Distributed Vector Commitments and Their Applications

Rui Gao* Huaqun Wang* Zhiguo Wan † Yuncong Hu ‡§¶

Abstract

Vector commitment (VC) schemes enable a prover to commit to a vector and later open any position with a short proof. However, existing VC schemes are designed for centralized settings, and cannot work in decentralized systems, where the input vector is distributed across multiple machines. Similarly, traditional VC schemes cannot leverage distributed parallel computation across multiple machines for acceleration.

To tackle this issue, we introduce a new notion—distributed VC (DVC), which allows multiple machines, each holding only a subvector of the input vector, to collectively commit to the entire vector and generate position proofs in a distributed manner. To the best of our knowledge, there is no prior work on DVCs and no existing work can trivially derive an efficient DVC scheme. The key challenge is that both commitments and proofs depend on the entire vector, while no single machine holds the complete vector in distributed settings.

We propose the first DVC scheme, HLE-DVC, which leverages M machines to process the distributed vector $\mathbf{v}$ of length N in parallel, with each machine holding a subvector of length $\frac{N}{M}$. HLE-DVC achieves compact proof size-$O(\log M)$ and allows each machine to generate all its position proofs in a single communication round, with communication cost $O(\log M)$ and computation cost $O(\frac{N \log N}{M})$. Moreover, HLE-DVC supports batch proving, proof aggregation, and efficient updates. We conduct the experiments and open-source the code. Using 256 machines to generate all proofs for a committed vector of length 2^{30} takes 17,515 seconds. This achieves a 256× parallel speedup over HLE-DVC on a single machine, and is 142× faster than Hyperproofs (a famous single machine VC scheme). The communication cost per machine is 0.768 KB.

*Nanjing University of Posts and Telecommunications, The State Key Laboratory of Tibetan Intelligence.

†Hangzhou Normal University.

‡Department of Computer Science and Engineering, Shanghai Jiao Tong University.

§Corresponding author.

¶This paper is the extended version of USENIX 2026 ,https://www.usenix.org/system/files/conference/usenixsecurity26/sec26_prepub_gao.pdf

1 Introduction

A vector commitment (VC) scheme allows a prover to commit a vector $\mathbf{v} = (v_0, ..., v_N)$ and then generate a position proof π_i to prove that the i-th element v_i is equal to v^*. Afterwards, a verifier can verify whether $v_i = v^*$ based on the commitment and the proof π_i. The VC scheme has a wide range of applications and has been used in many scenarios such as the stateless blockchain [1–5], verifiable elementary databases (VEDB) [6–10] and electronic auditing [11, 12].

Although vector commitment (VC) schemes have a wide range of applications, they cannot deal with two types of scenarios: 1. **Distributed architectures**. *Traditional VC schemes cannot work in distributed architectures.* Take the VEDB as an example, where a server holds an elementary database, which can be regarded as a vector storing data from multiple clients (e.g., a bank storing deposit amount for its clients' accounts). The server commits to the database using a VC scheme. When queried about a specific position, it returns the value along with a position proof, demonstrating the correctness of the result. However, in real-world deployments, due to legal constraints, or the sheer size of the database, the complete database is often distributed across multiple machines. Traditional VC schemes cannot support committing and proof generation in distributed settings since no one holds the entire vector. 2. **Parallel acceleration**. *The proof generation in VC schemes incurs a high overhead, and this workload cannot be alleviated through parallelization.* When the vector length reaches 2^{30}, it would take about 50 days to generate position proofs for all elements (aSVC [13] on a cloud server). Such inefficiency severely hinders the practical deployment of applications based on VC schemes. To alleviate the computational burden of VC schemes, a natural idea is to distribute the computation across M machines to run in parallel, thereby achieving up to an $M\times$ speedup. However, existing VC schemes do not support distributed parallel computation for acceleration.

To deal with the distributed setting and address the challenges mentioned above, we propose the concept of dis-

tributed VC (DVC)[1], which enables M machines to distributedly generate vector commitments and position proofs. The input vector $\mathbf{v}$ of length N involves M subvectors: $\mathbf{v} = (\mathbf{v}_0, \ldots, \mathbf{v}_{M-1})$. Each machine $\mathcal{P}_k$ ($k \in [0, M)$) is responsible for generating position proofs for all elements in the subvector $\mathbf{v}_k = (v_{k,0}, ..., v_{k,n-1})$ of length $n = \frac{N}{M}$ ($v_{k,i}$ is the i-th element in the k-th subvector $\mathbf{v}_k$).

The DVC scheme is particularly well-suited for two application scenarios: 1). In distributed settings, DVC enables generating commitments and position proofs distributedly. Thus, the DVC can extend single-machine VC applications to more practical distributed versions, such as distributed verifiable elementary database (DVEDB), multinational banking systems, and stateless blockchain sharding. 2). In centralized settings such as proof of reserve (PoR [15]), if the input vector of VC schemes is too long for a single machine to generate all position proofs efficiently, the prover can split the vector and distribute it across M machines, which then compute in parallel and interactively, achieving an $M\times$ speedup.

Challenges. In a distributed setting, each machine holds only one subvector, yet the generation of the position proof relies on the complete vector. This necessitates communication among machines to generate proofs. Intuitively, producing position proofs for all elements incurs a communication overhead of at least $\mathrm{O}(N)$ per machine, which is highly inefficient. Moreover, this distributed architecture also poses challenges to implementing additional functionalities, such as additive homomorphism, aggregation, and maintainability. To the best of our knowledge, no DVC scheme has ever been proposed in the literature. Here we list three naive DVC design ideas:

(1) ***DVC schemes from distributed polynomial commitment schemes (DPCS)***. DPCS [16–20] allows for the distributed generation of polynomial commitments and evaluation proofs. A natural approach is to extend the vector $\mathbf{v}$ into a distributed polynomial $\mathcal{F}(\cdot)$, and generate evaluation proofs distributedly as position proofs. So DPCS-DVC schemes can run evaluation proving algorithms N times for generating position proofs for all elements, which incurs the communication cost per machine of at least $\mathrm{O}(N)$. A promising improved approach is to use an FFT-like method for all proofs' generation. KZG-based PCS can generate N evaluation proofs for N points in $\mathrm{O}(N\log N)$ time using an FFT-like approach [21–23]. However, implementing this method in a distributed setting introduces $\mathrm{O}(\log N)$ communication rounds [24] and $\mathrm{O}(\frac{N}{M})$ communication overhead per machine.

(2) ***DVC scheme derived from homomorphic VC schemes***. An existing additive homomorphic[2] VC scheme can be trivially transformed into a DVC scheme in the following way: Each machine $\mathcal{P}_k$ set a new vector $\hat{\mathbf{v}}_k$ of length N: $\hat{\mathbf{v}}_k = (\overline{\mathbf{v}_0}, ..., \overline{\mathbf{v}_{M-1}})$, where $\overline{\mathbf{v}_k} = \mathbf{v}_k$, but for other $d \in [0, M) \wedge d \neq k$, $\overline{\mathbf{v}_d} = (0, ..., 0) = \mathbf{0}^n$, which is a subvector of length n filled with 0. Each $\mathcal{P}_k$ can produce a sub-commitment of $\hat{\mathbf{v}}_k$ and all sub-position proofs for all elements in $\hat{\mathbf{v}}_k$. Then, by leveraging additive homomorphism, the complete commitment and position proofs can be obtained by summing all sub-commitments and sub-position proofs, respectively.

We can instantiate this idea using aSVC, resulting in an aSVC-DVC scheme with generating all proofs time of $\mathrm{O}(N\log N)$ and communication overhead of $\mathrm{O}(N)$. It still incur high computation and communication costs.

(3) ***DVC scheme from Merkle commitment scheme***. Another DVC design approach is based on the Merkle tree [25]: each machine $\mathcal{P}_k$ generates a Merkle sub-tree T_k for his subvector $\mathbf{v}_k$ and publishes the root of the sub-tree. Then machines construct a Merkle tree T, where the leaf nodes are set as the roots of sub-trees $[\mathrm{T}_k]_{k\in[0,M)}$. The root of the tree T serves as the commitment to the vector $\mathbf{v}$. The machine $\mathcal{P}_k$ can generate all position proofs for elements in $\mathbf{v}_k$ according to the tree T_k and T. This approach is efficient as it costs per machine $\mathrm{O}(\frac{N}{M})$ computational cost and $\mathrm{O}(\log M)$ communication overhead. However, since the use of hash functions, Merkle-DVC cannot achieve additive homomorphism, so that it is hard for this approach to support advanced functionalities such as batch proofs, aggregation proofs, and additive homomorphism.

In summary, existing schemes cannot be trivially transformed into an efficient DVC scheme with both low communication overhead and full functionality.

Our approach. Firstly, based on the low degree extension (LDE) and multilinear extension (MLE), we propose a new extension method called the hybrid lagrange extension (HLE). As shown in Figure 1, HLE first uses LDE to extend each subvector of length $n = N/M$ into a univariate polynomial of degree less than n. Subsequently, HLE employs MLE to aggregate all univariate polynomials, forming a multivariate polynomial. Secondly, we propose a decomposition method for HLE, which enables distributed proof generation. This makes our DVC scheme only costs $\mathrm{O}(\log M)$ communication overhead per machine to generate all proofs. Another highlight of this decomposition method is that it can transform all existing MLE-based VC schemes into DVC schemes.

Based on the HLE and the decomposition method, we construct the HLE-DVC scheme. The machines can generate commitments in $\mathrm{O}(N/M)$ time with $\mathrm{O}(1)$ communication overhead. The time cost for generating position proofs for all elements is $\mathrm{O}(\frac{N}{M}\cdot\log N)$, and the communication cost per machine is $\mathrm{O}(\log M)$. The proof size is $\mathrm{O}(\log M)$. After an element is updated, all position proofs can be updated in $\mathrm{O}(\frac{N}{M}+\log M)$ time. Additionly, HLE-DVC supports aggregating proofs and batch proving.

In Table 1, we summarize the complexities of the three

[1] Some previous works [7, 14] have already introduced a different concept of DVC, which refers to the ability to update a position proof after changes to the vector without access to the entire vector. All these schemes cannot support distributed proof-generation. The DVC proposed in this paper refers to generating the commitment and proofs in distributed setting.

[2] Given vectors $\mathbf{v}_{1st}$, $\mathbf{v}_{2nd}$ and $\mathbf{v}_{sum} = \mathbf{v}_{1st} + \mathbf{v}_{2nd}$, a VC scheme is additive homomorphic if $\mathsf{Commit}(\mathbf{v}_{1st}) + \mathsf{Commit}(\mathbf{v}_{2nd}) = \mathsf{Commit}(\mathbf{v}_{sum})$. The proofs for $\mathbf{v}_{sum}$ can be obtained by summing the proofs for $\mathbf{v}_{1st}$ and $\mathbf{v}_{2nd}$.

Table 1: Comparison among 3 strawman DVC schemes derived from existing works, our HLE-DVC and two promising DVCs (introduced in Section 2.2) obtained by applying our decomposition lemma from Section 4 to existing MLE-based VC schemes. N denotes the vector length, M denotes the number of machines. Commit time is the per machine time cost for committing a vector. $\Pi_{d,i}$ is a proof for position (d,i). $\Pi_{\mathbf{I}}$ is a batch proof for an index set $\mathbf{I}$ of length $|\mathbf{I}|$. ProveAll time lists the time cost for generating all position proofs. Agg time lists the time cost for aggregating $|\mathbf{I}|$ individual proofs. UpdAll time lists the time cost for updating all position proofs after an element is changed. Regarding aggregation, "s:" denotes single-subvector aggregation, and "c:" denotes cross-subvectors aggregation. When s/c are not explicitly indicated, it is assumed that the aggregation complexity is the same for both cases. Homo∼? means additively homomorphism; the Comm∼ size refers to the communication size per machine.

	DVC Schemes	Commit time	$\lvert\Pi_{d,i}\rvert$	$\lvert\Pi_{\mathbf{I}}\rvert$	ProveAll time	Agg time	UpdAll time	Homo∼?	Comm∼ size
Strawmans	**KZG-DPCS-DVC** [17]	N/M	1	1	$\frac{N^2}{M}$	$\lvert\mathbf{I}\rvert\log^2\lvert\mathbf{I}\rvert$	$\frac{N}{M}$	✓	N
	aSVC-DVC [13]	N/M	1	1	$N\log N$	$\lvert\mathbf{I}\rvert\log^2\lvert\mathbf{I}\rvert$	N	✓	N
	Merkle-DVC [25]	N/M	$\log N$	×	$\frac{N}{M}$	×	$\log N$	×	$\log M$
Our works	**HLE-DVC**	N/M	$\log M$	s: $\log M$ c: $\log(M\log M)$	$\frac{N}{M}\log N$	s: $\lvert\mathbf{I}\rvert\log^2\lvert\mathbf{I}\rvert$ c: $\lvert\mathbf{I}\rvert\log^2(\lvert\mathbf{I}\rvert)+M\log M$	$\frac{N}{M}+\log M$	✓	$\log M$
	Hyperproofs-DVC [3]	N/M	$\log N$	$\log(\lvert\mathbf{I}\rvert\log N)$	$\frac{N}{M}\log N$	$\lvert\mathbf{I}\rvert\log N$	$\log N$	✓	$\log M$
	[26]-DVC	N/M	$\log M$	s: $\log M$ c: $M\log M$	$\frac{N}{M}\log N$	$\lvert\mathbf{I}\rvert\log^2\lvert\mathbf{I}\rvert$	$\frac{N}{M}+\log M$	✓	$\log M$

strawman constructions, our proposed scheme, and two DVC schemes derived from our decomposition method and existing VC works (introduced in Section 2).

1.1 Contributions

In this paper, we make the following contributions:

1. We are the first to identify the distribution challenges and scalability limitations in the VC schemes, and to propose the notion of distributed vector commitment (DVC), which enables M machines to generate the vector commitment and position proofs in a distributed manner.

2. We propose HLE-DVC, the first DVC scheme, which introduces the following key innovations:

- We introduce a new vector extension method, **Hybrid Lagrange Extension (HLE)** based on the MLE and LDE. LDE enables a small proof size of $O(\log M)$ and supports a native aggregation method. The tree structure of MLE leads to low communication cost.
- For HLE, we propose a specialized decomposition method that can be executed in a distributed manner. Furthermore, we develop a distributed approach for generating all position proofs. It realizes low proof size-$O(\log M)$, low per machine communication cost—$O(\log M)$, and low time cost —$O(\frac{N}{M}\cdot\log N)$. This decomposition method can transform existing MLE-based VC schemes into DVC schemes, which can be seen as an independent interest.
- We further present algorithms for batch proving, updates, and aggregation. Additionally, HLE-DVC supports maintainability through the compiler proposed in BalanceProofs [27], ensuring sublinear-time updates.

To the best of our knowledge, HLE-DVC is the first DVC scheme that achieves such low computational and communication overheads. To accommodate different scenarios, we present two modified versions in the Appendix H.1.

3. Formally, we give the security definition of the DVC scheme and prove the security of HLE-DVC. We implement the experiments for HLE-DVC, evaluate its performance, and make its code publicly available: https://github.com/gaozheruiye/HLE-DVC. Using 256 machines to generate all proofs for a vector of length 2^{30} takes 17,515 seconds. This achieves a 256× speedup over HLE-DVC on a single machine, and is 142× faster than Hyperproofs [5]. The communication cost per machine is 0.768 KB.

1.2 Technical overview

The technical overview consists of four parts: §**(1)** introduces how HLE extends a vector into an HLE polynomial. §**(2)** explains how to generate a position proof with a full HLE polynomial. §**(3)** explores the generation of position proofs in a distributed setting. Notably, the position proofs in both §(2) and §(3) are polynomials. §**(4)** replaces the polynomials involved in §(2) and §(3) with their polynomial commitments, to propose HLE-DVC, which achieves the sublinear proof size. Figure 2 presents an example, where 4 machines generate position proofs for an vector of length 16 distributedly.

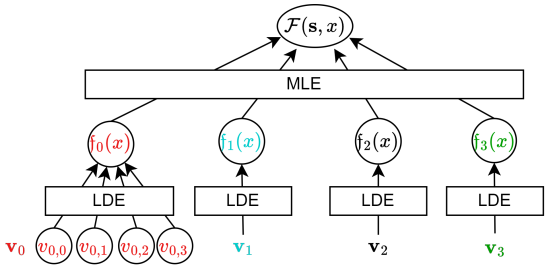

Figure 1: Hybrid Lagrange extension.

(1) Hybrid Lagrange Extension (HLE). Firstly, we propose a new extension method, HLE based on low degree extension (LDE) and multilinear extension (MLE). Figure 1 shows how HLE extends vector to a multivariate polynomial, where the vector $\mathbf{v}=(\mathbf{v}_0,\ldots,\mathbf{v}_3)$ consists of 4 subvectors,

each containing 4 elements, i.e., vector length $N = 16$, machine number $M = 4$ and the subvector length $n = N/M = 4$. The k-th subvector $\mathbf{v}_k$ is held by the k-th machine $\mathcal{P}_k$. First of all, for $k \in [0,4)$, each subvector $\mathbf{v}_k = (v_{k,0}, ..., v_{k,3})$ is extended to a univariate polynomial $\mathfrak{f}_k(x)$, where for all $i \in [0,n)$, $\mathfrak{f}_k(\omega_n^i) = v_{k,i}$, where ω_n is the n-th primitive root of unity in $\mathbb{F}$. We design an MLE-like method to aggregate polynomials $[\mathfrak{f}_k(x)]_{k\in[0,M)}$ to an HLE polynomial $\mathcal{F}(\mathbf{s},x)$: Define the k-th HLE sub-polynomial $\mathcal{F}_k(\mathbf{s},x)$ as

$$\mathcal{F}_k(\mathbf{s},x) = \mathsf{eq}(\mathbf{s}, \langle k \rangle)\mathfrak{f}_k(x)$$

where $\langle k \rangle \in \{0,1\}^\rho$ is the binary representation of k, (for simplicity, we set $\rho = \log M$. Herein, $\rho = 2$), $\mathbf{s} = (s_0, ..., s_{\rho-1})$ is a variate vector of length ρ and $\mathsf{eq}(\mathbf{s}, \langle k \rangle)$ is an identity function, which for $d \in [0,M)$, $\mathsf{eq}(\langle d \rangle, \langle k \rangle) = 1$ if $\langle d \rangle = \langle k \rangle$, and $\mathsf{eq}(\langle d \rangle, \langle k \rangle) = 0$ otherwise. Finally, the HLE polynomial $\mathcal{F}(\mathbf{s},x)$ is defined as the sum of all HLE sub-polynomials:

$$\mathcal{F}(\mathbf{s},x) = \sum_{k=0}^{M-1} \mathcal{F}_k(\mathbf{s},x)$$

For example, as shown in the "HLE" part of Figure 2, a vector $\mathbf{v} = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$ can be HLE-represented as $\mathcal{F}(\mathbf{s},x) = \mathsf{eq}(\mathbf{s}, \langle 0 \rangle)\mathfrak{f}_0(x) + \mathsf{eq}(\mathbf{s}, \langle 1 \rangle)\mathfrak{f}_1(x) + \mathsf{eq}(\mathbf{s}, \langle 2 \rangle)\mathfrak{f}_2(x) + \mathsf{eq}(\mathbf{s}, \langle 3 \rangle)\mathfrak{f}_3(x) = (1-s_0)(1-s_1)\mathfrak{f}_0(x) + (1-s_0)s_1\mathfrak{f}_1(x) + s_0(1-s_1)\mathfrak{f}_2(x) + s_0 s_1 \mathfrak{f}_3(x)$. The HLE polynomial satisfies that for all $d \in [0,M), i \in [0,n)$, $\mathcal{F}(\langle d \rangle, \omega_n^i) = \sum_{k=0}^{M-1} \mathcal{F}_k(\langle d \rangle, \omega_n^i) = \sum_{k=0}^{M-1} \mathsf{eq}(\langle d \rangle, \langle k \rangle)\mathfrak{f}_k(\omega_n^i) = \mathsf{eq}(\langle d \rangle, \langle d \rangle)\mathfrak{f}_k(\omega_n^i) = \mathfrak{f}_k(\omega_n^i) = v_{d,i}$.

(2) HLE decomposition and the position proofs generation. Next, assuming that a prover has obtained the **entire** vector $\mathbf{v}$ and thus a **complete** HLE $\mathcal{F}(\mathbf{s},x)$ of vector $\mathbf{v}$, given a fixed index $d \in [0,M)$ and $i \in [0,n)$, we will use Fig. 2 to explain how the prover generates a position proof $\Pi_{d,i}$ for $v_{d,i}$ (the i-th element in the subvector $\mathbf{v}_d$), and how to compute all position proofs $[\Pi_{d,i}]_{i\in[0,n)}$ for $[v_{d,i}]_{i\in[0,n)}$ (all elements in $\mathbf{v}_d$). In particular, herein, the position proof consists of polynomials directly. However, in the formal DVC construction in Section 5, the sublinear proof size can be obtained by replacing the polynomials with their PST commitments [28].

