, and approximated the time growth based on the algorithm complexity. We did not use multi-threading in any of the experiments to ensure fairness. In each experiment we set $|\vec{\alpha}| = |\vec{\beta}|$ to be the batch size. The results are shown in Figure 7.

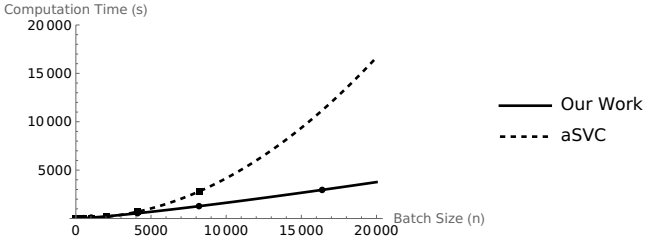

Figure 7: Computation time comparison of batch update algorithms. Our work is roughly five times faster at batch size 20,000.

Based on the figure, we observe that the improvement of time complexity is significant. Ethereum currently has a block size of around 20,000. At this batch size, our algorithm is roughly five times faster than the naive algorithm. To put it back to the real-world scenario, if we fix the processing time to be one hour, with 20,000 transactions, our algorithm can handle around 20,000 users. In contrast, the naive algorithm can only handle around 5,000 users. Our algorithm can handle around four times more users in the same time frame.

8.2. Fast Computation of All Proofs

We observe that our technique of computing all proofs in $2n \log n + o(n \log n)$ elliptic curve scalar multiplications is directly applicable to the Ethereum scenario [8], reducing its complexity from $\sqrt{6kn \log n}$ to $\sqrt{4kn \log n}$. We use their estimates of k and n to demonstrate this potential improvement in Table 3.

TABLE 3: Application of our technique in the Ethereum scenario, in terms of elliptic curve scalar multiplications, rounded to $.001 \times 10^6$.

$\log_2 n$	Original	Ours	Improvement
28	6.796×10^6	5.549×10^6	1.247×10^6
30	14.068×10^6	11.487×10^6	2.582×10^6
32	29.059×10^6	23.727×10^6	5.332×10^6

9. Conclusion

In this work, we tackled the computational challenges inherent in maintaining efficient, scalable proof-serving nodes within stateless blockchain systems by optimizing KZG-based vector commitments. Traditional methods for updating KZG proofs in response to state changes are computationally intensive, particularly as the number of users and transactions scales. To address this, we presented a novel batch update algorithm that achieves quasi-linear time complexity, reducing the computational cost of updating proofs from $O\left(|\vec{\alpha}| \cdot |\vec{\beta}|\right)$ to $O\left(\left(|\vec{\alpha}| + |\vec{\beta}|\right) \log^2(|\vec{\alpha}| + |\vec{\beta}|)\right)$. This advancement significantly lowers the resource requirements for proof-serving nodes, making it feasible to handle larger user groups and greater transaction volumes. We demonstrated that our method is approximately five times faster than naive approaches at Ethereum-level block sizes, establishing its practical utility in real-world blockchain systems.

We further explored fast computation of all KZG proofs while minimizing elliptic curve operations. Additionally, our proposed history proof query algorithm supports efficient generation of proofs at various stages of the state. Our contributions represent a substantial improvement in the scalability and efficiency of proof-serving nodes for stateless blockchain designs.

This work not only advances the technical foundations of stateless blockchain architectures but also paves the way for more decentralized and accessible blockchain networks capable of sustaining high transaction throughput without compromising efficiency.

References

[1] M. Christ and J. Bonneau, "Limits on revocable proof systems, with applications to stateless blockchains," Cryptology ePrint Archive, Report 2022/1478, 2022, https://eprint.iacr.org/2022/1478.

[2] A. Tomescu, I. Abraham, V. Buterin, J. Drake, D. Feist, and D. Khovratovich, "Aggregatable subvector commitments for stateless cryptocurrencies," in *SCN 20*, ser. LNCS, C. Galdi and V. Kolesnikov, Eds., vol. 12238. Springer, Heidelberg, Sep. 2020, pp. 45–64.

[3] S. Srinivasan, A. Chepurnoy, C. Papamanthou, A. Tomescu, and Y. Zhang, "Hyperproofs: Aggregating and maintaining proofs in vector commitments," in *USENIX Security 2022*, K. R. B. Butler and K. Thomas, Eds. USENIX Association, Aug. 2022, pp. 3001–3018.

[4] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to IOPs and stateless blockchains," in *CRYPTO 2019, Part I*, ser. LNCS, A. Boldyreva and D. Micciancio, Eds., vol. 11692. Springer, Heidelberg, Aug. 2019, pp. 561–586.

[5] S. Agrawal and S. Raghuraman, "KVaC: Key-Value Commitments for blockchains and beyond," in *ASIACRYPT 2020, Part III*, ser. LNCS, S. Moriai and H. Wang, Eds., vol. 12493. Springer, Heidelberg, Dec. 2020, pp. 839–869.

[6] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 315–334.

[7] L. Reyzin, D. Meshkov, A. Chepurnoy, and S. Ivanov, "Improving authenticated dynamic dictionaries, with applications to cryptocurrencies," in *FC 2017*, ser. LNCS, A. Kiayias, Ed., vol. 10322. Springer, Heidelberg, Apr. 2017, pp. 376–392.

[8] "Updating and generating Kate witnesses in amortized sqrt(n) time," https://ethresear.ch/t/updating-and-generating-kate-witnesses-in-amortized-sqrt-n-time/7520, accessed: April 26, 2024.

[9] D. Feist and D. Khovratovich, "Fast amortized KZG proofs," Cryptology ePrint Archive, Report 2023/033, 2023, https://eprint.iacr.org/2023/033.

[10] C. Papamanthou, S. Srinivasan, N. Gailly, I. Hishon-Rezaizadeh, A. Salumets, and S. Golemac, "Reckle trees: Updatable merkle batch proofs with applications," Cryptology ePrint Archive, Paper 2024/493, 2024. [Online]. Available: https://eprint.iacr.org/2024/493

[11] W. Wang, A. Ulichney, and C. Papamanthou, "BalanceProofs: Maintainable vector commitments with fast aggregation," Cryptology ePrint Archive, Report 2022/864, 2022, https://eprint.iacr.org/2022/864.

[12] M. Campanelli, D. Fiore, N. Greco, D. Kolonelos, and L. Nizzardo, "Incrementally aggregatable vector commitments and applications to verifiable decentralized storage," in *ASIACRYPT 2020, Part II*, ser. LNCS, S. Moriai and H. Wang, Eds., vol. 12492. Springer, Heidelberg, Dec. 2020, pp. 3–35.

[13] C. Papamanthou, E. Shi, R. Tamassia, and K. Yi, "Streaming authenticated data structures," in *EUROCRYPT 2013*, ser. LNCS, T. Johansson and P. Q. Nguyen, Eds., vol. 7881. Springer, Heidelberg, May 2013, pp. 353–370.

[14] C. Papamanthou, E. Shi, and R. Tamassia, "Signatures of correct computation," in *TCC 2013*, ser. LNCS, A. Sahai, Ed., vol. 7785. Springer, Heidelberg, Mar. 2013, pp. 222–242.

[15] S. Gorbunov, L. Reyzin, H. Wee, and Z. Zhang, "Pointproofs: Aggregating proofs for multiple vector commitments," in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM Press, Nov. 2020, pp. 2007–2023.

[16] S. Srinivasan, I. Karantaidou, F. Baldimtsi, and C. Papamanthou, "Batching, aggregation, and zero-knowledge proofs in bilinear accumulators," in *ACM CCS 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM Press, Nov. 2022, pp. 2719–2733.

[17] M. Dutta, C. Ganesh, S. Patranabis, S. Prakash, and N. Singh, "Batching-efficient RAM using updatable lookup arguments," Cryptology ePrint Archive, Paper 2024/840, 2024. [Online]. Available: https://eprint.iacr.org/2024/840

[18] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *ASIACRYPT 2010*, ser. LNCS, M. Abe, Ed., vol. 6477. Springer, Heidelberg, Dec. 2010, pp. 177–194.