Generating a single position proof for an element. Proving that $v_{d,i} = v^*$ is equivalent to prove the HLE polynomial, $\mathcal{F}(\langle d \rangle, \omega_n^i) = v^*$. Fig. 2 shows an example, where $\mathbf{v} = (0,1,...,15)$, $N = 16$, $M = 4$, $\rho = \log M = 2$ and each subvector contains $n = 4$ elements. To prove that $\mathcal{F}(\langle 2 \rangle, \omega_4^3) = 11$, The **ultimate goal** of the prover is to find $\rho + 1 = 3$ polynomials $\{\mathfrak{q}_{2,0}(\mathbf{s},x), \mathfrak{q}_{2,1}(\mathbf{s},x)\mathfrak{q}_{2,2,3}(\mathbf{s},x)\}$, s.t.

$$\begin{aligned}\mathcal{F}(\mathbf{s},x) =& (s_0 - \langle 2 \rangle_0) \cdot \mathfrak{q}_{2,0}(\mathbf{s},x) + (s_1 - \langle 2 \rangle_1) \cdot \mathfrak{q}_{2,1}(\mathbf{s},x) + \\ & \mathfrak{q}_{2,2,3}(\mathbf{s},x)(x - \omega_4^3) + 11 \cdot \mathsf{eq}(\mathbf{s}, \langle 2 \rangle) \end{aligned} \quad (1)$$

where $\langle 2 \rangle_c$ denotes the c-th bit of $\langle 2 \rangle$. The soundness is straightforward and the formal proof is given in lemma 4.1.

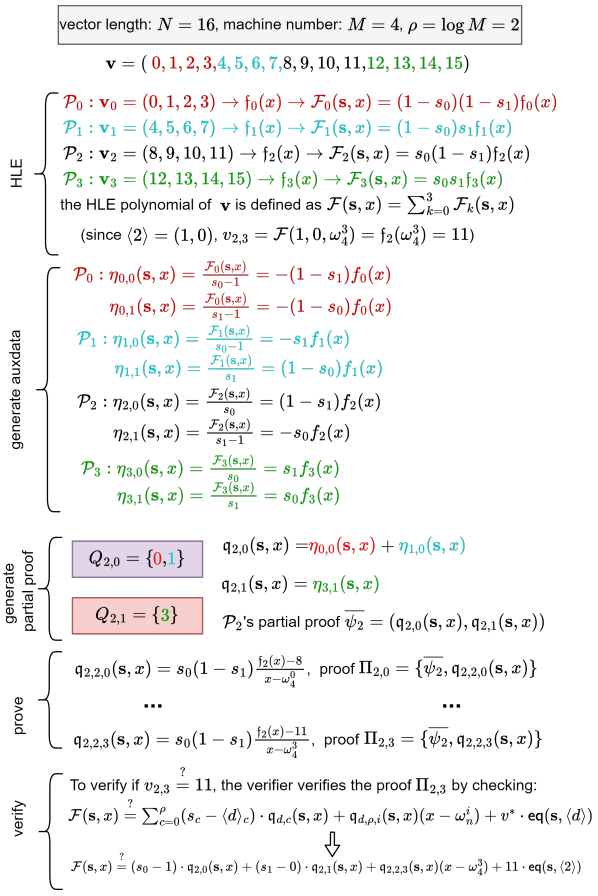

Figure 2: How 4 machines generate proofs for a vector of length 16 distributedly. $(\mathcal{P}_0, ..., \mathcal{P}_3)$ represent 4 machines.

As we previously assumed, the prover has access to the entire vector $\mathbf{v}$ and the HLE $\mathcal{F}(\mathbf{s},x)$, so that he can compute all these polynomials independently using polynomial divisions. Finally, the position proof $\Pi_{2,3}$ for $v_{2,3} = 11$ is set as these polynomials: $\{\mathfrak{q}_{2,0}(\mathbf{s},x), \mathfrak{q}_{2,1}(\mathbf{s},x)\mathfrak{q}_{2,2,3}(\mathbf{s},x)\}$. One can verify the proof $\Pi_{2,3}$ by checking if the equation (1) holds.

Generating all position proofs for all elements in a subvector. Trivially, the proofs $[\Pi_{2,i}]_{i\in[0,4)}$ for all elements $[v_{2,i}]_{i\in[0,4)}$ in $\mathbf{v}_2$ collectively contain $(\rho+1) \cdot n = 12$ polynomials. However, we found a special decomposition method to compute proofs, so that all proofs $[\Pi_{2,i}]_{i\in[0,4)}$ share the same first 2 polynomials $\mathfrak{q}_{2,0}(\mathbf{s},x)$ and $\mathfrak{q}_{2,1}(\mathbf{s},x)$. Consequently, all proofs $[\Pi_{2,i}]_{i\in[0,4)}$ involve only $2+4=6$ polynomials $\mathfrak{q}_{2,0}(\mathbf{s},x)$, $\mathfrak{q}_{2,1}(\mathbf{s},x)$ and $[\mathfrak{q}_{2,2,i}(\mathbf{s},x)]_{i\in[0,4)}$.

Firstly, we introduce a partition method (d-partition, definition 4.1), which partitions the index interval $[0,M)$ into ρ sets $[Q_{d,c}]_{c\in[0,\rho)}$ and $\{d\}$, s.t. $[0,M) = \{d\} \vee Q_{d,0} \vee ... \vee Q_{d,\rho-1}$. The sets $[Q_{d,c}]_{c\in[0,\rho)}$ has the property that for all $c \in [0,\rho)$ and $k \in Q_{d,c}$, $\langle k \rangle_c = 1 - \langle d \rangle_c$. For example $[0,4) = \{2\} \vee Q_{2,0} \vee Q_{2,1}$, where $Q_{2,0} = \{0,1\}$ and $Q_{2,1} = \{3\}$. Note that $\langle 0 \rangle_0 = \langle 1 \rangle_0 = 0 = 1 - \langle 2 \rangle_0$ and $\langle 3 \rangle_1 = 1 = 1 - \langle 2 \rangle_1$. Naturally,

the HLE polynomial $\mathcal{F}(\mathbf{s},x)$ can be decomposed as

$$\mathcal{F}(\mathbf{s},x)=\sum_{k=0}^{3}\mathcal{F}_k(\mathbf{s},x)=\sum_{c=0}^{1}\sum_{k\in Q_{2,c}}\mathcal{F}_k(\mathbf{s},x)+\mathcal{F}_2(\mathbf{s},x) \quad (2)$$

For all $c\in[0,2)$, since the properties of the set $Q_{2,c}$, there exists a polynomial $\mathfrak{q}_{2,c}(\mathbf{s},x)$, such that

$$\sum_{k\in Q_{2,c}}\mathcal{F}_k(\mathbf{s},x)=\mathfrak{q}_{2,c}(\mathbf{s},x)\cdot(s_c-\langle 2\rangle_c) \quad (3)$$

Moreover, for all $i\in[0,4)$, since $\mathfrak{f}_2(\omega_4^i)=v_{2,i}$, there exists a polynomial $\mathfrak{q}_{2,i}^*(x)=\frac{\mathfrak{f}_2(x)-v_{2,i}}{x-\omega_4^i}$, s.t. $\mathfrak{f}_2(x)=\mathfrak{q}_{2,i}^*(x)(x-\omega_4^i)+v_{2,i}$. We can set $\mathfrak{q}_{2,2,i}(\mathbf{s},x)=\mathsf{eq}(\mathbf{s},\langle 2\rangle)\mathfrak{q}_{2,i}^*(x)$, s.t.

$$\mathcal{F}_2(\mathbf{s},x)=\mathsf{eq}(\mathbf{s},\langle 2\rangle)\mathfrak{f}_2(x)=\mathfrak{q}_{2,2,i}(\mathbf{s},x)(x-\omega_4^i)+v_{2,i}\cdot\mathsf{eq}(\mathbf{s},\langle 2\rangle) \quad (4)$$

Finally, combining equations (2), (3) and (4), we find 6 polynomials $\mathfrak{q}_{2,0}(\mathbf{s},x)$, $\mathfrak{q}_{2,1}(\mathbf{s},x)$ and $[\mathfrak{q}_{2,2,i}(\mathbf{s},x)]_{i\in[0,4)}$, s.t., for all $i\in[0,n)$, $\mathcal{F}(\mathbf{s},x)=\sum_{c=0}^{1}(s_c-\langle 2\rangle_c)\cdot\mathfrak{q}_{2,c}(\mathbf{s},x)+\mathfrak{q}_{2,2,i}(\mathbf{s},x)(x-\omega_n^i)+v_{2,i}\cdot\mathsf{eq}(\mathbf{s},\langle 2\rangle)$. The position proofs $[\Pi_{2,i}]_{i\in[0,n)}$ for $[v_{2,i}]_{i\in[0,n)}$ consists of:

$$[\Pi_{2,i}]_{i\in[0,4)}=[\{\mathfrak{q}_{2,0}(\mathbf{s},x),\mathfrak{q}_{2,1}(\mathbf{s},x),\mathfrak{q}_{2,2,i}(\mathbf{s},x)\}]_{i\in[0,4)}$$

Note that all proofs for $[v_{2,i}]_{i\in[0,4)}$ share the same first 2 polynomials, and involve only 6 distinct polynomials. Since $[\mathfrak{q}_{2,c}(\mathbf{s},x)]_{c\in[0,2)}$ is reused in position proofs for every element in $\mathbf{v}_2$, we define $[\mathfrak{q}_{2,c}(\mathbf{s},x)]_{c\in[0,2)}$ as $\mathcal{P}_2$'s partial proof.

(3) Generate HLE and position proofs distributedly

In the distributed setting, each machine $\mathcal{P}_k$ only holds the subvector $\mathbf{v}_k$ and $\mathcal{F}_k(\mathbf{s},x)$. Therefore, directly generating position proofs through interactive methods would result in significant communication overhead. Therefore, we design a special method for generating the HLE polynomial and position proofs, which involves four algorithms. We continue to use Fig. 2 to illustrate the intuition.

1. **Generating an HLE polynomial**. Each machine $\mathcal{P}_k$ extends the subvector $\mathbf{v}_k$ to an HLE sub-polynomial $\mathcal{F}_k(\mathbf{s},x)$ as described in § (1). Then $\mathcal{P}_k$ sends $\mathcal{F}_k(\mathbf{s},x)$ to the master node, and master node set the HLE polynomial as $\mathcal{F}(\mathbf{s},x)=\sum_{k=0}^{M-1}\mathcal{F}_k(\mathbf{s},x)$. (As shown in "HLE" part of Figure 2)

2. **Generating auxiliary data (auxdata)**. Each machine $\mathcal{P}_k$ generates auxdata $\overline{\eta_k}$ using the HLE sub-polynomial $\mathcal{F}_k(\mathbf{s},x)$. For $c\in[0,\rho)$, $\mathcal{P}_k$ computes $\eta_{k,c}(\mathbf{s},x)=\frac{\mathcal{F}_k(\mathbf{s},x)}{s_c-(1-\langle k\rangle_c)}$.

Then $\mathcal{P}_k$ publicly outputs his auxdata $\overline{\eta_k}=[\eta_{k,c}(\mathbf{s},x)]_{c\in[0,\rho)}$. ("generate auxdata" part of Figure 2)

3. **Generating partial proofs**. Machines generate partial proofs using auxdata $[\overline{\eta_k}]_{k\in[0,M)}$. Take $\mathcal{P}_2$ as an example. For $c\in[0,\rho)$, $\mathcal{P}_2$ computes $\mathfrak{q}_{2,c}(\mathbf{s},x)=\sum_{k\in Q_{2,c}}\frac{\mathcal{F}_k(\mathbf{s},x)}{s_c-\langle 2\rangle_c}=\sum_{k\in Q_{2,c}}\frac{\mathcal{F}_k(\mathbf{s},x)}{s_c-(1-\langle k\rangle_c)}=\sum_{k\in Q_{2,c}}\eta_{k,c}(\mathbf{s},x)$.

Finally, $\mathcal{P}_2$ sets his partial proof as: $\overline{\psi_2}=[\mathfrak{q}_{2,c}(\mathbf{s},x)]_{c\in[0,\rho)}$. ("generate partial proof" part of Figure 2)

Polynomials in $\overline{\psi_2}$ satisfies the equation 3. Furthermore, we introduce a new data structure, HLE-Aux trees to help a master node to compute all partial proofs in $\mathrm{O}(M\log M)$ time. Then the master node sends the partial proof to each machine.

4. **Generating position proofs with partial proofs**. To prove that $v_{2,3}=11$, $\mathcal{P}_2$ computes $\mathfrak{q}_{2,2,3}(\mathbf{s},x)=\mathsf{eq}(\mathbf{s},x)\frac{\mathfrak{f}_2(x)-v_{2,3}}{x-\omega_4^3}$, and sets the position proof $\Pi_{2,3}=\{\overline{\psi_2},\mathfrak{q}_{2,2,3}(\mathbf{s},x)\}$ independently with his partial proof $\overline{\psi_2}$. ("prove" part of Figure 2).

(4) The HLE-DVC scheme. Based on the HLE and HLE decomposition, we design a DVC scheme, HLE-DVC.

To generate an HLE commitment to $\mathbf{v}$ distributedly, each machine $\mathcal{P}_k$ extends $\mathbf{v}_k$ to a HLE sub-polynomial $\mathcal{F}_k(\mathbf{s},x)$. Then $\mathcal{P}_k$ computes an HLE sub-commitment by computing the PST commitment to $\mathcal{F}_k(\mathbf{s},x)$: $C_{\mathbf{v}_k}=\mathsf{PST}(\mathcal{F}_k(\mathbf{s},x))$. Finally, the HLE commitment to $\mathbf{v}$ is set as: $C_{\mathbf{v}}=\sum_{k=0}^{M-1}C_{\mathbf{v}_k}$. Essentially, $C_{\mathbf{v}}$ is the PST commitment to $\mathcal{F}(\mathbf{s},x)$.

The algorithms for generating all position proofs are essentially the same as those described in § (1) (2) (3) of Section 1.2. The only difference is that the polynomials in the position proofs are replaced with their PST commitments.

Updating after a change to one element $v_{u,j}$ involves three phases: 1. Update the HLE commitment. 2. Position proofs corresponding to $\mathcal{P}_d$ ($d\neq u$) can be updated by only updating his partial proof. 3. Position proofs corresponding to $\mathcal{P}_u$ can be updated following the idea proposed in aSVC [13].

The algorithm for aggregating some position proofs can be categorized into two cases: 1. Position proofs for elements in the same subvector $\mathbf{v}_d$ can be aggregated using partial fraction decomposition [13]. 2. Position proofs for elements in different subvectors can be aggregated using partial fraction decomposition and inner product argument [29].

Efficiency analysis. Overall, HLE-DVC achieves high efficiency in both storage cost, communication cost and computation cost. Traditional single-machine VC schemes, such as aSVC based on KZG-PCS and Hyperproofs based on PST-PCS, require $\mathrm{O}(N)$ storage for the public parameters and the original vector, and $\mathrm{O}(N\log N)$ computation to generate all position proofs. In contrast, in HLE-DVC, each machine only stores the subvector of length $\mathrm{O}(N/M)$ and the proving key of size $\mathrm{O}(\frac{N\log M}{M})$. For proving cost, HLE-DVC reduces the total time for generating all proofs from $\mathrm{O}(N\log N)$ to $\mathrm{O}(\frac{N\log N}{M})$, which is an almost $M\times$ speedup.

The efficiency comes from the distributed design of storage and computation. The position proof for index (d,i) can be decomposed into two components: (1) Partial proof $\overline{\psi_d}$, which depends on the entire vector; (2) The subvector position proof $\pi_{d,\rho,i}$, which depends only on the subvector $\mathbf{v}_d$. During the proof generation phase, the machines first interactively compute partial proofs in $\mathrm{O}(\frac{N\log M}{M})$ time. After this step, any machine can **independently** generate proofs for the elements in its own subvector, with a cost of $\mathrm{O}(\frac{N\log(N/M)}{M})$

for all proofs. In sum, generating all proofs in the entire vector needs $O(\frac{N\log M}{M}) + O(\frac{N\log(N/M)}{M}) = O(\frac{N\log N}{M})$. The more detailed complexities analysis is presented in Section 5.7.

Throughout the proving phase, each machine only needs to use its own proving key, whose size is $O(N/M\log M)$, and the total communication cost of a single machine is only $O(\log M)$. Throughout the update phase, the machine that updates the vector can update all position proofs using its own update key of size $O(N/M\log M)$.

1.3 Applications of HLE-DVC

DVC has two major application scenarios: multiple machines parallel-accelerating proof generation and implementing conventional VC applications in distributed settings.

Parallel accelerating. Due to the distributed storage and computation capabilities of HLE-DVC, as described in Section 1.2, § (4), generating all proofs requires only $O(\frac{N\log N}{M})$, achieving an $M\times$ speedup over the single-machine version. It is suitable for single-center settings with heavy computational demands, such as large proof of reserve systems.

Distributed VC applications. Generally speaking, HLE-DVC enables almost all traditional VC applications to be deployed in a distributed environment. As a representative example, we introduce the distributed verifiable elementary database (DVEDB), where multiple servers distributedly hold an elementary database. DVEDB systems leverage the DVC algorithms to implement the same functionalities as that in VEDB. We provide the detailed constructions of DVEDB in Appendix A. In addition, DVC can also enable more advanced functionalities, such as stateless blockchain extension.

To support practical DVC applications, a DVC scheme needs to tackle two practical challenges: *1. Batch processing*. Batch processing requires a DVC scheme to support batch proving, batch verification, and proof aggregation, which significantly enhances its performance when handling large-scale workloads. Batch proofs and batch verification across multiple commitments can be natively achieved via additive homomorphism, leveraging the Schwartz-Zippel lemma. *2. Efficient updates*. When a single element changes, the system should quickly update the commitment all associated proofs. Updating all elements' proofs after a single element change should incur a time overhead less than $O(N)$. Moreover, when performing large-scale updates, additive homomorphism allows updates to committed data to be carried out directly on the commitments without needing to reveal the underlying values.

2 Related works

2.1 Vector commitment schemes

The definition of VC schemes was first formalized by Catalano and Fiore [7] in 2013. The two schemes most relevant to our scheme are aSVC [13] and Hyperproofs [3]. *aSVC* converts the vector $\mathbf{v}$ into a univariate polynomial using LDE. Then it relies on the KZG PCS to generate commitments and position proofs. The advantage of aSVC is its ability to achieve $O(1)$ proof size, $O(1)$ verification time, and $O(N\log N)$ time overhead for generating all proofs. However, it requires $O(N)$ time to update all position proofs after an element is changed. *Hyperproofs* uses MLE to extend the vector $\mathbf{v}$ into a multilinear polynomial and generates commitments and proofs based on the PST PCS. Hyperproofs only require $O(\log N)$ update time and $O(N\log N)$ time for generating all proofs, but incur $O(\log N)$ proof size and $O(\log N)$ verification time.

In 2023, BalanceProofs [27] introduced a general compiler that takes an aggregatable-only VC scheme as input and outputs a new VC scheme that is both aggregatable and maintainable. Moreover, BalanceProofs used this compiler to transform aSVC into an aggregatable and maintainable scheme, reducing the time complexity of update-all from $O(N)$ to $O(\sqrt{N}\log N)$. Our work can also leverage this approach to achieve maintainability. Merkle-based VC schemes [25, 30] achieve $O(\log N)$ proof size, $O(\log N)$ verification time. Its greatest advantage lies in the $O(N)$ time for generating all proofs and $O(\log N)$ update time. However, due to the inherent limitations of hash functions, they lack desirable properties such as batch proving, proof aggregation and homomorphism. The comparisons among related VC schemes are summarized in Appendix B.

2.2 Promising DVC schemes

To the best of our knowledge, there is no previous works on DVC schemes, and no work can trivially derive a satisfactory DVC scheme. Apart from the three naive DVC constructions mentioned in Section 1, some MLE-based schemes can be transformed into DVC schemes by applying the decomposition method introduced in Section 4. We present more construction details in Appendix C and briefly analyze their performance here. (These promising DVC constructions rely on Section 4 and are also originally proposed by us, and their complexities are summarized in Table 1.)

(1) *DVC scheme derived from Hyperproofs [3].* In Section 1, we discussed general homomorphic VC-cases. However, by leveraging tree-based VC schemes, many redundant computations can be avoided, thereby significantly reducing cost. Hyperproofs is a MLE-based homomorphic VC scheme based on the PST PCS. By combining it with the decomposition method proposed in Section 4, it can derive a DVC scheme, Hyperproofs-DVC, which achieves an $O(\log M)$ communication complexity and $O(\frac{N}{M}\log N)$ overhead for generating all proofs. However, it suffers from two drawbacks: The proof size is relatively large, $O(\log N)$; It does not support native aggregation and must rely on inner product arguments (IPA) for aggregation, which will significantly increases both the proving cost and the proof size.