[19] M. Rezghi and L. Eldén, "Diagonalization of tensors with circulant structure," *Linear Algebra and its Applications*, vol. 435, no. 3, pp. 422–447, 2011.

[20] H.-T. Kung, "Fast evaluation and interpolation," Ph.D. dissertation, Carnegie-Mellon University, 1973.

[21] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, 3rd ed. Cambridge University Press, 2013.

[22] A. Gerasoulis, "A fast algorithm for the multiplication of generalized Hilbert matrices with vectors," *Mathematics of Computation*, vol. 50, no. 181, pp. 179–188, 1988.

[23] A. Gerasoulis, M. D. Grigoriadis, and L. Sun, "A fast algorithm for Trummer's problem," *SIAM Journal on Scientific and Statistical Computing*, vol. 8, no. 1, pp. s135–s138, 1987. [Online]. Available: https://doi.org/10.1137/0908017

Appendix A. Proof of Aggregation Formula

In this appendix, we finish the proof of the KZG proof aggregation formula.

Theorem A.1. *We can aggregate single proofs into an aggregated proof with*

$$Q = \sum_{j=0}^{m-1} \frac{Q_{i_j}}{\delta_j}.$$

Proof. Observe that

$$Q_{i_j} = \left[\frac{p(s)}{s - \omega^{i_j}} - \frac{v_{i_j}}{s - \omega^{i_j}} \right]_1.$$

Therefore,

$$\frac{Q_{i_j}}{\delta_j} = \left[\frac{p(s)}{\left(s - \omega^{i_j}\right) \prod_{k \neq j} \left(\omega^{i_j} - \omega^{i_k}\right)} - \frac{v_{i_j}}{\left(s - \omega^{i_j}\right) \prod_{k \neq j} \left(\omega^{i_j} - \omega^{i_k}\right)} \right]_1.$$

We can compare this with

$$Q = \frac{p(s) - I(s)}{Z(s)} = \left[\frac{p(s)}{\prod_{j=0}^{m-1} \left(s - \omega^{i_j}\right)} - \frac{\sum_{j=0}^{m-1} \frac{\prod_{k \neq j} (s - \omega^{i_k})}{\prod_{k \neq j} \left(\omega^{i_j} - \omega^{i_k}\right)} v_{i_j}}{\prod_{j=0}^{m-1} \left(s - \omega^{i_j}\right)} \right]_1.$$

Observe that

$$\frac{v_{i_j}}{\left(s - \omega^{i_j}\right) \prod_{k \neq j} \left(\omega^{i_j} - \omega^{i_k}\right)} = \frac{\frac{\prod_{k \neq j} (s - \omega^{i_k})}{\prod_{k \neq j} \left(\omega^{i_j} - \omega^{i_k}\right)} v_{i_j}}{\prod_{j=0}^{m-1} \left(s - \omega^{i_j}\right)}.$$

Therefore, it is sufficient that we prove

$$\frac{p(s)}{\prod_{j=0}^{m-1}\left(s-\omega^{i_j}\right)}=\sum_{j=0}^{m-1}\frac{p(s)}{\left(s-\omega^{i_j}\right)\prod_{k\neq j}\left(\omega^{i_j}-\omega^{i_k}\right)}.$$

Observe that the Lagrange interpolation on every $\left(\omega^{i_j},1\right)$ is the constant function $f(s)=1$. Therefore,

$$f(s)=1=\sum_{j=0}^{m-1}\frac{\prod_{k\neq j}\left(s-\omega^{i_k}\right)}{\prod_{k\neq j}\left(\omega^{i_j}-\omega^{i_k}\right)}.$$

Dividing both sides by $\prod_{j=0}^{m-1}\left(s-\omega^{i_j}\right)$ gives us the desired result

$$\frac{1}{\prod_{j=0}^{m-1}\left(s-\omega^{i_j}\right)}=\sum_{j=0}^{m-1}\frac{1}{\left(s-\omega^{i_j}\right)\prod_{k\neq j}\left(\omega^{i_j}-\omega^{i_k}\right)}$$

which we can plug back in to verify the original equation. □

Appendix B. Proof of Fourier Transformation Computation

In this appendix, we prove the following computation used in Section 6.