(2) *DVC schemes derived from Campanelli et al. [26].* [26] proposed a general VC-construction aimed at achieving memory/time trade-offs in traditional VC schemes. By combining [26] with the decomposition method proposed in Section 4, it can also be transformed into a DVC. However, its proof additionally requires a low-degree test and a subvector commitment, and its verification involves checking three equations (whereas our HLE requires only one), which introduce extra costs. As a result, HLE-DVC enjoys advantages in proof size, verification cost, and aggregation performance in practice.

3 Preliminaries

Let N denote the length of the vector and M denote the number of machines. We assume that M and N are powers of 2 and divide $p-1$. Let $n = N/M$ and $\rho = \log M$. The k-th machine is denoted as $\mathcal{P}_k$ and the verifier is denoted as $\mathcal{V}$. We assume that $\mathcal{P}_0$ is the master node (in practice, the master node can be set to anyone). Let $\langle k \rangle = (\langle k \rangle_0, ..., \langle k \rangle_{\rho-1}) \in \{0,1\}^\rho$, denote the ρ-bit binary representation of k, and $\langle k \rangle_c$ denotes the c-th bit of $\langle k \rangle$. The notation with a bar (e.g., $\overline{\eta}$) represents a vector. We use $\mathbb{F}$ to denote a finite field (a prime field $\mathbb{F}_p$ for a large prime p) and λ to denote the security parameter. We assume that M and n are powers of 2 and divide $p-1$. Let ω_n be the n-th primitive root of unity in $\mathbb{F}$. The PST commitment to the polynomial $\mathcal{F}(\cdot)$ is denoted as $\mathsf{PST}(\mathcal{F}(\cdot))$. $[\mathbf{v}_k]_{k\in[0,M)}$ means $(\mathbf{v}_0, ..., \mathbf{v}_{M-1})$. $r \xleftarrow{\$} \mathbb{F}$ means r is randomly chosen from $\mathbb{F}$. "$\vee$" means the union of sets.

Lemma 3.1. ***(Univariate polynomial remainder lemma** [8, 31, 32]). A univariate polynomial $\mathfrak{f}(x^*) = v^*$, if and only if there exists a witness polynomial $\mathfrak{q}(x)$, s.t. $\mathfrak{f}(x) = \mathfrak{q}(x)(x - x^*) + v^*$.*

Definition 3.1. ***(Binary prefix tree).** As shown in Figure 3, a binary prefix tree is a binary tree, where the g-layer internal node of a tree $\mathtt{T}$ is denoted as $\mathtt{T.Node}_{b_0...b_g}$, where $b_0, ..., b_g \in \{0,1\}$ (the layer below the root node is defined as layer 0). The left child of $\mathtt{T.Node}_{b_0...b_g}$ is denoted as $\mathtt{T.Node}_{b_0...b_g 0}$, and the right child as $\mathtt{T.Node}_{b_0...b_g 1}$. For a binary prefix tree with M leaf nodes, the k-th leaf node is $\mathtt{T.Node}_{\langle k \rangle}$.*

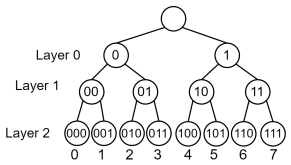

Figure 3: A binary prefix tree $\mathtt{T}$. The k-th leaf node of $\mathtt{T}$ is denoted as $\mathtt{T.Node}_{\langle k \rangle}$.

• **Bilinear pairing.** $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic multiplicative groups of prime order p. P is a generator of $\mathbb{G}_1$ and Q is a generator of $\mathbb{G}_2$. The pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear pairing if it satisfies the bilinearity, non-degeneracy and computability. Bilinearity requires that $\forall (U, V, a, b) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{F} \times \mathbb{F} : e(aU, bV) = e(U, V)^{ab}$. A pairing instance is denoted as $\mathsf{bp} = \mathsf{BilGen}(1^\lambda) = (p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, Q)$.

• **PST polynomial commitment scheme (PCS) for multivariate polynomials. [28] [8]** A PCS for multivariate polynomials is a tuple of four protocols $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Prove}, \mathsf{Verify})$. The detailed descriptions of the PST PCS's algorithms are presented in Appendix D.1. The PST PCS used in this paper is a non-hiding version.

• **Inner Product Arguments (IPA).** Bünz et al. [29] give a non-interactive IPA for the inner pairing products relation. The detailed definition of the IPA protocol is presented in Appendix D.2. **IPA complexity.** IPA.Prove takes $O(\rho)$ time, IPA.Verify takes $O(\log \rho)$ time and the proof size is $|\pi_{IPA}| = O(\log \rho)$.

• **Low degree extension (LDE).** A vector $\mathbf{v} = [v_0, ..., v_{n-1}]$ can be represented by a polynomial $\mathfrak{f}(x)$ using LDE: $\mathfrak{f}(x) = \sum_{i=0}^{n-1} \mathcal{L}_i(x) v_i$, where $\mathcal{L}_i(x) = \prod_{j\in[0,M), j\neq i} \frac{x-\omega_n^j}{\omega_n^i - \omega_n^j}$. $\mathcal{L}_i(x)$ has a property that $\mathcal{L}_i(\omega_n^i) = 1$ and $\mathcal{L}_i(\omega_n^j) = 0$, if $j \in [0,M), i \neq j$. Thus for all $i \in [0,n)$, $\mathfrak{f}(\omega_n^i) = v_i$.

• **Identity function.** Let $\mathsf{eq} : \{0,1\}^\rho \times \{0,1\}^\rho \to \{0,1\}$ be the identity function, which can be expressed as:

$$\mathsf{eq}(\mathbf{s}, \mathbf{y}) = \prod_{c=0}^{\rho-1} ((1-s_c)(1-y_c) + s_c y_c) \tag{5}$$

such that $\mathsf{eq}(\mathbf{s}, \mathbf{y}) = 1$ if $\mathbf{s} = \mathbf{y}$, and $\mathsf{eq}(\mathbf{s}, \mathbf{y}) = 0$, otherwise.

• **Multilinear extension (MLE).** Let $M = 2^\rho$, where ρ is an integer. A vector $\mathbf{v} = [v_0, ..., v_{M-1}]$ can be represented by a multivariate polynomial $\mathcal{F} : \mathbb{F}^\rho \to \mathbb{F}$: $\mathcal{F}(\mathbf{s}) = \sum_{k=0}^{M-1} \mathsf{eq}(\mathbf{s}, \langle k \rangle) v_k$, s.t. for any $d \in [0,M)$, $\mathcal{F}(\langle d \rangle) = \sum_{k=0}^{M-1} \mathsf{eq}(\langle d \rangle, \langle k \rangle) v_k = v_d$.

4 Core techniques

This section introduces the core techniques of this paper. In this section, the position proof consists of polynomials directly. However, in the formal DVC construction in section 5, the sublinear proof size can be achieved by replacing the polynomials with their polynomial commitments and polynomial evaluation proofs.

• Section 4.1 presents the definition of HLE based on LDE and MLE. HLE is used to extend a vector $\mathbf{v} = (\mathbf{v}_0, \ldots, \mathbf{v}_{M-1})$ to an HLE polynomial $\mathcal{F}(\mathbf{s}, x)$.

• Section 4.2 firstly introduces a HLE polynomial remainder lemma 4.1, showing that a single position proof for any element $v_{d,i} = v^*$ ($d \in [0,M), i \in [0,n)$) consists of $\rho + 1$ witness polynomials ($\rho = \log M$). Secondly, it introduces an HLE decomposition lemma 4.3, showing that the n proofs for all

elements in the subvector $\mathbf{v}_d$ only involve $n+\rho$ distinct polynomials. We show how to compute these proofs when a single machine holds the complete HLE polynomial $\mathcal{F}(\mathbf{s},x)$.

• Section 4.3 discusses how to compute position proofs interactively in a distributed setting.

4.1 Hybrid Lagrange Extension

This technique originates from combining the ideas of MLE and LDE. For each subvector held by a single machine, we use the LDE to extend the subvector to a univariate polynomial. At the level of distributed machines, we employ the MLE to aggregate all univariate polynomials into an HLE polynomial.

HLE Sub-Polynomial. The HLE sub-polynomial $\mathcal{F}_k(\mathbf{s},x)$ is derived from the subvector $\mathbf{v}_k \in \mathbb{F}^n$ of length n held by machine $\mathcal{P}_k$. $\mathcal{P}_k$ firstly extends $\mathbf{v}_k$ to a univariate polynomial $\mathfrak{f}_k(x)$ of degree less than n using LDE, s.t. for all $i \in [0,n)$, $\mathfrak{f}_k(\omega_n^i) = v_{k,i}$. As shown in Figure 1 and 2, the HLE sub-polynomial held by $\mathcal{P}_k$ is defined as($\mathbf{s}$ is a variables vector of length $\rho = \log M$):

$$\mathcal{F}_k(\mathbf{s},x) = \mathsf{eq}(\mathbf{s},\langle k\rangle)\cdot\mathfrak{f}_k(x)$$

which satisfies the following conditions for all $d \in [0,M)$ and $i \in [0,n)$:

If $d = k$, $\mathcal{F}_k(\langle d\rangle,\omega_n^i) = \mathsf{eq}(\langle k\rangle,\langle k\rangle)\cdot\mathfrak{f}_k(\omega_n^i) = v_{k,i} = v_{d,i}$.

if $d \neq k$, $\mathcal{F}_k(\langle d\rangle,\omega_n^i) = \mathsf{eq}(\langle d\rangle,\langle k\rangle)\cdot\mathfrak{f}_k(\omega_n^i) = 0$.

HLE Polynomial. The HLE polynomial $\mathcal{F}(\mathbf{s},x)$ is extended from the vector $\mathbf{v} = (\mathbf{v}_0,\ldots,\mathbf{v}_{M-1})$ and is defined as $\mathcal{F}(\mathbf{s},x) = \sum_{k=0}^{M-1}\mathcal{F}_k(\mathbf{s},x) = \sum_{k=0}^{M-1}\mathsf{eq}(\mathbf{s},\langle k\rangle)\cdot\mathfrak{f}_k(x)$, which is the sum of all the HLE sub-polynomials. For example, a vector $\mathbf{v} = [\mathbf{v}_0,\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3]$ can be HLE-represented as $\mathcal{F}(\mathbf{s},x) = (1-s_0)(1-s_1)\mathfrak{f}_0(x) + (1-s_0)s_1\mathfrak{f}_1(x) + s_0(1-s_1)\mathfrak{f}_2(x) + s_0s_1\mathfrak{f}_3(x)$. It satisfies for all $d \in [0,M), i \in [0,n)$, $\mathcal{F}(\langle d\rangle,\omega_n^i) = v_{d,i}$.

4.2 HLE decomposition

Lemma 4.1. ***(HLE polynomial remainder lemma)***. *$\mathcal{F}(\mathbf{s},x)$ is the HLE of $\mathbf{v} = (\mathbf{v}_0,\ldots,\mathbf{v}_{M-1})$. Let $\rho = \log M$. Given $d \in [0,M)$, $i \in [0,n)$, $\mathcal{F}(\langle d\rangle,\omega_n^i) = v^*$, if and only if there exist $\rho+1$ polynomials, $\{[\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c\in[0,\rho)},\mathfrak{q}_{d,\rho,i}(\mathbf{s},x)\}$, s.t.*

$$\mathcal{F}(\mathbf{s},x) = \sum_{c=0}^{\rho-1}(s_c - \langle d\rangle_c)\cdot\mathfrak{q}_{d,c}(\mathbf{s},x) + \mathfrak{q}_{d,\rho,i}(\mathbf{s},x)(x-\omega_n^i) + v^*\cdot\mathsf{eq}(\mathbf{s},\langle d\rangle) \tag{6}$$

PROOF: $\Leftarrow$ Proof of necessity: if there exist $\rho+1$ polynomials, $\{[\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c\in[0,\rho)},\mathfrak{q}_{d,\rho,i}(\mathbf{s},x)\}$ satisting the equation 6, then, $\mathcal{F}(\langle d\rangle,\omega_n^i) = \sum_{c=0}^{\rho-1}(\langle d\rangle_c - \langle d\rangle_c)\cdot\mathfrak{q}_{d,c}(\langle d\rangle_c,\omega_n^i) + \mathfrak{q}_{d,\rho,i}(\langle d\rangle_c,\omega_n^i)(\omega_n^i - \omega_n^i) + v^*\cdot\mathsf{eq}(\langle d\rangle,\langle d\rangle) = v^*$. Thus $\mathcal{F}(\langle d\rangle,\omega_n^i) = v^*$.

$\Rightarrow$ Proof of sufficiency: these polynomials can be obtained through successive polynomial divisions: first divide $(\mathcal{F}(\mathbf{s},x) - v^*\cdot\mathsf{eq}(\mathbf{s},\langle d\rangle))$ by $(x-s_0)$, obtaining the quotient polynomial $\mathfrak{q}_{d,0}(\mathbf{s},x)$, then divide the resulting remainder by $(x-s_1)$, obtaining the quotient polynomial $\mathfrak{q}_{d,1}(\mathbf{s},x)$, and continue this process until the final two division by $(x-s_{\rho-1})$ and $(x-\omega_n^i)$ to obtain $\mathfrak{q}_{d,\rho-1}(\mathbf{s},x)$ and $\mathfrak{q}_{d,\rho,i}(\mathbf{s},x)$. By the multivariate polynomial remainder lemma E.1, polynomials $\{[\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c\in[0,\rho)},\mathfrak{q}_{d,\rho,i}(\mathbf{s},x)\}$ always satisfy the equation 6. □

Via lemma 4.1, one can prove $v_{d,i} = \mathcal{F}(\langle d\rangle,\omega_n^i) = v^*$ by showing the valid $\rho+1$ witness polynomials. Thus we define the position proof $\Pi_{d,i}$ for $v_{d,i}$ as $\Pi_{d,i} = \{[\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c\in[0,\rho)},\mathfrak{q}_{d,\rho,i}(\mathbf{s},x)\}$. The verifier can verify the proof by checking if $\mathcal{F}(\mathbf{s},x) \stackrel{?}{=} \sum_{c=0}^{\rho-1}(s_c - \langle d\rangle_c)\cdot\mathfrak{q}_{d,c}(\mathbf{s},x) + \mathfrak{q}_{d,\rho,i}(\mathbf{s},x)(x-\omega_n^i) + v^*\cdot\mathsf{eq}(\mathbf{s},\langle d\rangle)$.

Trivially, proofs $[\Pi_{d,i}]_{i\in[0,n)}$ for all elements $[v_{d,i}]_{i\in[0,n)}$ collectively contain $(\rho+1)\cdot n$ polynomials. However, we found a special method for computing proofs, so that all proofs $[\Pi_{d,i}]_{i\in[0,n)}$ share the same first ρ polynomials $[\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c\in[0,\rho)}$. Consequently, all proofs $[\Pi_{d,i}]_{i\in[0,n)}$ involves only $n+\rho$ distinct polynomials $[\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c\in[0,\rho)}$ and $[\mathfrak{q}_{d,\rho,i}(\mathbf{s},x)]_{i\in[0,n)}$. This method is presented in lemma 4.3.

To better illustrate lemma 4.3, we introduce a special d-partition method (definition 4.1) and a partition division lemma 4.2.

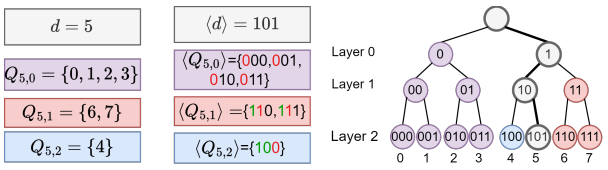

Figure 4: d-partition when $M = 8, \rho = \log M = 3$ and $d = 5$. 5-partition partitions $[0,8) = \{5\} \vee Q_{5,0} \vee Q_{5,1} \vee Q_{5,2}$. The second column of this figure shows the binary decompositions of d and the corresponding sets $[Q_{5,c}]_{c\in[0,3)}$, where the green color string represents the longest prefix which is the same as $\langle 5\rangle = 101$, while the red color bit represents the first bit which is not the same as $\langle 5\rangle$ (e.g., $4 = 100$). Taking $Q_{5,1}$ as an example, the set $Q_{5,1}$ is a set of all $k \in [0,8)$, satisfying that the first 1 bit of $\langle k\rangle$ is the same as that of $\langle 5\rangle$ (i.e. $\langle k\rangle_0 = \langle 5\rangle_0 = 1$), while $\langle k\rangle_1 \neq \langle 5\rangle_1$ (i.e. $\langle k\rangle_1 = 1 - \langle 5\rangle_1 = 1-0 = 1$). From another perspective, given a binary prefix tree $\mathtt{T}$ of 8 leaf nodes, where the k-th leaf node is $\mathtt{T.Node}_{\langle k\rangle}$, $Q_{5,1} = \{6,7\}$ is the set of all leaf nodes' indices of the sub-tree with the root node $\mathtt{T.Node}_{11}$.

Definition 4.1. ***(d-partition for $[0,M)$)***. *Given a fixed $d \in [0,M)$, the interval $[0,M)$ can be partitioned into ρ sets $[Q_{d,c}]_{c\in[0,\rho)}$ and $\{d\}$, s.t. $[0,M) = \{d\} \vee Q_{d,0} \vee \ldots \vee Q_{d,\rho-1}$, where the set $Q_{d,c}$ contains all $k \in [0,M)$, s.t. the first c bits of $\langle k\rangle$ are the same as that of $\langle d\rangle$, while $\langle k\rangle_c \neq \langle d\rangle_c$. Formally, **$d$-partition is defined as** $[Q_{d,c}]_{c\in[0,\rho)}$ as follows: for a fixed*

$d \in [0,M)$, *set* $Q_{d,0} = \{k | \langle k \rangle_0 \neq \langle d \rangle_0\}$, *and for* $c \in [1,\rho)$, *the set* $Q_{d,c} = \{k | k \in [0,M), \langle k \rangle_c \neq \langle d \rangle_c, \langle k \rangle_0 = \langle d \rangle_0, ..., \langle k \rangle_{c-1} = \langle d \rangle_{c-1}\}$.

From another perspective, given a binary prefix tree $\mathtt{T}$ (definition 3.1) with M leaf nodes, $Q_{d,c}$ is the set of all leaf nodes indices of the sub-tree with the root node $\mathtt{T.Node}_{b_0...b_c}$, where $b_0 = \langle d \rangle_0, ..., b_{c-1} = \langle d \rangle_{c-1}$ and $b_c = 1 - \langle d \rangle_0$. Take $M = 8, d = 5$ as an example, as shown in Figure 4, the path corresponding to the 5-leaf node $\mathtt{T.Node}_{101}$ is $(\mathtt{T.Node}_1, \mathtt{T.Node}_{10}, \mathtt{T.Node}_{101})$. Correspondingly, $Q_{5,0} = \{0,1,2,3\}$ denotes all leaf nodes under the sibling of $\mathtt{T.Node}_1$, and $Q_{5,1} = \{6,7\}$ denotes all leaf nodes under the sibling of $\mathtt{T.Node}_{10}$. Similarly, $Q_{5,2} = \{4\}$ is the sibling of $\mathtt{T.Node}_{101}$.

Lemma 4.2. ***(Partition division).*** *Given a fixed* $d \in [0,M)$, *and the corresponding d-partition* $[Q_{d,c}]_{c \in [0,\rho)}$, *for all* $c \in [0,\rho)$, *there exists a polynomial* $\mathfrak{q}_{d,c}(\mathbf{s},x)$, *such that*

$$\sum_{k \in Q_{d,c}} \mathcal{F}_k(\mathbf{s},x) = \mathfrak{q}_{d,c}(\mathbf{s},x) \cdot (s_c - \langle d \rangle_c)$$

PROOF: Recall that for all $k \in Q_{d,c}$, $\langle k \rangle_c \neq \langle d \rangle_c$ and $\langle k \rangle_c \in \{0,1\}, \langle d \rangle_c \in \{0,1\}$, thus $\langle k \rangle_c = 1 - \langle d \rangle_c$. Then we have:

$$\begin{aligned}(1-s_c)(1-\langle k \rangle_c) + s_c \langle k \rangle_c &= (1-s_c)\langle d \rangle_c + s_c(1-\langle d \rangle_c) \\ &= \langle d \rangle_c + s_c - 2s_c\langle d \rangle_c\end{aligned}$$

So if $\langle d \rangle_c = 0$, then $(1-s_c)(1-\langle k \rangle_c) + s_c\langle k \rangle_c = s_c = s_c - \langle d \rangle_c$. If $\langle d \rangle_c = 1$, then $(1-s_c)(1-\langle k \rangle_c) + s_c\langle k \rangle_c = 1 - s_c = -(s_c - \langle d \rangle_c)$.

Then $\frac{(1-s_c)(1-\langle k \rangle_c)+s_c}{s_c - \langle d \rangle_c} = \begin{cases} 1 & \text{if } \langle d \rangle_c = 0, \langle k \rangle_c = 1 \\ -1 & \text{if } \langle d \rangle_c = 1, \langle k \rangle_c = 0 \end{cases}$.

So for all $k \in Q_{d,c}$, $(s_c - \langle d \rangle_c)$ divides $(1-s_c)(1-\langle k \rangle_c) + s_c\langle k \rangle_c$. Furthermore, considering the equation (5) and $\mathcal{F}_k(\mathbf{s},x) = \mathsf{eq}(\mathbf{s}, \langle k \rangle) \cdot \mathfrak{f}_k(x)$, so $(s_c - \langle d \rangle_c)$ can divide $\mathcal{F}_k(\mathbf{s},x)$ for all $k \in Q_{d,c}$. Thus we can set a polynomial $\mathfrak{q}_{d,c}(\mathbf{s},x) = \sum_{k \in Q_{d,c}} \frac{\mathcal{F}_k(\mathbf{s},x)}{s_c - \langle d \rangle_c}$, which satisfies that $\sum_{k \in Q_{d,c}} \mathcal{F}_k(\mathbf{s},x) = \mathfrak{q}_{d,c}(\mathbf{s},x) \cdot (s_c - \langle d \rangle_c)$. □

Lemma 4.3. ***(HLE decomposition lemma).*** $\mathcal{F}(\mathbf{s},x)$ *is the HLE of* $\mathbf{v} = (\mathbf{v}_0, ..., \mathbf{v}_{M-1})$. *Given a fixed* $d \in [0,M)$, *and the corresponding d-partition* $[Q_{d,c}]_{c \in [0,\rho)}$, *one can compute* $\rho + n$ *polynomials* $[\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c \in [0,\rho)}$ *and* $[\mathfrak{q}_{d,\rho,i}(\mathbf{s},x)]_{i \in [0,n)}$ *by:*

$$[\mathfrak{q}_{d,c}(\mathbf{s},x) = \sum_{k \in Q_{d,c}} \frac{\mathcal{F}_k(\mathbf{s},x)}{s_c - \langle d \rangle_c}]_{c \in [0,\rho)}$$

$$[\mathfrak{q}_{d,\rho,i}(\mathbf{s},x) = \mathsf{eq}(\mathbf{s}, \langle d \rangle) \frac{\mathfrak{f}_d(x) - v_{d,i}}{x - \omega_n^i}]_{i \in [0,n)}$$

$[\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c \in [0,\rho)}$ *and* $[\mathfrak{q}_{d,\rho,i}(\mathbf{s},x)]_{i \in [0,n)}$ *satisfy that for all* $i \in [0,n)$, $\Pi_{d,i} = \{[\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c \in [0,\rho)}, \mathfrak{q}_{d,\rho,i}(\mathbf{s},x)\}$ *is a valid proof for* $v_{d,i}$, *namely:* $\mathcal{F}(\mathbf{s},x) = \sum_{c=0}^{\rho-1}(s_c - \langle d \rangle_c) \cdot \mathfrak{q}_{d,c}(\mathbf{s},x) + \mathfrak{q}_{d,\rho,i}(\mathbf{s},x)(x - \omega_n^i) + v_{d,i} \cdot \mathsf{eq}(\mathbf{s}, \langle d \rangle)$.

PROOF: Since $[0,M) = \{d\} \vee Q_{d,0} \vee ... \vee Q_{d,\rho-1}$, we have:

$$\mathcal{F}(\mathbf{s},x) = \sum_{k=0}^{M-1} \mathcal{F}_k(\mathbf{s},x) = \sum_{c=0}^{\rho-1} \sum_{k \in Q_{d,c}} \mathcal{F}_k(\mathbf{s},x) + \mathcal{F}_d(\mathbf{s},x) \quad (7)$$

Moreover, lemma 4.2 shows that for all $c \in [0,\rho)$ and the polynomial $\mathfrak{q}_{d,c}(\mathbf{s},x) = \sum_{k \in Q_{d,c}} \frac{\mathcal{F}_k(\mathbf{s},x)}{s_c - \langle d \rangle_c}$,

$$\sum_{k \in Q_{d,c}} \mathcal{F}_k(\mathbf{s},x) = (s_c - \langle d \rangle_c)\mathfrak{q}_{d,c}(\mathbf{s},x) \quad (8)$$

Then considering that for $i \in [0,n)$, $\mathfrak{f}_d(\omega_n^i) = v_{d,i}$, following lemma 3.1, there exists a polynomial $\mathfrak{q}_{d,i}^*(x) = \frac{\mathfrak{f}_d(x) - v_{d,i}}{x - \omega_n^i}$, s.t. $\mathfrak{f}_d(x) = \mathfrak{q}_{d,i}^*(x)(x - \omega_n^i) + v_{d,i}$. Let $\mathfrak{q}_{d,\rho,i}(\mathbf{s},x) = \mathsf{eq}(\mathbf{s}, \langle d \rangle)\mathfrak{q}_{d,i}^*(x)$, and we have:

$$\mathcal{F}_d(\mathbf{s},x) = \mathsf{eq}(\mathbf{s}, \langle d \rangle)\mathfrak{f}_d(x) = \mathfrak{q}_{d,\rho,i}(\mathbf{s},x)(x - \omega_n^i) + v_{d,i} \cdot \mathsf{eq}(\mathbf{s}, \langle d \rangle) \quad (9)$$

Finally, combining equations (7), (8) and (9), we have found polynomials $[\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c \in [0,\rho)}$ and $[\mathfrak{q}_{d,\rho,i}(\mathbf{s},x)]_{i \in [0,n)}$, s.t., for all $i \in [0,n)$, $\mathcal{F}(\mathbf{s},x) = \sum_{c=0}^{\rho-1}(s_c - \langle d \rangle_c) \cdot \mathfrak{q}_{d,c}(\mathbf{s},x) + \mathfrak{q}_{d,\rho,i}(\mathbf{s},x)(x - \omega_n^i) + v_{d,i} \cdot \mathsf{eq}(\mathbf{s}, \langle d \rangle)$. □

Since $[\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c \in [0,\rho)}$ is reused in position proofs for every element in $\mathbf{v}_d$, we set $[\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c \in [0,\rho)}$ as $\mathcal{P}_d$'s partial proof.

4.3 Generating position proofs distributedly

In the distributed setting, each machine $\mathcal{P}_k$ only holds $\mathbf{v}_k$ and its HLE sub-polynomial $\mathcal{F}_k(\mathbf{s},x)$. Therefore, we design a special method to generate proofs distributedly.

To reduce communication during the proving process, we divide the proof generation process into three algorithms:

1. **Generating auxdata**. Each machine $\mathcal{P}_k$ generates auxdata $\overline{\eta_k}$ using the HLE sub-polynomial $\mathcal{F}_k(\mathbf{s},x)$.

2. **Generating partial proofs**. Each machine $\mathcal{P}_d$ can compute his partial proof $\overline{\psi_d}$ using all auxdata $[\overline{\eta_k}]_{k \in [0,M)}$ in $O(M)$ time. Furthermore, we present an algorithm for **Generating all partial proofs** to generate all machines' partial proofs in $O(M \log M)$ time.

3. **Generating all position proofs**. With the partial proof $\overline{\psi_d}$, $\mathcal{P}_d$ independently compute proofs for all elements in the subvector $\mathbf{v}_d$.

4.3.1 Generating auxdata

For $c \in [0,\rho)$, $\mathcal{P}_k$ computes:

$$\eta_{k,c}(\mathbf{s},x) = \frac{\mathcal{F}_k(\mathbf{s},x)}{s_c - (1 - \langle k \rangle_c)}$$

Then $\mathcal{P}_k$ publicly outputs his auxdata $\overline{\eta_k} = [\eta_{k,c}(\mathbf{s},x)]_{c \in [0,\rho)}$. (Per machine computational cost: $O(\log M)$ polynomials; communication cost: $O(\log M)$ polynomials)

4.3.2 Generating partial proofs

Generating one partial proof. For $c \in [0,\rho)$, $\mathcal{P}_d$ computes

$$\mathfrak{q}_{d,c}(\mathbf{s},x) = \sum_{k\in Q_{d,c}} \frac{\mathcal{F}_k(\mathbf{s},x)}{s_c - \langle d\rangle_c} = \sum_{k\in Q_{d,c}} \frac{\mathcal{F}_k(\mathbf{s},x)}{s_c - (1-\langle k\rangle_c)} = \sum_{k\in Q_{d,c}} \eta_{k,c}(\mathbf{s},x)$$

Finally, $\mathcal{P}_d$ sets his partial proof as: $\overline{\Psi_d} = [\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c\in[0,\rho)}$. (Computational cost: $O(M)$; communication cost: $O(M)$)

Generating all partial proofs simultaneously. Trivially, running the partial proof generation algorithm M times can produce all partial proofs $[\overline{\Psi_d}]_{d\in[0,M)}$ in $O(M^2)$ time. However, we propose a more efficient approach where a master node can generate all partial proofs in $O(M\log M)$ time. An interesting fact is that if the first $(c+1)$ bits of $\langle d\rangle$ and $\langle d'\rangle$ are identical, then $Q_{d,c} = \{k | k\in[0,M), \langle k\rangle_0 = \langle d\rangle_0, ..., \langle k\rangle_{c-1} = \langle d\rangle_{c-1}, \langle k\rangle_c \neq \langle d\rangle_c\} = Q_{d',c}$. Consequently, $\mathfrak{q}_{d,c}(\mathbf{s},x) = \sum_{k\in Q_{d,c}} \eta_{k,c}(\mathbf{s},x) = \sum_{k\in Q_{d',c}} \eta_{k,c}(\mathbf{s},x) = \mathfrak{q}_{d',c}(\mathbf{s},x)$. This implies that a lot of computation can be avoided. To utilize this repetitive property to reduce computational overhead, we introduce the HLE-Aux trees in definition 4.2.

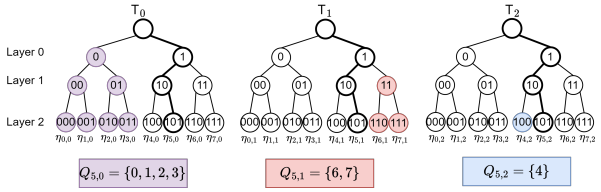

Figure 5: HLE-Aux trees $\{\mathtt{T}_0, \mathtt{T}_1, \mathtt{T}_2\}$ (let $M=8$) used to generate partial proofs (for simplicity, the variables $(\mathbf{s},x)$ are omitted). The k-th leaf node of $\mathtt{T}_c$ stores $\eta_{k,c}$. Each non-leaf node is the sum of its two child nodes. For $d=5$,$\langle d\rangle = 101$: $\mathfrak{q}_{5,0} = \sum_{k\in Q_{5,0}} \eta_{k,0} = \eta_{0,0}+\eta_{1,0}+\eta_{2,0}+\eta_{3,0} = \mathtt{T}_c.\mathtt{Node}_{0,0}$, and $\mathfrak{q}_{5,1} = \sum_{k\in Q_{5,1}} \eta_{k,1} = \eta_{6,1}+\eta_{7,1} = \mathtt{T}_c.\mathtt{Node}_{1,11}$, and $\mathfrak{q}_{5,2} = \sum_{k\in Q_{5,2}} \eta_{k,2} = \eta_{4,2} = \mathtt{T}_c.\mathtt{Node}_{2,100}$. $\mathcal{P}_5$'s partial proof is set as $\overline{\Psi_5} = (\mathfrak{q}_{5,0}, \mathfrak{q}_{5,1}, \mathfrak{q}_{5,2})$.

Definition 4.2. ***HLE-Aux trees.*** *As shown in Figure 5, HLE-Aux trees $[\mathtt{T}_c]_{c\in[0,\rho)}$ are ρ trees that help to compute all the partial proofs within $O(M\log M)$ overhead. Each tree $\mathtt{T}_c$ is a binary prefix tree (definition 3.1). The internal node g of the $\mathtt{T}_c$ layer is denoted as $\mathtt{T}_c.\mathtt{Node}_{b_0\ldots b_g}$. Specifically, we assign each node to store a value: The value of the k-th leaf node of $\mathtt{T}_c$ is set as $\mathtt{T}_c.\mathtt{Node}_{\langle k\rangle} = \eta_{k,c}$. The value of an internal node is set as the sum of its two child nodes, i.e. $\mathtt{T}_c.\mathtt{Node}_{b_0\ldots b_g} = \mathtt{T}_c.\mathtt{Node}_{b_0\ldots b_g 0} + \mathtt{T}_c.\mathtt{Node}_{b_0\ldots b_g 1}$.*

Given the HLE-Aux trees $[\mathtt{T}_c]_{c\in[0,\rho)}$, recall that for all $d\in[0,M), c\in[0,\rho)$, $Q_{d,c}$ is the set of all leaf nodes' indices of the sub-tree with the root node $\mathtt{T}_c.\mathtt{Node}_{b_0\ldots b_c}$, where $b_0 = \langle d\rangle_0, ..., b_{c-1} = \langle d\rangle_{c-1}$ and $b_c = 1-\langle d\rangle_c$. So $\mathfrak{q}_{d,c} = \sum_{k\in Q_{d,c}} \eta_{k,c} = \mathtt{T}_c.\mathtt{Node}_{b_0\ldots b_c}$. Finally, $\mathcal{P}_d$'s partial proof is set as $\overline{\Psi_d} = [\mathfrak{q}_{d,c}(\mathbf{s},x)]_{c\in[0,\rho)}$.

As a result, the master node can first generate ρ HLE-Aux trees with a complexity of $O(M\log M)$ and then derive all partial proofs simultaneously based on the values of the tree nodes. Figure 5 shows an example for generating $\mathcal{P}_5$'s partial proof when $M=8$. The computational cost for master node is $O(M\log M)$, and the communication complexity for each machine is $O(\log M)$.

Generating all position proofs. To prove each element $v_{d,i} = v^*$, $\mathcal{P}_d$ computes $\mathfrak{q}^*_{d,i}(x) = \frac{\mathfrak{f}_d(x)-v^*}{x-\omega_n^i}$, $\mathfrak{q}_{d,\rho,i}(\mathbf{s},x) = \mathsf{eq}(\mathbf{s},\langle d\rangle)\mathfrak{q}^*_{d,i}(x)$. The position proof for $v_{d,i}$ is set as $\Pi_{d,i} = \{\overline{\Psi_d}, \mathfrak{q}_{d,\rho,i}(\mathbf{s},x)\}$. This algorithm can be computed locally, which do not need communication. The generation of all proofs can be accelerated using an FFT-like way [21, 22].

5 HLE-DVC, a distributed vector commitment scheme

5.1 Definitions of the general DVC schemes and HLE-DVC

Suppose there are M machines, $\mathcal{P}_0, ..., \mathcal{P}_{M-1}$, and $\mathcal{P}_0$ is the master node. The entire vector $\mathbf{v}$ is distributed across M machines as $\mathbf{v} = (\mathbf{v}_0, ..., \mathbf{v}_{M-1})$, where each $\mathcal{P}_k$ only holds $\mathbf{v}_k$. The distributed vector commitment scheme aims to commit to $\mathbf{v}$ and to generate position proofs in a distributed fashion. $[\mathcal{P}_k : \mathbf{v}_k]_{k\in[0,M)}$ means $\mathbf{v}_k$ is only held by $\mathcal{P}_k$.

Definition 5.1. ***(Distributed vector commitment scheme).*** *A distributed vector commitment scheme consists of the following* PPT *algorithms* DVC = (Setup, DistCommit, Prove, Verify, BatchProve, BatchVerify, UpdComProof, AggProve, AggVerify*).*

The full version of the definition of a DVC scheme is given in Appendix E. Then we give the definition of our proposed DVC scheme, HLE-DVC.

Definition 5.2. ***(HLE-DVC scheme.)*** *An HLE-DVC scheme consists of the following* PPT *algorithms* (Setup, DistCommit, GenAux, GenPartialProof, GenAllPartialProof, Prove, ProveAll, Verify, BatchProve, BatchVerify, UpdComProof, Agg.Prove, Agg.Verify*): (definitions of last three algorithms are listed in Subsections 5.4 and 5.5.)*

- $\mathsf{Setup}(1^\lambda, N, M) \to ([\overline{pk_k}]_{k\in[0,M)}, [\overline{vk_k}, \overline{upk_k}]_{k\in[0,M)}, \overline{BatVk})$: Given the max vector length N, the number of machines M and the security parameter λ as input. Output proving keys $[\overline{pk_k}]_{k\in[0,M)}$, verification keys $[\overline{vk_k}]_{k\in[0,M)}$ and update keys $[\overline{upk_k}]_{k\in[0,M)}$, and batch verification key $\overline{BatVk}$.
- $\mathsf{DistCommit}([\mathcal{P}_k : \overline{pk_k}, \mathbf{v}_k]_{k\in[0,M)}) \to (C_\mathbf{v}, [C_{\mathbf{v}_k}]_{k\in[0,M)}[\mathcal{P}_k : \mathcal{F}_k(\mathbf{s},x)]_{k\in[0,M)})$. The vector to be committed are split into M subvectors $\{\mathbf{v}_k | k\in[0,M)\}$, and each machine $\mathcal{P}_k$ holds $\mathbf{v}_k$. Machines distributedly generate an HLE commitment $C_\mathbf{v}$.

• $\mathsf{GenAux}([\mathcal{P}_k : \overline{pk_k}, \mathcal{F}_k(\mathbf{s},x)]_{k\in[0,M)}) \to [\overline{\eta_k}]_{k\in[0,M)}$. Each $\mathcal{P}_k$ generates and publishes his auxdata $\overline{\eta_k} = (\eta_{k,0},...,\eta_{k,\rho-1})$.

• $\mathsf{GenPartialProof}([\overline{\eta_k}]_{k\in[0,M)}) \to \overline{\Psi_d}$. $\mathcal{P}_d$ generates his partial proof $\overline{\Psi_d}$.

• $\mathsf{GenAllPartialProof}([\overline{\eta_k}]_{k\in[0,M)}) \to [\overline{\Psi_d}]_{d\in[0,M)}$. The master node $\mathcal{P}_0$ generates all partial proofs.

• $\mathsf{Prove}(\overline{pk_k}, \overline{\Psi_d}, v^*, i, \mathcal{F}_d(\mathbf{s},x)) \to \Pi_{d,i}$. $\mathcal{P}_d$ outputs a proof $\Pi_{d,i}$ to prove that $v_{d,i} = v^*$ using the partial proof $\overline{\Psi_d}$.

• $\mathsf{Verify}(C_\mathbf{v}, d, i, v^*, \Pi_{d,i}, \overline{vk_d}) \to \{0,1\}$. $\mathcal{V}$ checks the proof $\Pi_{d,i}$ to verify if $v_{d,i} = v^*$.

• $\mathsf{ProveAll}([\mathcal{P}_d : \overline{pk_d}, \overline{\Psi_d}, \mathcal{F}_d(\mathbf{s},x)]_{d\in[0,M)}) \to [\Pi_{d,i}]_{d\in[0,M),i\in[0,n)}$. All machines output all position proofs $[\Pi_{d,i}]_{d\in[0,M),i\in[0,n)}$ for every element $[v_{d,i}]_{d\in[0,M),i\in[0,n)}$.

• $\mathsf{BatchProve}(\overline{pk_d}, d, \mathbf{I}, \mathbf{v}^*_\mathbf{I}, \mathcal{F}_d(\mathbf{s},x)) \to \Pi_\mathbf{I}$. Given an index set $\mathbf{I} = [(d,i_j)]_{j\in|\mathbf{I}|-1}$, $\mathcal{P}_d$ outputs a proof $\Pi_\mathbf{I}$ to prove that $\mathbf{v}^*_\mathbf{I}$ is the $\mathbf{I}$-subvector of $\mathbf{v}_d$.

• $\mathsf{BatchVerify}(C_\mathbf{v}, d, \mathbf{I}, \mathbf{v}^*_\mathbf{I}, \Pi_\mathbf{I}, \overline{vk_d}, \overline{BatVk}) \to \{0,1\}$. $\mathcal{V}$ checks the proof $\Pi_\mathbf{I}$ to verify if $\mathbf{v}^*_\mathbf{I}$ is the $\mathbf{I}$-subvector of $\mathbf{v}_d$.

5.2 Security definition

Definition 5.3. ***(Security definition of a DVC scheme).*** *A DVC scheme is secure if it satisfies computational binding, and the position proof satisfying correctness, soundness.*

When discussing the definition for computation binding, and position proof's correctness and soundness, the machines are regarded as a single entity $\mathcal{P}$ and $\mathsf{DistCommit}$ is simplified as: $C_\mathbf{v} \leftarrow \mathsf{DistCommit}(\mathbf{v})$. For simplicity, we refer to all public parameters output by Setup as pp, and pp will be implicitly input into other algorithms. PPT means 'Probabilistic Polynomial Time', and $\mathsf{negl}(\cdot)$ is a negligible function.