Lemma 6.1. *τ_i has a closed-form formula as*

$$\tau_i=\frac{n-2i-1}{2}.$$

Proof. Observe that

$$\tau_i=\frac{\omega^i}{1-\omega^{n-1}}+\frac{\omega^{2i}}{1-\omega^{n-2}}+\ldots+\frac{\omega^{(n-1)i}}{1-\omega}.$$

Noticing that the j-th term

$$\begin{aligned}\frac{\omega^{ij}}{1-\omega^{n-j}}&=\frac{\omega^{ij}\prod_{k\neq j,k\neq 0}\left(1-\omega^{n-k}\right)}{\prod_{k=1}^{n-1}\left(1-\omega^{n-k}\right)}\\&=\frac{\omega^{ij}\prod_{k\neq(n-j),k\neq 0}\left(1-\omega^{k}\right)}{\prod_{k=1}^{n-1}\left(1-\omega^{k}\right)}.\end{aligned}$$

We can then apply the idea of analytic continuity and the expansion of $l_i(s)$ from Equation (3)

$$\begin{aligned}\frac{\omega^{ij}}{1-\omega^{n-j}}&=\lim_{x\to 1}\frac{\omega^{ij}\prod_{k\neq(n-j)}\left(x-\omega^{k}\right)}{\prod_{k=0}^{n-1}\left(x-\omega^{k}\right)}\\&=\lim_{x\to 1}\frac{\omega^{ij}l_{(n-j)}(x)}{x^n-1}\\&=\lim_{x\to 1}\frac{\omega^{ij}x^{n-1}+\omega^{(i-1)j}x^{n-2}+\ldots+\omega^{(i-n+1)j}}{x^n-1}.\end{aligned}$$

Then, we can add up the terms of τ_i

$$\begin{aligned}\tau_i&=\sum_{j=1}^{n-1}\lim_{x\to 1}\frac{\omega^{ij}x^{n-1}+\omega^{(i-1)j}x^{n-2}+\ldots+\omega^{(i-n+1)j}}{x^n-1}\\&=\lim_{x\to 1}\frac{\sum_{k=0}^{n-1}\left(\sum_{j=1}^{n-1}\omega^{(i-k)j}\right)x^{n-k-1}}{x^n-1}.\end{aligned}$$

Here we observe that $\sum_{j=1}^{n-1}\omega^{(i-k)j}$ can be simplified with the following lemma.

Lemma B.1. *for every integer $1\leq\psi<n$*

$$\sum_{j=1}^{n-1}\omega^{j\psi}=-1.$$

Proof. Let $\phi=\gcd(\psi,n)$. Observe that ω^{ψ} is a generator for the group G_ϕ generated by ω^{ϕ}. G_ϕ has exactly $\frac{n}{\phi}$ elements. Therefore, the sum

$$\begin{aligned}\sum_{j=0}^{n-1}\omega^{j\psi}&=\phi\left(1+\omega^{\psi}+\omega^{2\psi}+\ldots+\omega^{(n-1)\psi}\right)\\&=\psi\cdot\frac{1-\omega^{n\psi}}{1-\omega}=0.\end{aligned}$$

That is to say

$$\sum_{j=1}^{n-1}\omega^{j\psi}=\sum_{j=0}^{n-1}\omega^{j\psi}-1=-1$$

and we have our proof. □

Therefore, τ_i has exactly one term where $i-k=0$ and

$$\sum_{j=1}^{n-1}\omega^{(i-k)j}=n-1.$$

For all the rest of the terms

$$\sum_{j=1}^{n-1}\omega^{(i-k)j}=-1.$$

We can then write it as

$$\begin{aligned}\tau_i&=\lim_{x\to 1}\frac{-\left(\sum_{j=0}^{n-1}x^j\right)+nx^{n-i-1}}{x^n-1}\\&=\lim_{x\to 1}\frac{-\left(\sum_{j=1}^{n-1}j\cdot x^{j-1}\right)+n(n-i-1)x^{n-i-2}}{nx^{n-1}}\\&=\frac{-\frac{n(n-1)}{2}+n(n-i-1)}{n}\\&=\frac{n-2i-1}{2}\end{aligned}$$

by L'Hospital's rule. □