Computational binding. Given a fixed commitment C, for any PPT adversary $\mathcal{A}$, and he cannot find two distinct vectors $\mathbf{v}_{1\text{st}}$ and $\mathbf{v}_{2\text{nd}}$, s.t. $C_\mathbf{v} \leftarrow \mathsf{DistCommit}(\mathbf{v}_{1\text{st}})$ and $C_\mathbf{v} \leftarrow \mathsf{DistCommit}(\mathbf{v}_{2\text{nd}})$.

$$\Pr\left[\begin{array}{c}\mathsf{pp}\leftarrow\mathsf{Setup}(1^\lambda,M,n);(C,\mathbf{v}_{1\text{st}},\mathbf{v}_{2\text{nd}})\leftarrow\mathcal{A}(\mathsf{pp});\\ C_\mathbf{v}\leftarrow\mathsf{DistCommit}(\mathbf{v}_{1\text{st}})\wedge C_\mathbf{v}\leftarrow\mathsf{DistCommit}(\mathbf{v}_{2\text{nd}})\\ \wedge\mathbf{v}_{1\text{st}}\neq\mathbf{v}_{2\text{nd}}\end{array}\right]\leq\mathsf{negl}(\lambda).$$

Correctness. A proof generated correctly can pass the verification with probability 1.

DVC soundness. Soundness consists of two aspects: the soundness of individual proofs and the soundness of multiple-position proofs.

Individual soundness says that no adversary can output two inconsistent proofs for different values $v' \neq v$ at position (d,i) with respect to an adversarially-produced commitment $C_\mathbf{v}$.

Individual soundness. For any PPT adversary $\mathcal{A}$: (The verification key is used to accelerate the computation, and here we disregard the existence of the verification key.):

$$\Pr\left[\begin{array}{c}(\mathsf{pp})\leftarrow\mathsf{Setup}(1^\lambda,M,n);\\ (C_\mathbf{v},d,i,v,v',\Pi_{d,i},\Pi'_{d,i})\leftarrow\mathcal{A}(\mathsf{pp});\\ 1\leftarrow\mathsf{Verify}(C_\mathbf{v},d,i,v,\Pi_{d,i})\wedge\\ 1\leftarrow\mathsf{Verify}(C_\mathbf{v},d,i,v',\Pi'_{d,i})\wedge\\ v\neq v'\end{array}\right]\leq\mathsf{negl}(\lambda).$$

Multiple-position soundness (including batch soundness and cross-subvectors aggregation soundness) says that no adversary can output two inconsistent proofs, where one proves $v_{d,i} = v$ but another proves $v_{d,i} = v'$ with respect to an adversarially-produced commitment $C_\mathbf{v}$.

Batch soundness. For any PPT adversary $\mathcal{A}$, and the index set $\mathbf{I}$ and $\mathbf{I}'$ are only related to a single subvector $\mathbf{v}_d$:

$$\Pr\left[\begin{array}{c}(\mathsf{pp})\leftarrow\mathsf{Setup}(1^\lambda,N,M);\\ (C_\mathbf{v},\mathbf{I},\mathbf{I}',\mathbf{v}_\mathbf{I},\mathbf{v}'_\mathbf{I},\Pi_\mathbf{I},\Pi'_\mathbf{I})\leftarrow\mathcal{A}(\mathsf{pp});\\ 1\leftarrow\mathsf{Verify}(C_\mathbf{v},d,\mathbf{I},\mathbf{v}_\mathbf{I},\Pi_\mathbf{I})\wedge\\ 1\leftarrow\mathsf{Verify}(C_\mathbf{v},d,\mathbf{I}',\mathbf{v}'_\mathbf{I},\Pi'_\mathbf{I})\wedge\\ \exists(d,i)\in\mathbf{I}\cap\mathbf{I}',\\ \Pi_\mathbf{I}\text{ proves }v_{d,i}=v,\\ \Pi'_\mathbf{I}\text{ proves }v_{d,i}=v',\\ \text{s.t.}v\neq v'\end{array}\right]\leq\mathsf{negl}(\lambda).$$

5.3 Construction of distributed vector commitment scheme

Below we introduce each algorithm of the HLE-DVC.

• $\mathsf{Setup}(1^\lambda,N,M) \to ([\overline{pk_k}]_{k\in[0,M)}, [\overline{vk_k}, \overline{upk_k}]_{k\in[0,M)}, \overline{BatVk})$. Let $\rho = \log M$, $n = N/M$, the max polynomial degree $\mathbf{D} = (\mathbf{2}^\rho, n)$ and run $\mathsf{PST.Setup}(1^\lambda,\mathbf{D}) \to \mathsf{pp}_{\mathsf{PST}}$. For $k \in [0,M)$, compute $\mathsf{pp}'_{\mathsf{PST},k} = [\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle k\rangle)\cdot x^i)]_{i\in[0,n)}$. For $k \in [0,M)$, compute $\mathsf{pp}''_{\mathsf{PST},k} = [\mathsf{PST}(\frac{\mathsf{eq}(\mathbf{s},\langle k\rangle)x^i}{s_c-(1-\langle k\rangle_c)})]_{i\in[0,n),c\in[0,\rho)}$. Let $\overline{BatVk} = [\alpha^i_x Q]_{i\in[0,n)}$[3]. The proving key for the machine $\mathcal{P}_k$ is set as:

$$\overline{pk_k} = \{\mathsf{pp}'_{\mathsf{PST},k}, \mathsf{pp}''_{\mathsf{PST},k}\}$$

The verification key for $\mathcal{P}_k$ is:

$$\overline{vk_k} = ((\alpha_{s_0} - \langle k\rangle_0)Q, ..., (\alpha_{s_{\rho-1}} - \langle k\rangle_{\rho-1})Q, \mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle k\rangle)))$$

The update key for $\mathcal{P}_k$ is

$$\overline{upk_k} = \{[\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle k\rangle)\frac{x^n-1}{x-\omega^i_n})]_{i\in[0,n)}, [\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle k\rangle)\mathcal{L}_i(x))]_{i\in[0,n)},$$
$$[\mathsf{PST}(\frac{\mathsf{eq}(\mathbf{s},\langle k\rangle)\mathcal{L}_i(x)}{s_c-(1-\langle k\rangle_c)})]_{i\in[0,n),c\in[0,\rho)}, [\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle k\rangle)\frac{\mathcal{L}_j(x)-1}{x-\omega^i_n})]_{i\in[0,n)}\}$$

Output proving keys $[\overline{pk_k}]_{k\in[0,M)}$, verification keys $[\overline{vk_k}]_{k\in[0,M)}$, update keys $[\overline{upk_k}]_{k\in[0,M)}$ and batch verification key $\overline{BatVk}$.

Note that the machine $\mathcal{P}_k$ only needs to store $\overline{pk_k}$ and $\overline{upk_k}$, whose storage complexities are $O(N/M\log M)$. The verifier who wants to verify the proof related to the k-th subvector $\mathbf{v}_k$ only needs to store the k-th verification key $\overline{vk_k}$. Moreover, verifing batch proof needs the batch verification key $\overline{BatVk}$.

• $\mathsf{DistCommit}([\mathcal{P}_k : \overline{pk_k}, \mathbf{v}_k]_{k\in[0,M)}) \to (C_\mathbf{v}, [C_{\mathbf{v}_k}]_{k\in[0,M)}[\mathcal{P}_k : \mathcal{F}_k(\mathbf{s},x)]_{k\in[0,M)})$. Each machine $\mathcal{P}_k$ extends his subvector $\mathbf{v}_k$ to a polynomial $\mathfrak{f}_k(x)$, s.t. for

[3] $\mathsf{pp}'_{\mathsf{PST},k}$ enables computing $\mathsf{PST}(\mathcal{F}_k(\mathbf{s},x)) = \mathsf{PST}(\mathsf{eq}(s,\langle k\rangle)\mathfrak{f}_k(x))$ in $O(n)$ time. (Since $\mathfrak{f}(x) = \sum_{i=0}^{n-1} f_i x^i$, then we have: $\mathsf{PST}(\mathcal{F}(\mathbf{s},x)) = \mathsf{PST}(\mathsf{eq}(s,\langle k\rangle)\mathfrak{f}_k(x)) = \sum_{i=0}^{n-1} f_i \cdot \mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle k\rangle)\cdot x^i)$.) $\mathsf{pp}''_{\mathsf{PST},k}$ enables computing $\mathsf{PST}(\frac{\mathcal{F}_k(\mathbf{s},x)}{s_c-(1-\langle k\rangle_c)})$ in $O(n)$ time. $\overline{BatVk}$ is used in the verification for the batch position proof.

$i \in [0,n), \mathfrak{f}_k(\omega_n^i) = v_{k,i}$. Then let $\mathcal{F}_k(\mathbf{s},x) = \mathsf{eq}(\langle k \rangle, \mathbf{s}) \cdot \mathfrak{f}_k(x)$. $\mathcal{P}_k$ computes the sub-commitment $C_{\mathbf{v}_k} = \mathsf{PST}(\mathcal{F}_k(\mathbf{s},x))$. Each machine $\mathcal{P}_k$ sends his sub-commitment $C_{\mathbf{v}_k}$ to the master node $\mathcal{P}_0$.

$\mathcal{P}_0$ computes $C_{\mathbf{v}} = \sum_{k=0}^{M-1} C_{\mathbf{v}_k}$. Essentially, $C_{\mathbf{v}}$ is the PST commitment to the HLE polynomial of the vector $\mathbf{v}$. Output $C_{\mathbf{v}}$ as the HLE vector commitment.

- $\mathsf{GenAux}([\mathcal{P}_k : \overline{pk_k}, \mathcal{F}_k(\mathbf{s},x)]_{k\in[0,M)}) \to [\overline{\eta_k}]_{k\in[0,M)}$. For all $c \in [0,\rho)$, each $\mathcal{P}_k$ computes $\eta_{k,c} = \mathsf{PST}(\frac{\mathcal{F}_k(\mathbf{s},x)}{s_c - (1-\langle k \rangle_c)})$.

$\mathcal{P}_k$ outputs and publishes his auxdata $\overline{\eta_k} = [\eta_{k,c}]_{c\in[0,\rho)}$.

- $\mathsf{GenPartialProof}([\overline{\eta_k}]_{k\in[0,M)}) \to \overline{\Psi_d}$. Each machine $\mathcal{P}_d$ computes his partial proof $\overline{\Psi_d}$. For $c \in [0,\rho)$, $\mathcal{P}_d$ computes $\pi_{d,c} = \sum_{k\in Q_{d,c}} \eta_{k,c}$, where $Q_{d,c} = \{k | k \in [0,M), \langle k \rangle_0 = \langle d \rangle_0, ..., \langle k \rangle_{c-1} = \langle d \rangle_{c-1}, \langle k \rangle_c \neq \langle d \rangle_c\}$. Finally, set the partial proof as $\overline{\Psi_d} = (\pi_{d,0}, ..., \pi_{d,\rho-1})$.
- $\mathsf{GenAllPartialProof}([\overline{\eta_k}]_{k\in[0,M)}) \to [\overline{\Psi_d}]_{d\in[0,M)}$. The algorithm is essentially the same as in Section 4.3.2, except that auxdata $\eta_{k,c}$ is no longer a polynomial but its PST commitment. The master node $\mathcal{P}_0$ generates c HLE-Aux trees $[\mathtt{T}_c]_{c\in[0,\rho)}$. For all $d \in [0,M), c \in [0,\rho)$, $\mathcal{P}_0$ sets $\pi_{d,c} = \mathtt{T}_c.\mathtt{Node}_{b_0...b_c}$, where $b_0 = \langle d \rangle_0, ..., b_{c-1} = \langle d \rangle_{c-1}$ and $b_c = 1 - \langle d \rangle_c$. Finally, for $d \in [0,M)$, $\mathcal{P}_0$ sets the partial proof of $\mathcal{P}_d$ as $\overline{\Psi_d} = (\pi_{d,0}, ..., \pi_{d,\rho-1})$.
- $\mathsf{Prove}(\overline{pk_k}, \overline{\Psi_d}, v^*, i, \mathcal{F}_d(\mathbf{s},x)) \to \Pi_{d,i}$. To prove that $v_{d,i} = v^*$, $\mathcal{P}_d$ computes $\mathfrak{q}^*_{d,i}(x) = \frac{\mathfrak{f}_d(x) - v^*}{x - \omega_n^i}$. and $\pi_{d,\rho,i} = \mathsf{PST}(\mathsf{eq}(\mathbf{s}, \langle d \rangle)\mathfrak{q}^*_{d,i}(x))$. Then $\mathcal{P}_d$ sets the position proof $\Pi_{d,i} = \{\overline{\Psi_d}, \pi_{d,\rho,i}\}$.
- $\mathsf{Verify}(C_{\mathbf{v}}, d, i, v^*, \Pi_{d,i}, \overline{vk_d}) \to \{0,1\}$. $\mathcal{V}$ parses the $\Pi_{d,i} = (\pi_{d,0}, ..., \pi_{d,\rho-1}, \pi_{d,\rho,i})$. According to lemma 4.3, to verify if $v_{d,i} \stackrel{?}{=} v^*$, $\mathcal{V}$ checks if

$$e(C_{\mathbf{v}} - v^* \cdot \mathsf{PST}(\mathsf{eq}(\mathbf{s}, \langle d \rangle)), Q) \stackrel{?}{=} \prod_{c=0}^{\rho-1} e(\pi_{d,c}, (\alpha_{s_c} - \langle d \rangle_c)Q) \cdot e(\pi_{d,\rho,i}, (\alpha_x - \omega_n^i)Q)$$

- $\mathsf{ProveAll}([\mathcal{P}_d : \overline{pk_d}, \overline{\Psi_d}, \mathcal{F}_d(\mathbf{s},x)]_{d\in[0,M)}) \to [\Pi_{d,i}]_{d\in[0,M),i\in[0,n)}$. For $d \in [0,M)$, each machine $\mathcal{P}_d$ computes for all $i \in [0,n)$, $\mathfrak{q}^*_{d,i}(x) = \frac{\mathfrak{f}_d(x) - v_{d,i}}{x - \omega_n^i}$. Then computes $\pi_{d,\rho,i} = \mathsf{PST}(\mathsf{eq}(\mathbf{s}, \langle d \rangle)\mathfrak{q}^*_{d,i}(x))$, and sets the position proof for $v_{d,i}$ as $\Pi_{d,i} = \{\overline{\Psi_d}, \pi_{d,\rho,i}\}$. Specially, the Feist-Khovratovich's [13, 21, 22] technique enables the machine to compute all $[\pi_{d,\rho,i}]_{i\in[0,n)}$ in $\mathrm{O}(n \log n)$ time.
- $\mathsf{BatchProve}(\overline{pk_d}, d, \mathbf{I}, \mathbf{v}^*_{\mathbf{I}}, \mathcal{F}_d(\mathbf{s},x)) \to \Pi_{\mathbf{I}}$. Given an index set $\mathbf{I} = [(d, i_j)]_{j\in[0,|\mathbf{I}|-1)}$, where all indices in the set $\mathbf{I}$ are related to the vector $\mathbf{v}_d$, and a claimed $\mathbf{I}$-subvector $\mathbf{v}^*_{\mathbf{I}} = [v^*_{d,i_j}]_{(d,i_j)\in\mathbf{I}}$. Denote the set of all i_j involved in $\mathbf{I}$ as $I = [i_j]_{(d,i_j)\in\mathbf{I}}$. $\mathcal{P}_d$ computes $A_I(x) = \prod_{i\in I}(x - \omega_n^i)$ and extends $\mathbf{v}^*_{\mathbf{I}}$ to a univariate polynomial $\mathfrak{r}(x)$, s.t. for all $i \in I$, $\mathfrak{r}(\omega_n^i) = v^*_{d,i}$. $\mathcal{P}_d$ computes $\mathfrak{q}^*_{d,I}(x) = \frac{\mathfrak{f}_d(x) - \mathfrak{r}(x)}{A_I(x)}$. Compute $\pi_{d,\rho,I} = \mathsf{PST}(\mathsf{eq}(\mathbf{s}, \langle d \rangle) \cdot \mathfrak{q}^*_{d,I}(x))$. Finally, set the batch proof $\Pi_{\mathbf{I}} = \{\overline{\Psi_d}, \pi_{d,\rho,I}\}$.
- $\mathsf{BatchVerify}(C_{\mathbf{v}}, d, \mathbf{I}, \mathbf{v}^*_{\mathbf{I}}, \Pi_{\mathbf{I}}, \overline{vk_d}, \overline{BatVk}) \to \{0,1\}$. $\mathcal{V}$ sets $I = [i_j]_{(d,i_j)\in\mathbf{I}}$, computes $A_I(x) = \prod_{i\in I}(x - \omega_n^i)$, and extends $\mathbf{v}^*_{\mathbf{I}}$ to a polynomial $\mathfrak{r}(x)$. Then $\mathcal{V}$ parses $\Pi_{\mathbf{I}} = (\pi_{d,0}, ..., \pi_{d,\rho-1}, \pi_{d,\rho,I})$ and checks if $e(C_{\mathbf{v}} - \mathsf{PST}(\mathsf{eq}(\mathbf{s}, \langle d \rangle) \cdot \mathfrak{r}(x)), Q) \stackrel{?}{=} \prod_{c=0}^{\rho-1} e(\pi_{d,c}, (\alpha_{s_c} - \langle d \rangle_c)Q) \cdot e(\pi_{d,\rho,I}, A_I(\alpha_x)Q)$.

5.4 Update the commitment and all proofs

When $\mathcal{P}_u$ updates his vector $\mathbf{v}_u$ to $\mathbf{v}'_u$ by updating $v'_{u,j} = v_{u,j} + \delta$, the HLE commitment need to be updated to $C'_{\mathbf{v}}$. All partial proofs and position proofs need to be updated correspondingly. The update algorithm is formalized as follow:

- $\mathsf{UpdComProof}(C_{\mathbf{v}}, \overline{upk_d}, [\overline{\Psi_d}]_{d\in[0,M)}, \overline{\eta_u}, v'_{u,j}, [\Pi_{d,i}]_{d\in[0,M),i\in[0,n)}) \to (C'_{\mathbf{v}}, [\overline{\Psi_d}']_{d\in[0,M)}, [\Pi'_{d,i}]_{d\in[0,M),i\in[0,n)})$.

Due to the page limit, we present the construction of $\mathsf{UpdComProof}$ in Appendix F.

5.5 Aggregate proofs

Given an index set $\mathbf{I} = [(d_j, i_j)]_{j\in[0,|\mathbf{I}|-1)}$, and a claimed $\mathbf{I}$-subvector $\mathbf{v}^*_{\mathbf{I}} = [v^*_{d,i_j}]_{(d,i_j)\in\mathbf{I}}$, one can efficiently aggregate several individual proofs $[\Pi_{d_j,i_j}]_{j\in[0,|\mathbf{I}|-1)}$ (the j-th proof Π_{d_j,i_j} is a proof for $v_{d_j,i_j} = v^*_j$) into a single, succinct proof $\Pi_{Agg,\mathbf{I}}$. The algorithms for aggregation are formalized as follows:

- $\mathsf{Agg.Prove}(\mathbf{I}, \mathbf{v}^*_{\mathbf{I}}, [\Pi_{d_j,i_j}]_{(d_j,i_j)\in\mathbf{I}}) \to \Pi_{Agg,\mathbf{I}}$.
- $\mathsf{Agg.Verify}(C_{\mathbf{v}}, \mathbf{I}, \mathbf{v}^*_{\mathbf{I}}, \Pi_{Agg,\mathbf{I}}) \to \{0,1\}$.

Proof aggregation can be categorized into two cases:

Single-subvector aggregation: all indices in the set $\mathbf{I}$ are related to a single subvector $\mathbf{v}_d$ (i.e. $\mathbf{I} = [(d, i_j)]_{j\in[0,|\mathbf{I}|-1)}$). We achieve single-subvector aggregation using partial fraction decomposition [13]. Due to page limits, we present the construction of single-subvector aggregation in Appendix G.2.

Cross-subvectors aggregation: the set $\mathbf{I} = [(d_j, i_j)]_{j\in[0,|\mathbf{I}|-1)}$ We achieve cross-subvector aggregation through a two-step process. First, the proofs are grouped and aggregated within each subvector, resulting in M single-subvector aggregation proofs. Then, these M batch proofs are further aggregated using an IPA-based method. Due to the page limits, we present the construction of single-subvector aggregation in Appendix G.3.

5.6 Security analysis

Theorem 5.1. *HLE-DVC scheme is a secure vector commitment scheme according to the security definition 5.3. HLE-DVC is binding if the PST PCS is binding. The individual position proof, batch proof and the aggregation proof satisfies correctness and soundness, if the PST PCS is computational binding and knowledge sound, the q-SDH problem and q-BSDH problem hold, and the IPA scheme is secure.*

PROOF: The full proof can be found in the Appendix E. □

5.7 Complexity analysis

Table 2: Time complexities of algorithms.

Setup	N	Verify	$\log M$
DistCommit	$\frac{N}{M}+M$	BatchProve	$\frac{N}{M}+\lvert\mathbf{I}\rvert\log^2\lvert\mathbf{I}\rvert$
GenAux	$\frac{N}{M}\log M$	BatchVerify	$\lvert\mathbf{I}\rvert\log^2\lvert\mathbf{I}\rvert$
GenPartialProof	M	Agg.Prove (single)	$\lvert\mathbf{I}\rvert\log^2\lvert\mathbf{I}\rvert$
GenAllPartialProof	$M\log M$	Agg.Prove (cross)	$\lvert\mathbf{I}\rvert\log M$
Prove	$\frac{N}{M}$	Agg.Verify (cross)	$\log(\lvert\mathbf{I}\rvert\log M)$
ProveAll	$\frac{N}{M}\log\frac{N}{M}$	UpdComProof	$\frac{N}{M}+\log M$

Table 3: Storage and communication size complexities of proving key $\overline{pk_k}$, verification key $\overline{vk_k}$, update key $\overline{upk_k}$, auxdata $\overline{\eta_k}$, partial proof $\overline{\psi_k}$, and others. There are four algorithms involved in communication, and their communication size per machine is presented.

storage size			
$\overline{pk_k}$	$N/M\log M$	single proof $\Pi_{d,i}$	$\log M$
$\overline{upk_k}$	$\frac{N}{M}\log M$	all proofs $[\Pi_{d,i}]_{d\in[0,M),i\in[0,n)}$	$M+N$
$\overline{vk_k}$	$\log M$	batch proof $\Pi_{\mathbf{I}}$	$\log M$
$\overline{\eta_k}$	$\log M$	single-agg proof $\Pi_{Agg,\mathbf{I}}$	$\log M$
$\overline{\psi_k}$	$\log M$	cross-agg proof $\Pi_{Agg,\mathbf{I}}$	$\log(\lvert\mathbf{I}\rvert\log M)$
communication size (per machine)			
DistCommit	1	GenAllPartialProof	$\log M$
GenAux	$\log M$	UpdComProof	1

Table 2 shows time complexities of the algorithms, and Table 3 shows storage complexities of the algorithms' outputs and communication cost of algorithms. Note that the size of a single proof is $(\log M+1)$. However, Lemma 4.3 indicates that the total size of position proofs for all elements in a subvector is $(\log M+N/M)$. Furthermore, the HLE-Aux tree shows that all partial proofs collectively occupy a total size of M. In summary, the overall size of all proofs is $(N+M)$.

6 Experiments

We evaluate the performance of the HLE-DVC scheme using mcl libaray (Go language) on a Tencent cloud server (16-core CPU, 64 GB memory, 5 Mbps bandwidth). We use the curve BLS12-381 as the bilinear pairing group. An element in $\mathbb{G}_1/\mathbb{G}_2/\mathbb{G}_T$ is stored as 48/96/576 bytes. A field element in $\mathbb{F}$ is stored as 32 bytes. We choose to use the non-hiding version PST PCS, so that the DVC scheme is also non-hiding. The whole implementation is open-sourced: https://github.com/gaozheruiye/HLE-DVC and https://zenodo.org/records/18163723.

Storage overhead. Table 4 shows the storage cost of some outputs in HLE-DVC scheme. HLE-DVC has logarithmic proof size, When $M=256$, $N=2^{30}$, though the size of entire proving key is 258 GB, each machine $\mathcal{P}_k$ only needs to store $\mathrm{pp}'_{\mathrm{PST},k}$ and $\mathrm{pp}''_{\mathrm{PST},k}$, i.e., 1.8 GB. Both the single proof size and batch proof size are 432 bytes. As analyzed in Subsection

Table 4: Storage overhead of the HLE-DVC scheme.

Storage size				
		$N=2^{26}$	$N=2^{28}$	$N=2^{30}$
Proving key per machine (MB) $\overline{pk_d}$	$M=16$	1,006.6	4,026.5	16,106.1
	$M=64$	352.3	1,409.3	5,637.1
	$M=256$	113.2	453.0	1,811.9
Verification key (bytes) $\overline{vk_d}$	$M=16$	432	432	432
	$M=64$	624	624	624
	$M=256$	816	816	816
Single proof (bytes) $\Pi_{d,i}$	$M=16$	240	240	240
	$M=64$	336	336	336
	$M=256$	432	432	432
Batch proof (bytes) $\Pi_{\mathbf{I}}$ ($\lvert\mathbf{I}\rvert=1024$)	$M=16$	240	240	240
	$M=64$	336	336	336
	$M=256$	432	432	432
All single proofs (GB) $[\Pi_{d,i}]_{d\in[0,M),i\in[0,n)}$	$M=16$	3.2	12.9	51.5
	$M=64$	3.2	12.9	51.5
	$M=256$	3.2	12.9	51.5
Cross-aggregation proof (Kb) $\Pi_{Agg,\mathbf{I}}$ ($\lvert\mathbf{I}\rvert=1024$)	$M=16$	3.84	3.84	3.84
	$M=64$	21.5	21.5	21.5
	$M=256$	47.5	47.5	47.5
Communication size				
Auxdata (bytes) $\overline{\eta}_d$	$M=16$	192	192	192
	$M=64$	288	288	288
	$M=256$	384	384	384
Partial proof (bytes) $\overline{\psi}_d$	$M=16$	192	192	192
	$M=64$	288	288	288
	$M=256$	384	384	384

Table 5: Time overhead of the HLE-DVC scheme ($M=256$, $N=2^{30}$, $\lvert\mathbf{I}\rvert=1024$, and the time unit is seconds).

Setup	665,911	BatchProve	324
DistCommit	364	BatchVerify	0.006
ProveAll	17,515	Agg.Prove (single)	0.123
Verify	0.005	Agg.Prove (cross)	9.49
UpdComProof	648	Agg.Verify (cross)	0.044

5.7, the total size of all proofs for elements is $N+M$ elements in $\mathbb{G}_1$, i.e. 5.1 GB.

Communication overhead. Table 4 shows the communication cost of the HLE-DVC scheme. There are only three scenarios in total that involve communication. (1). DistCommit requires the constant 48 bytes communication per machine for transferring sub-commitment. (2). Generating all proofs requires logarithmic communication for each machine to upload the auxdata and download the partial proof from the master node. When $M = 256$ and $N = 2^{30}$, the total communication cost per machine is only 0.768 KB. (3). Updating all proof requires the constant 48 bytes communication per machine.

Time overhead. In Table 5, we tested time overheads of algorithms. When $M = 256$, $N = 2^{30}$, generating all proofs (including generating auxdata, partial proofs and the proofs) costs 17,515 s, achieving a 142× parallel speedup compared to Hyperproofs. Updating all proofs costs 648 s and aggregating 1024 cross-subvectors proofs costs 9.49 s.

Table 6: Comparisons among our work, Hyperproofs-DVC, KZG-DPCS-DVC, and a traditional VC Hyperproofs [3]. "H∼" represents Hyperproofs. Hyperproofs-DVC is proposed by us by applying the techniques in Section 4 to Hyperproofs. $M = 256, N = 2^{30}, |\mathbf{I}| = 1024$. "s:" means single-subvector aggregation and "c:" means cross-subvectors aggregation. When s/c are not explicitly indicated, the aggregation complexity is the same for both cases. The data for KZG-DPCS-DVC are obtained through theoretical analysis.

	Our work		Strawman	VC
Schemes	HLE-DVC	H∼-DVC	KZG-DPCS-DVC	H∼
Commit (s)	364	364	364	93201
Proof size (bytes)	432	1,440	48	1,440
Prove all (s)	17,515	10,100	3.48×10^{11}	2,493,084
Communication (KB)	0.768	0.768	5.15×10^{7}	⊘
Verify (s)	0.005	0.017	0.0006	0.017
Update all proofs (s)	648	0.002	648	0.002
Agg-proof size (Kb)	s: 0.43; c: 47.5	64.8	0.048	64.8
Agg-proof time (s)	s: 0.12; c: 9.49	126.5	0.12	126.5

Comparison. We compare HLE-DVC with Hyperproofs [3], Hyperproofs-DVC (introduced in Section 2.2) and the strawman KZG-DPCS [17]-DVC in Table 6. Benefiting from the distributed architecture, the HLE-DVC scheme enables the parallel computation, achieving an 142× speedup over the single-machine VC, Hyperproofs. HLE-DVC achieves the shortest proof size and verification cost.

Ethical Considerations

This work is primarily theoretical and focuses on cryptographic foundations. It does not rely on any real-world datasets, involves no human participants or private entities, and thus does not involve any stakeholders. Consequently, it does not pose a direct risk of harm or infringement of interests to any individuals or groups.

Although the work is theoretical in nature, we acknowledge that most cryptographic primitives may have dual-use implications. Techniques related to DVC schemes can be incorporated into larger systems that operate on sensitive data or support privacy-preserving functionalities. In inappropriate contexts, such systems might be used to facilitate opaque data handling or amplify existing imbalances in information control. These risks arise at the system and governance levels rather than from the primitive itself.

We believe the benefits outweigh these risks: DVC schemes enhance the scalability of existing VC constructions, accelerate proof generation through parallel computation, and ensure the verifiability of data in distributed databases. These properties generally mitigate harmful behavior by providing strong security guarantees. Responsible deployments of DVC-applications require careful system-level design, clear governance, and proper auditing to prevent unintended negative applications.

Open Science

The implementation of HLE-DVC is presented in an GitHub repository, https://github.com/gaozheruiye/HLE-DVC and in a permanent link https://zenodo.org/records/18163723. We have included detailed instructions for setup and execution to ensure that other researchers can reproduce our results.

This paper does not involve any data set, and the values in the committed vector are randomly generated.

Acknowledgements

We thank the anonymous reviewers and our shepherd for their invaluable feedback.

Yuncong Hu is supported by the National Natural Science Foundation of China (Grant Nos. 62502306), the ExploreX project of SJTU, and gifts/awards from Xiaomi. Rui Gao and Huaqun Wang are supported in part by the National Natural Science Foundation of China under Grant U23B2002, Grant 62272238. Zhiguo Wan is supported by the National Natural Science Foundation of China under Grant 62272425.

We would like to thank Jianbang Dai for his assistance with the implementation and code development, which greatly contributed to this work.

References

[1] T. Dryja, "Utreexo: A dynamic hash-based accumulator optimized for the bitcoin utxo set," *Cryptology ePrint Archive*, 2019.

[2] L. Reyzin, D. Meshkov, A. Chepurnoy, and S. Ivanov, "Improving authenticated dynamic dictionaries, with ap-

plications to cryptocurrencies," in *Financial Cryptography and Data Security: 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers 21*. Springer, 2017, pp. 376–392.

[3] S. Srinivasan, A. Chepurnoy, C. Papamanthou, A. Tomescu, and Y. Zhang, "Hyperproofs: Aggregating and maintaining proofs in vector commitments," in *USENIX Security 22*, 2022, pp. 3001–3018.

[4] S. Gorbunov, L. Reyzin, H. Wee, and Z. Zhang, "Pointproofs: Aggregating proofs for multiple vector commitments," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, CCS 2020*, 2020, pp. 2007–2023.

[5] Z. Luo, Y. Jia, A. V. O. Gracia, and A. Kate, "Cauchyproofs: Batch-updatable vector commitment with easy aggregation and application to stateless blockchains," *Cryptology ePrint Archive*, 2025.

[6] D. Catalano, D. Fiore, and M. Messina, "Zero-knowledge sets with short proofs," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2008, pp. 433–450.

[7] D. Catalano and D. Fiore, "Vector commitments and their applications," in *Public-Key Cryptography–PKC 2013*. Springer, 2013, pp. 55–72.

[8] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Advances in Cryptology, ASIACRYPT 2010*. Springer, 2010, pp. 177–194.

[9] Y. Hu, K. Hooshmand, H. Kalidhindi, S. J. Yang, and R. A. Popa, "Merkle^2: A low-latency transparency log system," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 285–303.

[10] Y. Hu, S. Kumar, and R. A. Popa, "Ghostor: toward a secure {Data-Sharing} system from decentralized trust," in *17th USENIX symposium on networked systems design and implementation (NSDI 20)*, 2020, pp. 851–877.

[11] R. Gao, Z. Wan, Y. Hu, and H. Wang, "A succinct range proof for polynomial-based vector commitment," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 3152–3166.

[12] Y. Ji and K. Chalkias, "Generalized proof of liabilities," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3465–3486.

[13] A. Tomescu, I. Abraham, V. Buterin, J. Drake, D. Feist, and D. Khovratovich, "Aggregatable subvector commitments for stateless cryptocurrencies," in *International Conference on Security and Cryptography for Networks, SCN 2020*. Springer, 2020, pp. 45–64.

[14] A. Chepurnoy, C. Papamanthou, S. Srinivasan, and Y. Zhang, "Edrax: A cryptocurrency with stateless transaction validation," *Cryptology ePrint Archive*, 2018.

[15] A. Dutta and S. Vijayakumaran, "Mprove: A proof of reserves protocol for monero exchanges," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2019, pp. 330–339.

[16] T. Xie, J. Zhang, Z. Cheng, F. Zhang, Y. Zhang, Y. Jia, D. Boneh, and D. Song, "zkbridge: Trustless cross-chain bridges made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 3003–3017.

[17] T. Liu, T. Xie, J. Zhang, D. Song, and Y. Zhang, "Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 1777–1793.

[18] C. Li, P. Zhu, Y. Li, C. Hong, W. Qu, and J. Zhang, "Hyperpianist: Pianist with linear-time prover and logarithmic communication cost," *Cryptology ePrint Archive*, 2024.

[19] M. Rosenberg, T. Mopuri, H. Hafezi, I. Miers, and P. Mishra, "Hekaton: Horizontally-scalable zksnarks via proof aggregation," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 929–940.

[20] Y. Hu, P. Mishra, X. Wang, J. Xie, K. Yang, Y. Yu, and Y. Zhang, "DFS: delegation-friendly zksnark and private delegation of provers," in *34th USENIX Security Symposium, USENIX Security 2025*, 2025, pp. 2065–2084.

[21] D. Feist and D. Khovratovich, "Fast amortized KZG proofs," 2023. [Online]. Available: https://eprint.iacr.org/2023/033

[22] J. Zhang, T. Xie, T. Hoang, E. Shi, and Y. Zhang, "Polynomial commitment with a {One-to-Many} prover and applications," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2965–2982.

[23] A. Tomescu, R. Chen, Y. Zheng, I. Abraham, B. Pinkas, G. G. Gueta, and S. Devadas, "Towards scalable threshold cryptosystems," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 877–893.

[24] T. Lu, Y.-F. Chen, B. Hechtman, T. Wang, and J. Anderson, "Large-scale discrete fourier transform on tpus," *IEEE Access*, vol. 9, pp. 93 422–93 432, 2021.

[25] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the theory and application of cryptographic techniques*. Springer, 1987, pp. 369–378.

[26] M. Campanelli, A. Nitulescu, C. Ràfols, A. Zacharakis, and A. Zapico, "Linear-map vector commitments and their practical applications," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2022, pp. 189–219.

[27] W. Wang, A. Ulichney, and C. Papamanthou, "Balanceproofs: Maintainable vector commitments with fast aggregation," in *USENIX Security 23*, 2023, pp. 4409–4426.

[28] C. Papamanthou, E. Shi, and R. Tamassia, "Signatures of correct computation," in *Theory of Cryptography Conference, TCC 2013*. Springer, 2013, pp. 222–242.

[29] B. Bünz, M. Maller, P. Mishra, N. Tyagi, and P. Vesely, "Proofs for inner pairing products and applications," in *Advances in Cryptology–ASIACRYPT 2021*. Springer, 2021, pp. 65–97.

[30] B. Libert and M. Yung, "Concise mercurial vector commitments and independent zero-knowledge sets with short proofs," in *Theory of Cryptography Conference*. Springer, 2010, pp. 499–517.

[31] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Fast reed-solomon interactive oracle proofs of proximity," in *international colloquium on automata, languages, and programming, ICALP 2018*, vol. 14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018, pp. 1–17.

[32] A. Kattis, K. Panarin, and A. Vlasov, "Redshift: Transparent snarks from list polynomial commitment iops," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1400, 2019.

[33] I. A. Tomescu Nicolescu, "How to keep a secret and share a public key (using polynomial commitments)," Ph.D. dissertation, Massachusetts Institute of Technology, 2020.

[34] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to iops and stateless blockchains," in *Advances in Cryptology–CRYPTO 2019*. Springer, 2019, pp. 561–586.

[35] R. W. Lai and G. Malavolta, "Subvector commitments with application to succinct arguments," in *Advances in Cryptology–CRYPTO 2019*. Springer, 2019, pp. 530–560.

[36] M. Campanelli, D. Fiore, N. Greco, D. Kolonelos, and L. Nizzardo, "Incrementally aggregatable vector commitments and applications to verifiable decentralized storage," in *Advances in Cryptology–ASIACRYPT 2020*. Springer, 2020, pp. 3–35.

[37] C. Papamanthou, E. Shi, R. Tamassia, and K. Yi, "Streaming authenticated data structures," in *Advances in Cryptology–EUROCRYPT 2013*. Springer, 2013, pp. 353–370.

[38] Y. Qian, Y. Zhang, X. Chen, and C. Papamanthou, "Streaming authenticated data structures: Abstraction and implementation," in *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, 2014, pp. 129–139.

[39] H. Wang, S.-M. Yiu, Y. Zhao, and Z. L. Jiang, "Updatable, aggregatable, succinct mercurial vector commitment from lattice," in *IACR International Conference on Public-Key Cryptography*. Springer, 2024, pp. 3–35.

[40] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo, "Structure-preserving signatures and commitments to group elements," in *Advances in Cryptology–CRYPTO 2010*. Springer, 2010, pp. 209–236.

[41] S. Agrawal and S. Raghuraman, "Kvac: Key-value commitments for blockchains and beyond," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020, pp. 839–869.

[42] J. Kuszmaul, "Verkle trees," *Verkle Trees*, vol. 1, no. 1, pp. 1–10, 2019.

A Application: distributed verifiable elementary databases (DVEDB)

Traditional VC-based verifiable elementary databases (VEDB) assume that a single server holds the entire data vector[4] and can thus compute commitments and generate position proofs. However, real-world deployments often distribute the database across multiple machines due to privacy or legal restrictions and storage constraints when datasets are too large for one server. In such distributed settings, no party possesses the complete vector, making traditional VC schemes unusable: neither the global commitment nor correct position proofs can be generated. This fundamental limitation prevents existing VC-based VEDB systems from supporting modern distributed architectures.

[4] How to transform a database into a vector: If the data is numerical (e.g., clients are account holders, the server is a bank, and the data represents account balances), then the database can be directly represented as a vector of data values. If the data consists of files, then the database is represented as a vector of hashes of the data files.

To address this issue, we introduce the notion of distributed verifiable elementary database (DVEDB), which involves two types of entities: clients and servers. The number of clients is less than N, and they are partitioned across the M servers, with each server $\mathcal{P}_k$ responsible for at most $n = N/M$ clients.

Specifically, the server $\mathcal{P}_k$ holds a subdatabase DB_k, which stores data $\mathbf{v}_k = [v_{k,i}]_{i\in[0,n)}$ of many clients $[C_{k,i}]_{i\in[0,n)}$. The entire database is represented as a concatenated vector $\mathbf{v} = (\mathbf{v}_0, ..., \mathbf{v}_{M-1})$. DVEDB systems leverage the DVC algorithms to implement the same functionalities as that in VEDB.

The DVC-based DVEDB system involves these algorithms: $\mathsf{DVEDB} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{QueryProve}, \mathsf{Verify}, \mathsf{Update})$. Common operations in database scenarios, such as additions, deletions, and modifications, can all be implemented using the Update algorithm. The constructions of the algorithms are as follows:

- $\mathsf{Setup}(1^\lambda, N, M, [\mathcal{P}_k : DB_k]_{k\in[0,M)}) \to (\mathsf{pp}, \mathbf{v})$: N is the max size of the database, M is the number of machines, and $n = N/M$ is the max size of the subdatabase. Each machine $\mathcal{P}_k$ processes its subdatabase DB_k into the form of a subvector $\mathbf{v}_k$ as discussed in the footnote 4. The entire database DB is represented as a concatenated vector $\mathbf{v} = (\mathbf{v}_0, ..., \mathbf{v}_{M-1})$. Execute the setup algorithm of the DVC scheme and output the public parameter $\mathsf{pp} = \mathsf{DVC.Setup}(1^\lambda, N, M)$, where N is the length of the entire database-vector and M is the number of machines. pp will be implicitly input in other algorithms.
- $\mathsf{Commit}([\mathcal{P}_k : \mathbf{v}_k]_{k\in[0,M)}) \to (C_{DB})$: Machines compute the commitment to the database DB by computing $C_{DB} = \mathsf{DistCommit}([\mathcal{P}_k : \mathbf{v}_k]_{k\in[0,M)})$, and output the commitment C_{DB} publicly.
- $\mathsf{QueryProve}(d, i) \to (v^*, \Pi_{d,i})$: Upon receiving a query request for index (d, i), the server instructs the d-th machine $\mathcal{P}_d$ to return the queried value $v^* = v_{d,i}$, and generate a position proof $\Pi_{d,i} = \mathsf{DVC.Prove}(v^*, d, i, [\mathcal{P}_k : \mathbf{v}_k]_{k\in[0,M)}) \to \Pi_{d,i}$, proving that v^* is indeed the element stored at position (d, i) in the database.
- $\mathsf{Verify}((d, i), v^*, \Pi_{d,i}) \to \{0, 1\}$: The user verifies the proof $\Pi_{d,i}$ to see if v^* is indeed the element stored at position (d, i) in the database. (Run $\mathsf{DVC.Verify}(C_{DB}, d, i, v^*, \Pi_{d,i})$)
- $\mathsf{Update}(C_{DB}, v'_{u,j}, [\mathcal{P}_k : \mathbf{v}_k]_{k\in[0,M)}, [\Pi_{d,i}]_{d\in[0,M),i\in[0,n)}) \to (C'_{DB}, [\Pi'_{d,i}]_{d\in[0,M),i\in[0,n)})$: After an element $v_{u,j}$ is updated to $v'_{u,j}$, the server update the commitment and all proofs using $\mathsf{DVC.UpdComProof}(\cdot)$.

The algorithms described above form the core functionalities of a DVEDB. A well-designed DVEDB should also support efficient batch proving and the ability to aggregate individual proofs. Moreover, additive homomorphism is important in DVEDB, as it enables native, efficient updates for both commitments to the database and position proofs. Furthermore, it allows native batch verification across multiple commitments.

One can design DEVDB with additional security properties, such as key-value DEVDB and privacy-preserving DEVDB, by using the extended version of DVC proposed in Section H.

B Comparison among our work and other VC works

Table 7 shows the comparisons among the single-machine-version HLE-DVC and more other VC schemes.

C Promising DVC schemes

Existing VC schemes cannot be trivially transformed into satisfactory DVC schemes. However, equipped with the decomposition method in Section 4, MLE-based schemes can be converted into somewhat efficient DVC schemes. The schemes presented in this section is also our original contribution. *We recommend reading this section after finishing the main body of the paper. Skipping this section will not affect the overall readability.*

(1) *DVC scheme derived from Hyperproofs [3].* Hyperproofs is a VC scheme based on MLE and PST commitments. Hyperproofs-DVC can be obtained by applying the techniques in Section 4 to the original Hyperproofs. But for ease of exposition, we take a different perspective: it can also be derived from the "MLE+MLE" variant of HLE-DVC. Our HLE-DVC follows an "MLE+LDE" design, where each machine uses LDE to extend its subvector into a sub-polynomial, and then MLE is used to aggregate all sub-polynomials. If we instead choose to use MLE to extend subvectors into sub-polynomials, the resulting "MLE+MLE" variant of HLE-DVC would, in terms of both functionality and performance, be equivalent to Hyperproofs-DVC. One might raise the concern that the verification equation of Hyperproofs is expressed under PST evaluation proofs, which appears slightly different from that of HLE-DVC. However, they are essentially the same and can be converted into each other as shown in Appendix E.3.

The complexities of Hyperproofs-DVC is shown in Table 1, which achieves an $\mathrm{O}(\log M)$ communication cost and $\mathrm{O}(\frac{N}{M}\log N)$ overhead for generating all proofs. Its major advantage lies in maintainability, as all proofs can be updated with only $\mathrm{O}(N)$ cost. However, it suffers from two drawbacks: The proof size is relatively large, $\mathrm{O}(\log N)$; It does not support native aggregation and must rely on inner product arguments (IPA) for aggregation.

(2) *DVC schemes derived from Campanelli et al. [26].* [26] proposed a general VC-construction aimed at achieving memory/time trade-offs in traditional VC schemes. Its central idea can be summarized as follows. Firstly, for $k \in [0, n)$, generate the sub-commitment $C_{\mathbf{v}_k}$ to the subvector $\mathbf{v}_k$; Then commit to the sub-commitments $[C_{\mathbf{v}_k}]_{k\in[0,M)}$, and outputs the commitment $C_{\mathbf{v}}$ as the DVC commitment. The [26]'s position proof to an element $v_{d,i}$ includes three components: 1. Subvector commitment $C_{\mathbf{v}_d}$. 2. The proof that $C_{\mathbf{v}_d}$ is committed by $C_{\mathbf{v}}$. 3. The proof that $v_{d,i}$ is committed by $C_{\mathbf{v}_d}$. The VC derived

Table 7: Comparison among different vector commitment schemes. Naturally, the only two VC schemes satisfying aggregatable and maintainable are Hyperproofs and BalanceProofs. N denotes the vector length. Proof sizes and time complexities are in terms of group elements and group exponentiations/field operations, respectively. trans?(transparent) means no trusted setup; homo?(homomorphism) means additively homomorphism. The last row lists the single-machine-version of HLE-DVC.

Protocols	$\|\pi_i\|$	$\|\pi_{\mathbf{I}}\|$	ProveAll time	Agg time	UpdAllProof time	Trans ?	Homo ?	$\|pp\|$
AMT [33]	$\log N$	×	$N\log N$	×	$\log N$	×	✓	$N\log N$
aSVC [13]	1	1	$N\log N$	$\|\mathbf{I}\|\log^2\|\mathbf{I}\|$	N	×	✓	N
BBF [34]	1	1	$N\log^2 N$	$\|\mathbf{I}\|\log N$	$N\log N$	×	×	1
CF-CDH [4,7,35]	1	1	N^2	$\|\mathbf{I}\|$	N	×	✓	N^2
CF-RSA [7,35,36]	1	1	$N\log l$	$\|\mathbf{I}\|\log^2\|\mathbf{I}\|$	N	×	✓	1
CFG+RSA [36]	1	1	$N\log^2 l$	$\|\mathbf{I}\|\log\|\mathbf{I}\|\log N$	N	×	×	1
Lattice [37–39]	$\log N$	×	N	×	$\log N$	✓	✓	$\log N$
Merkle	$\log N$	×	N	×	$\log N$	✓	×	1
Merkle SNARK	$\log N$	1	N	$\|\mathbf{I}\|\log N\log(\|\mathbf{I}\|\log N)$	$\log N$	×	×	1
Pointproofs [4]	1	1	$N\log N$	$\|\mathbf{I}\|$	N	×	✓	N
Hyperproofs [3]	$\log N$	$\log(\|\mathbf{I}\|\log N)$	$N\log N$	$\|\mathbf{I}\|\log N$	$\log N$	×	✓	N
BalanceProofs [27]	1	1	$N\log N$	$\|\mathbf{I}\|\log^2\|\mathbf{I}\|$	$\sqrt{N}\log N$	×	✓	N
HLE-DVC	1	1	$N\log N$	$\|\mathbf{I}\|\log^2\|\mathbf{I}\|$	N	×	✓	N

from this idea enables a memory–time trade-off, but lacks most of the essential functionalities such as aggregation and homomorphism. So [26] adopt the PST polynomial commitment scheme [28] to construct the entire system. Its method of extending a vector into a polynomial is somewhat similar to our HLE. However, its proof additionally requires a low degree proof and a subvector commitment, which means that its verification involves checking three equations: 1. Checking if $C_{\mathbf{v}_d}$ is committed by $C_{\mathbf{v}}$. 2. Checking if $v_{d,i}$ is committed by $C_{\mathbf{v}_d}$. 3. Checking if the polynomial committed by $C_{\mathbf{v}_d}$ is a low degree polynomial.

By combining [26] with the decomposition method proposed in Section 4, it can also be transformed into a DVC scheme. However, its proof additionally requires a low-degree test and a subvector commitment and its verification involves checking three equations (whereas HLE-DVC only involves one equation), which introduce extra costs. As a result, HLE-DVC enjoys advantages in proof size, verification cost, and aggregation performance in practice.

[26]-DVC As a result, HLE-DVC enjoys advantages in proof size, verification cost, and aggregation performance in practice. (whereas our HLE requires only one). As a result, HLE-DVC enjoys advantages in proving time, proof size, verification cost, and aggregation performance. By combining [26] with the decomposition method proposed in Section 4, it can also be transformed into a DVC scheme, but its performance is strictly inferior to that of HLE-DVC in all aspects.

[26]-DVC can be constructed using the following approach: By combining it with the decomposition method proposed in Section 4, a [26]-DVC can be derived.

The advantage of this approach lies in its support for both aggregatability and additive homomorphism, with relatively low computational cost and low communication cost. However, since its verification involves checking three equations (whereas our HLE-DVC only requires one), our proposed scheme achieves benefits in both proving time, proof size, verification cost and aggregation performance.

D Preliminaries

D.1 PST polynomial commitment scheme (PCS) for multivariate polynomials

The PST PCS presented here is a non-hiding version:

Definition D.1. ***(PST polynomial commitment scheme for multivariate polynomials [28] [8]).*** *A polynomial commitment scheme for multivariate polynomials is a tuple of four protocols* $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Prove}, \mathsf{Verify})$*:*

- $\mathsf{Setup}(1^\lambda, \mathbf{D}, \rho, n) \to \mathsf{pp}_{PST}$. Take as input $\mathbf{D}$ (the max degree of the input polynomial, $\mathbf{D} = (\mathbf{D_s}, D_x)$, where $\mathbf{D_s} = (D_{s_0}, ..., D_{s_{\rho-1}})$). By default, we set $\mathbf{D_s} = \mathbf{2}^\rho = (2, ..., 2)$, a vector of length ρ filled with 2, and set $D_x = n$. Generate a bilinear pairing instance by $\mathsf{bp} = \mathsf{BilGen}(1^\lambda) = (p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, Q)$. Randomly choose the trapdoor: $(\alpha_{s_0}, ..., \alpha_{s_{\rho-1}}, \alpha_x) \xleftarrow{\$} \mathbb{F}^{\rho+1}$. Set the proving key of PST PCS as $\overline{pk_{PST}} = [(\alpha_{s_0}^{c_0} \cdot ... \cdot \alpha_{s_{\rho-1}}^{c_{\rho-1}} \cdot \alpha_x^i)P]_{c_0 \in \{0,1\}, ..., c_{\rho-1} \in \{0,1\}, i \in [0,n)}$. Set the verification key as $\overline{vk_{PST}} = \{\alpha_{s_0}Q, ..., \alpha_{s_{\rho-1}}Q, \alpha_x Q\}$. Output $\mathsf{pp}_{PST} = (\rho, M, \mathbf{D}, \mathsf{bp}, \overline{pk_{PST}}, \overline{vk_{PST}})$. pp_{PST} will be implicitly provided as input to the remaining algorithms.
- $\mathsf{Commit}(\mathcal{F}(\mathbf{s}, x)) \to C$. Take as input a polynomial $\mathcal{F}(\mathbf{s}, x)$. Compute the PST commitment $C = \mathcal{F}(\alpha_{s_0}, ..., \alpha_{s_{\rho-1}}, \alpha_x)P$ using the proving key $\overline{pk_{PST}}$. For simplicity, $\mathsf{PST.Commit}(\mathcal{F}(\mathbf{s}, x))$ is abbreviated as $\mathsf{PST}(\mathcal{F}(\mathbf{s}, x))$.
- $\mathsf{Prove}(v, \bar{\tau}, \mathcal{F}) \to \Pi$. Given a fixed point $\bar{\tau} = (\tau_{s_0}, ..., \tau_{s_{\rho-1}}, \tau_x)$, $\mathcal{P}$ generates a proof Π, proving that $\mathcal{F}(\bar{\tau}) = v$. Specifically, $\mathcal{P}$ computes polynomials $[\mathsf{q}_c(\mathbf{s}, x)]_{c \in [0, \rho+1)}$, s.t.

$\mathcal{F}(\mathbf{s},x)-v=\sum_{c=0}^{\rho-1}(s_c-\tau_{s_c})\mathfrak{q}_c(\mathbf{s},x)+(x-\tau_x)\mathfrak{q}_\rho(\mathbf{s},x)$. $\mathcal{P}$ sets $[\pi_c=\mathsf{PST}(\mathfrak{q}_c(\mathbf{s},x))]_{c\in[0,\rho+1)}$. $\mathcal{P}$ outputs an evaluation proof $\Pi=[\pi_c]_{c\in[0,\rho+1)}$.

• $\mathsf{Verify}(C,\overline{\tau},v,\Pi)\rightarrow b\in\{0,1\}$. $\mathcal{V}$ checks if $e(C-vP,Q)\stackrel{?}{=}\prod_{c=0}^{\rho-1}e(\pi_c,(\alpha_{s_c}-\tau_{s_c})Q)e(\pi_\rho,(\alpha_x-\tau_x)Q)$. $\mathcal{V}$ outputs 1 if the verification passes or 0 otherwise.

D.2 Inner Product Arguments (IPA)

CM is a commitment scheme proposed by Abe et al. [40] for vectors $\mathbf{A}\in\mathbb{G}_1^\rho,\mathbf{B}\in\mathbb{G}_2^\rho$ and the inner pairing product $Z=\mathsf{IP}(\mathbf{A},\mathbf{B})=\prod_{i=0}^{\rho-1}e(A_i,B_i)$. The commitment key of CM is set as $\mathbf{ck}=(\mathbf{V}\stackrel{\$}{\leftarrow}\mathbb{G}_2^\rho,\mathbf{W}\stackrel{\$}{\leftarrow}\mathbb{G}_1^\rho)$. $\mathbf{A},\mathbf{B}$ and Z are committed as: $\mathbf{C}\stackrel{\text{def}}{=}(C_0,C_1,C_2)=\mathsf{CM}(\mathbf{ck};\mathbf{A},\mathbf{B},Z)=(\mathsf{IP}(\mathbf{A},\mathbf{V}),\mathsf{IP}(\mathbf{W},\mathbf{B}),Z)$. Bünz et al. [29] give a non-interactive IPA for the relation: the public problem instance statement $(\mathbf{ck},\mathbf{C}\in\mathbb{G}_T)$ and the witness $(\mathbf{A}\in\mathbb{G}_1^\rho,\mathbf{B}\in\mathbb{G}_2^\rho)$ have the following relation: $\mathcal{R}_{IPA}^\rho=\{(\mathbf{ck},\mathbf{C}\in\mathbb{G}_T;\mathbf{A}\in\mathbb{G}_1^\rho,\mathbf{B}\in\mathbb{G}_2^\rho):\mathbf{C}=\mathsf{CM}(\mathbf{ck};\mathbf{A},\mathbf{B},\mathsf{IP}(\mathbf{A},\mathbf{B}))\}$

We abstract the argument for $\mathcal{R}_{IPA}$ as three algorithms:

• $\mathsf{IPA.Setup}(1^\lambda,\rho)\rightarrow\mathsf{pp}_{IPA}$. Output $\mathsf{pp}_{IPA}=\{\mathsf{BilGen}(1^\lambda),\mathbf{ck}=(\mathbf{V}\stackrel{\$}{\leftarrow}\mathbb{G}_2^\rho,\mathbf{W}\stackrel{\$}{\leftarrow}\mathbb{G}_1^\rho)\}$.

• $\mathsf{IPA.Prove}(\mathsf{pp}_{IPA};\mathbf{A},\mathbf{B})\rightarrow\pi_{IPA}$. Output a proof π_{IPA}, proving that $\mathbf{C}=(\mathbf{A},\mathbf{B},\mathsf{IP}(\mathbf{A},\mathbf{B}))$.

• $\mathsf{IPA.Verify}(\mathsf{pp}_{IPA},\mathbf{C},\pi_{IPA})\rightarrow\{0,1\}$. Verify the proof π_{IPA} that $\mathbf{C}=(\mathbf{A},\mathbf{B},\mathsf{IP}(\mathbf{A},\mathbf{B}))$.

IPA complexity. IPA.Prove takes $O(\rho)$ time, IPA.Verify takes $O(\log\rho)$ time and the proof size is $|\pi_{IPA}|=O(\log\rho)$ (the whole protocol can be seen in [29]).

E Security analysis

E.1 DVC definition

Definition E.1. ***(Distributed vector commitment scheme).*** *A distributed vector commitment scheme consists of the following* PPT *algorithms* DVC = (Setup, DistCommit, Prove, Verify, BatchProve, BatchVerify, UpdComProof, Agg.Prove, Agg.Verify)*. The vector to be committed are split into M subvectors $\{\mathbf{v}_k|k\in[0,M)\}$, and each machine $\mathcal{P}_k$ holds $\mathbf{v}_k$. Machines distributedly generate a commitment $C_\mathbf{v}$ and position proofs.*

• $\mathsf{Setup}(1^\lambda,N,M)\rightarrow\mathsf{pp}$. Given the max length of the vector length N, the number of machines M and the security parameter λ as input. Output public parameters pp, which will be implicitly provided as input to the remaining algorithms.

• $\mathsf{DistCommit}([\mathcal{P}_k:\mathbf{v}_k]_{k\in[0,M)})\rightarrow C_\mathbf{v}$. Machines distributedly generate an HLE commitment $C_\mathbf{v}$ and position proofs.

• $\mathsf{Prove}(v^*,d,i,[\mathcal{P}_k:\mathbf{v}_k]_{k\in[0,M)})\rightarrow\Pi_{d,i}$. Output a proof $\Pi_{d,i}$ to prove that $v_{d,i}=v^*$.

• $\mathsf{Verify}(C_\mathbf{v},d,i,v^*,\Pi_{d,i})\rightarrow\{0,1\}$. $\mathcal{V}$ checks the proof $\Pi_{d,i}$ to verify if $v_{d,i}=v^*$.

• $\mathsf{BatchProve}(d,\mathbf{I},\mathbf{v}_\mathbf{I}^*,[\mathcal{P}_k:\mathbf{v}_k]_{k\in[0,M)})\rightarrow\Pi_\mathbf{I}$. Given an index set $\mathbf{I}$, output a proof $\Pi_\mathbf{I}$ to prove that $\mathbf{v}_\mathbf{I}^*$ is the $\mathbf{I}$-subvector of $\mathbf{v}_d$.

• $\mathsf{BatchVerify}(C_\mathbf{v},d,\mathbf{I},\mathbf{v}_\mathbf{I}^*,\Pi_\mathbf{I})\rightarrow\{0,1\}$. $\mathcal{V}$ checks the proof $\Pi_\mathbf{I}$ to verify if $\mathbf{v}_\mathbf{I}^*$ is the $\mathbf{I}$-subvector of $\mathbf{v}_d$.

• $\mathsf{UpdComProof}(C_\mathbf{v},v'_{u,j},[\mathcal{P}_k:\mathbf{v}_k]_{k\in[0,M)},[\Pi_{d,i}]_{d\in[0,M),i\in[0,n)})\rightarrow(C'_\mathbf{v},[\Pi'_{d,i}]_{d\in[0,M),i\in[0,n)})$. After an element $v_{u,j}$ is updated to $v'_{u,j}$, update the commitment and all proofs.

• $\mathsf{Agg.Prove}(\mathbf{I},\mathbf{v}_\mathbf{I}^*,[\Pi_{d_j,i_j}]_{(d_j,i_j)\in\mathbf{I}})\rightarrow\Pi_{Agg,\mathbf{I}}$. Given $|\mathbf{I}|$ individual proofs for the index set $\mathbf{I}$, output the aggregated proof $\Pi_{Agg,\mathbf{I}}$.

• $\mathsf{Agg.Verify}(C_\mathbf{v},\mathbf{I},\mathbf{v}_\mathbf{I}^*,\Pi_{Agg,\mathbf{I}})\rightarrow\{0,1\}$. $\mathcal{V}$ checks the proof $\Pi_{Agg,\mathbf{I}}$ to verify if $\mathbf{v}_\mathbf{I}^*$ is the $\mathbf{I}$-subvector of $\mathbf{v}_d$.

E.2 Preliminaries

Assumption E.1. *(q-SDH) [3]. For any* PPT *adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c}\mathsf{bp}\leftarrow\mathsf{BilGen}(1^\lambda)=(p,e,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,P,Q);\\ \alpha\stackrel{\$}{\leftarrow}\mathbb{F};\\ \mathsf{pp}=((p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,e,P,Q),\alpha Q,\alpha P,\ldots,\alpha^q P):\\ (a,\frac{1}{\alpha+a}P)\leftarrow\mathcal{A}(1^\lambda,\mathsf{pp})\end{array}\right]<\mathsf{negl}(\lambda)$$

Assumption E.2. *(q-BSDH) [8]. For any* PPT *adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c}\mathsf{bp}\leftarrow\mathsf{BilGen}(1^\lambda)=(p,e,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,P,Q);\\ \alpha\stackrel{\$}{\leftarrow}\mathbb{F};\\ \mathsf{pp}=((p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,e,P,Q),\alpha Q,...,\alpha^q Q,\alpha P,\ldots,\alpha^q P):\\ (a,e(P,Q)^{\frac{1}{\alpha+a}}P)\leftarrow\mathcal{A}(1^\lambda,\mathsf{pp})\end{array}\right]<\mathsf{negl}(\lambda)$$

Lemma E.1. ***(Multivariate polynomial remainder lemma** [28]). $\mathcal{F}(\mathbf{s})$ is a multivariate polynomial. $\mathcal{F}(\tau_\mathbf{s})=v$, if and only if there exist ρ polynomials, $[\mathfrak{t}_c(\mathbf{s})]_{c\in[0,\rho)}$, s.t.*

$$\mathcal{F}(\mathbf{s})-v=\sum_{c=0}^{\rho-1}(s_c-\tau_{s_c})\cdot\mathfrak{t}_c(\mathbf{s})$$

Lemma E.2. ***(Univariate polynomial batch remainder lemma** [13]). Given a vector $\mathbf{v}=v_0,...,v_{n-1}$ of length n, and its LDE $\mathfrak{f}(x)$, s.t. for all $i\in[0,n)$, $\mathfrak{f}(\omega_n^i)=v_i$. Given an index set $I=[i_j]_{j\in[0,|I|-1]}$. Let $A_I(x)=\prod_{i\in I}(x-\omega_n^i)$. Given another vector $\mathbf{v}_I$ of length $|I|-1$. Let $\mathfrak{r}(x)$ be the LDE of $\mathbf{v}_I$, s.t. $j\in[0,|I|-1)$, $\mathfrak{r}(\omega_n^{i_j})=v_{I,j}$. The vector $\mathbf{v}_I$ is the I-subvector of $\mathbf{v}$, if and only if there exists a witness polynomial $\mathfrak{q}(x)$, s.t. $\mathfrak{f}(x)=\mathfrak{q}(x)A_I(x)+\mathfrak{r}(x)$.*

E.3 Security proof

Theorem E.1. *HLE-DVC scheme is a secure vector commitment scheme satisfying computational binding. Moreover, the individual position proof, batch proof and the aggregation proof satisfies correctness and soundness if the PST PCS is computational binding, and the q-SDH problem and q-BSDH problem hold, and the IPA scheme is secure.*

PROOF:

1. ***Computational binding***. The HLE commitment is essentially a PST commitment to the HLE polynomial $\mathcal{F}(\mathbf{s},x)$, and therefore satisfies the properties of *computational binding*.

2. ***Correctness***. The correctness of the position proof has been proved in lemma 4.3.

The correctness of aggregating proof relies on the correctness of the IPA protocol. The correctness of batch proof relies on the lemma E.2.

3. ***Individual soundness***: Firstly, we note that each HLE position proof for $v_{d,i}=v$ can be converted to a PST evaluation proof for $\mathcal{F}(\langle d\rangle,i)=v$.

The conversion can be implemented as follows: given a position proof for $v_{d,i}$, i.e. $\Pi_{d,i}=\{[\pi_{d,c}]_{c\in[0,\rho)},\pi_{d,\rho,i}\}$, s.t.

$$e(C_{\mathbf{v}}-v\cdot\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle d\rangle)),Q)=\prod_{c=0}^{\rho-1}e(\pi_{d,c},(\alpha_{s_c}-\langle d\rangle_c)Q)\cdot e(\pi_{d,\rho,i},(\alpha_x-\omega_n^i)Q)$$

Then, since $\mathsf{eq}(\langle d\rangle,\langle d\rangle)=1$, according to lemma E.1, we can compute ρ polynomials, $[\mathfrak{t}_{d,c}(\mathbf{s})]_{c\in[0,\rho)}$, s.t.

$$\mathsf{eq}(\mathbf{s},\langle d\rangle)-1=\sum_{c=0}^{\rho-1}(s_c-\langle d\rangle_c)\cdot\mathfrak{t}_{d,c}(\mathbf{s})$$

Let $[W_{d,c}=\pi_{d,c}+\mathsf{PST}(\mathfrak{t}_{d,c}(\mathbf{s}))]_{c\in[0,\rho)}$, so that $e(C_{\mathbf{v}}-vP,Q)=\prod_{c=0}^{\rho-1}e(W_{d,c},(\alpha_{s_c}-\langle d\rangle_c)Q)\cdot e(\pi_{d,\rho,i},(\alpha_x-\omega_n^i)Q)$

Thus $\Pi_{\mathsf{PST},d,i}=\{[W_{d,c}]_{c\in[0,\rho)},\pi_{d,\rho,i}\}$ is a valid evaluation proof for $\mathcal{F}(\langle d\rangle,\omega_n^i)=v^*$.

Next, we can prove the soundness of HLE-DVC's position proofs using a method similar to that for the evaluation proof of PST-PCS.

Then we will give a formal proof as follows. Suppose there exists an adversary $\mathcal{A}$ that breaks definition **??** of soundness. We show how to build another adversary $\mathcal{B}$ that breaks the $(\rho+n)$-SDH assumption (see assumption E.1).

$\mathcal{B}$ is given $(\rho+n)$-SDH public parameters $\mathrm{pp}_{SDH}=((p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,e,P,Q),\alpha Q,\alpha P,\ldots,\alpha^{\rho+n}P)$, and wants to break $(\rho+n)$-SDH by outputting $(a,\frac{1}{\alpha-a}P)$ for some $a\neq\alpha$. For this, $\mathcal{B}$ will leverage $\mathcal{A}$ to help him.

First, $\mathcal{B}$ "guesses" the position (d,i) on which $\mathcal{A}$ will forge, which he can do with probability $1/\mathrm{poly}(\lambda)$, where λ is our security parameter.

Second, $\mathcal{B}$ "tweaks" the $(\rho+n)$-SDH public parameters into the public parameters, which he then calls $\mathcal{A}$ with. Specifically, $\mathcal{B}$ sets $\alpha_{s_c}-\langle d\rangle_c=r_c(\alpha-\langle d\rangle_0),\forall c\in[0,\rho)$, and $\alpha_x-\omega_n^i=r_\rho(\alpha-\langle d\rangle_0)$, where $r_0=1$, the rest of the r_k's are random. Importantly, note that $\mathcal{B}$ can do this without knowledge of s, since $\mathcal{B}$ can compute the partial proof in the PST PCS, i.e. $\overline{\psi_{PST}}=[(\alpha_{s_0}^{c_0}\cdot\ldots\cdot\alpha_{s_{\rho-1}}^{c_{\rho-1}}\cdot\alpha_x^i)P]_{c_0\in\{0,1\},\ldots,c_{\rho-1}\in\{0,1\},i\in[0,n)}$ from the $(\alpha P,\ldots,\alpha^{\rho+n}P)$. Similarly, $\mathcal{B}$ can compute the verification key of PST as $\overline{vk_{PST}}=\{\alpha_{s_0}Q,\ldots,\alpha_{s_{\rho-1}}Q,\alpha_xQ\}$ from αQ. Then $\mathcal{B}$ can generate the public parameters in the HLE-DVC scheme using parameters of the PST PCS.

Third, $\mathcal{B}$ calls $\mathcal{A}$ with the "tweaked" public parameters as input and obtains a commitment $C_{\mathbf{v}}$ and two inconsistent position proofs $\Pi_{d,i}=\{[\pi_{d,c}]_{c\in[0,\rho)},\pi_{d,\rho,i}\}$ for $v_{d,i}=v$, and $\Pi'_{d,i}=\{[\pi'_{d,c}]_{c\in[0,\rho)},\pi'_{d,\rho,i}\}$ for $v_{d,i}=v'$. (If $\mathcal{A}$ outputs proofs for a different position $i'\neq i$, $\mathcal{B}$ retries.) Via the conversion method proposed before, we can convert these two position proofs to two PST evaluation proofs, $\Pi_{\mathsf{PST},d,i}=\{[W_{d,c}]_{c\in[0,\rho)},\pi_{d,\rho,i}\}$, for $\mathcal{F}(\langle d\rangle,\omega_n^i)=v$, and $\Pi'_{\mathsf{PST},d,i}=\{[W'_{d,c}]_{c\in[0,\rho)},\pi'_{d,\rho,i}\}$, for $\mathcal{F}(\langle d\rangle,\omega_n^i)=v'$.

These two PST evaluation proofs satisfy that: $e(C_{\mathbf{v}}-vP,Q)=\prod_{c=0}^{\rho-1}e(W_{d,c},(\alpha_{s_c}-\langle d\rangle_c)Q)\cdot e(\pi_{d,\rho,i},(\alpha_x-\omega_n^i)Q)$

$e(C_{\mathbf{v}}-v'P,Q)=\prod_{c=0}^{\rho-1}e(W'_{d,c},(\alpha_{s_c}-\langle d\rangle_c)Q)\cdot e(\pi'_{d,\rho,i},(\alpha_x-\omega_n^i)Q)$

Next, dividing the top equation by the bottom one and substitute $\alpha_{s_c}-\langle d\rangle_c=r_c(\alpha-\langle d\rangle_0),\forall c\in[0,\rho)$ and $\alpha_x-\omega_n^i=r_\rho(\alpha-\langle d\rangle_0)$:

$e((v'-v)P,Q)=\prod_{c=0}^{\rho-1}e(W_{d,c}-W'_{d,c},(\alpha_{s_c}-\langle d\rangle_c)Q)\cdot e(\pi_{d,\rho,i}-\pi'_{d,\rho,i},(\alpha_x-\omega_n^i)Q)$

The following equations are equivalent each other:

$$e((v'-v)P,Q)=\prod_{c=0}^{\rho-1}e(W_{d,c}-W'_{d,c},r_c(\alpha-\langle d\rangle_0)Q)\cdot e(\pi_{d,\rho,i}-\pi'_{d,\rho,i},r_\rho(\alpha-\langle d\rangle_0)Q) \quad (10)$$

$$e(P,Q)^{v'-v}=(\prod_{c=0}^{\rho-1}e(W_{d,c}-W'_{d,c},r_cQ)\cdot e(\pi_{d,\rho,i}-\pi'_{d,\rho,i},r_\rho Q))^{\alpha-\langle d\rangle_0} \quad (11)$$

$$e(P,Q)^{\frac{1}{\alpha-\langle d\rangle_0}}=(\prod_{c=0}^{\rho-1}e(W_{d,c}-W'_{d,c},r_cQ)\cdot e(\pi_{d,\rho,i}-\pi'_{d,\rho,i},r_\rho Q))^{\frac{1}{v'-v}} \quad (12)$$

$$e(\frac{1}{\alpha-\langle d\rangle_0}P,Q)=e(\prod_{c=0}^{\rho-1}\frac{r_c}{v'-v}(W_{d,c}-W'_{d,c}),Q)\cdot e(\frac{r_\rho}{v'-v}(\pi_{d,\rho,i}-\pi'_{d,\rho,i}),Q) \quad (13)$$

$$e(\frac{1}{\alpha-\langle d\rangle_0}P,Q)=e(\prod_{c=0}^{\rho-1}\frac{r_c}{v'-v}(W_{d,c}-W'_{d,c})+\frac{r_\rho}{v'-v}(\pi_{d,\rho,i}-\pi'_{d,\rho,i}),Q) \quad (14)$$

Thus, $\frac{1}{\alpha-\langle d\rangle_0}P=\prod_{c=0}^{\rho-1}\frac{r_c}{v'-v}(W_{d,c}-W'_{d,c})+\frac{r_\rho}{v'-v}(\pi_{d,\rho,i}-\pi'_{d,\rho,i})$ and $\mathcal{B}$ can output $(\langle d\rangle_0,\frac{1}{\alpha-\langle d\rangle_0}P)$ and break $(\rho+n)-$SDH.

4. **Batch soundness**. The proof of batch soundness closely follows that of individual soundness.

Recall that a valid batch proof $\Pi_{\mathbf{I}}$ for an index set $\mathbf{I}=[(d,i_j)]_{j\in[0,|\mathbf{I}|-1)}$, is $\Pi_{\mathbf{I}}=(\pi_{d,0},\ldots,\pi_{d,\rho-1},\pi_{d,\rho,I})$. Denote the set of all i_j involved in $\mathbf{I}$ as $I=[i_j]_{j\in[0,|\mathbf{I}|-1),(d,i_j)\in\mathbf{I}}$. Let $A_I(x)=\prod_{i\in I}(x-\omega_n^i)$ and extend $\mathbf{v}_{\mathbf{I}}^*$ to a univariate polynomial $\mathfrak{r}(x)$, s.t. for all $i\in I$, $\mathfrak{r}(\omega_n^i)=v_{d,i}$.

A valid proof $\Pi_{\mathbf{I}}$ satisfies that $e(C_{\mathbf{v}} - \mathsf{PST}(\mathsf{eq}(\mathbf{s}, \langle d \rangle) \cdot \mathfrak{r}(x)), Q) \stackrel{?}{=} \prod_{c=0}^{\rho-1} e(\pi_{d,c}, (\alpha_{s_c} - \langle d \rangle_c)Q) \cdot e(\pi_{d,\rho,I}, A_I(\alpha_x)Q)$,

Similarly, an HLE-DVC batch position proof $\Pi_{\mathbf{I}} = \{[\pi_{d,c}]_{c\in[0,\rho)}, \pi_{d,\rho,i}\}$, proving that $\mathbf{v}_{\mathbf{I}}$ is a $\mathbf{I}$-subvector of $\mathbf{v}_{\mathbf{I}}$, can be converted to a PST-like batch evaluation proof for $[\mathcal{F}(\langle d \rangle, \omega_n^{i_j}) = v_{d,i_j}]_{j\in[0,|\mathbf{I}|-1)}$ as follows:

Since $\mathsf{eq}(\langle d \rangle, \langle d \rangle) = 1$, we can compute ρ polynomials (according to lemma E.1), $[\mathfrak{t}_{d,c}(\mathbf{s})]_{c\in[0,\rho)}$, s.t.

$$\mathsf{eq}(\mathbf{s}, \langle d \rangle) - 1 = \sum_{c=0}^{\rho-1} (s_c - \langle d \rangle_c) \cdot \mathfrak{t}_{d,c}(\mathbf{s})$$

$$\mathfrak{r}(x) \cdot \mathsf{eq}(\mathbf{s}, \langle d \rangle) - \mathfrak{r}(x) = \sum_{c=0}^{\rho-1} (s_c - \langle d \rangle_c) \cdot \mathfrak{r}(x) \cdot \mathfrak{t}_{d,c}(\mathbf{s})$$

Let $[W_{d,c} = \pi_{d,c} + \mathsf{PST}(\mathfrak{r}(x) \cdot \mathfrak{t}_{d,c}(\mathbf{s}))]_{c\in[0,\rho)}$, so that

$$e(C_{\mathbf{v}} - \mathsf{PST}(\mathfrak{r}(x)), Q) = \prod_{c=0}^{\rho-1} e(W_{d,c}, (\alpha_{s_c} - \langle d \rangle_c)Q) \cdot e(\pi_{d,\rho,I}, A_I(\alpha_x)Q)$$

Thus $\Pi_{\mathsf{PST},d,I} = \{[W_{d,c}]_{c\in[0,\rho)}, \pi_{d,\rho,I}\}$ is a valid evaluation proof for $[\mathcal{F}(\langle d \rangle, \omega_n^{i_j}) = v_{d,i_j}]_{j\in[0,|\mathbf{I}|-1),(d,i_j)\in\mathbf{I}}$.

Similarly, we can implement the security proof as follows: Suppose there exists an adversary $\mathcal{A}$ that breaks definition **??** of batch soundness. We show how to build another adversary $\mathcal{B}$ that breaks the $(\rho+n)$-BSDH assumption (see assumption E.2).

$\mathcal{B}$ is given $(\rho+n)$-BSDH public parameters $\mathrm{pp}_{BSDH} = ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, Q), \alpha Q, ..., \alpha^{\rho+n}Q, \alpha P, \ldots, \alpha^{\rho+n}P)$, and wants to break $\rho+n$-BSDH by outputting $(a, \frac{1}{\alpha-a}P)$ for some $a \neq \alpha$. For this, $\mathcal{B}$ will leverage $\mathcal{A}$ to help him.

First, $\mathcal{B}$ "guesses" the position (d,i) on which $\mathcal{A}$ will forge, which he can do with probability $1/\mathrm{poly}(\lambda)$, where λ is our security parameter.

Second, $\mathcal{B}$ "tweaks" the $(\rho+n)$-BSDH public parameters into the public parameters, which he then calls $\mathcal{A}$ with. Specifically, $\mathcal{B}$ sets $\alpha_{s_c} - \langle d \rangle_c = r_c(\alpha - \langle d \rangle_0), \forall c \in [0,\rho)$, and $\alpha_x - \omega_n^i = r_\rho(\alpha - \langle d \rangle_0)$, where $r_0 = 1$, the rest of the r_k's are random. Importantly, note that $\mathcal{B}$ can do this without knowledge of s, since $\mathcal{B}$ can compute the partial proof in the PST PCS, i.e. $\overline{\Psi_{PST}} = [(\alpha_{s_0}^{c_0} \cdot \ldots \cdot \alpha_{s_{\rho-1}}^{c_{\rho-1}} \cdot \alpha_x^i)P]_{c_0\in\{0,1\},\ldots,c_{\rho-1}\in\{0,1\},i\in[0,n)}$ from the $(\alpha P, \ldots, \alpha^{\rho+n}P)$. Similarly, $\mathcal{B}$ can compute the verification key of PST as $\overline{vk_{PST}} = \{\alpha_{s_0}Q, ..., \alpha_{s_{\rho-1}}Q, \alpha_x Q\}$ from $(\alpha Q, ..., \alpha^{\rho+n}Q)$. Then $\mathcal{B}$ can generate the public parameters in the HLE-DVC scheme using parameters of the PST PCS.

Third, $\mathcal{B}$ calls $\mathcal{A}$ with the "tweaked" public parameters as input and obtains $(C_{\mathbf{v}}, \mathbf{I}, \mathbf{I}', \mathbf{v}_{\mathbf{I}}, \mathbf{v}'_{\mathbf{I}}, \Pi_{\mathbf{I}}, \Pi'_{\mathbf{I}})$, including a commitment $C_{\mathbf{v}}$ and two inconsistent batch position proofs $\Pi_{\mathbf{I}} = \{[\pi_{d,c}]_{c\in[0,\rho)}, \pi_{d,\rho,I}\}$ for the index set $\mathbf{I}$, and $\Pi'_{\mathbf{I}} = \{[\pi'_{d,c}]_{c\in[0,\rho)}, \pi'_{d,\rho,I}\}$ for the index set $\mathbf{I}'$, and the index set $\mathbf{I}$ and $\mathbf{I}'$ are only related to a single subvector $\mathbf{v}_d$. Specially, $(d,i) \in \mathbf{I} \cap \mathbf{I}'$ and $\Pi_{\mathbf{I}}$ proves that $v_{d,i} = v$, but $\Pi'_{\mathbf{I}}$ proves that $v_{d,i} = v'$. (If $(d,i) \notin \mathbf{I}$ or $(d,i) \notin \mathbf{I}'$ or $v = v'$, $\mathcal{B}$ retries.) Via the conversion method proposed before, we can convert these two position proofs to two PST evaluation proofs, $\Pi_{\mathsf{PST},d,I} = \{[W_{d,c}]_{c\in[0,\rho)}, \pi_{d,\rho,I}\}$, for $[\mathcal{F}(\langle d \rangle, \omega_n^{i_j}) = v_{d,i_j}]_{j\in[0,|\mathbf{I}|-1)}$, and $\Pi'_{\mathsf{PST},d,I'} = \{[W'_{d,c}]_{c\in[0,\rho)}, \pi'_{d,\rho,I'}\}$, for $[\mathcal{F}(\langle d \rangle, \omega_n^{i_j}) = v_{d,i_j}]_{j\in[0,|\mathbf{I}'|-1)}$.

These two PST evaluation proofs satisfy that:

$$e(C_{\mathbf{v}} - \mathsf{PST}(\mathfrak{r}(x)), Q) = \prod_{c=0}^{\rho-1} e(W_{d,c}, (\alpha_{s_c} - \langle d \rangle_c)Q) \cdot e(\pi_{d,\rho,I}, A_I(\alpha_x)Q)$$

$$e(C_{\mathbf{v}} - \mathsf{PST}(\mathfrak{r}'(x)), Q) = \prod_{c=0}^{\rho-1} e(W'_{d,c}, (\alpha_{s_c} - \langle d \rangle_c)Q) \cdot e(\pi'_{d,\rho,I'}, A_{I'}(\alpha_x)Q)$$

Next, dividing the top equation by the bottom one and substitute $\alpha_{s_c} - \langle d \rangle_c = r_c(\alpha - \langle d \rangle_0), \forall c \in [0,\rho)$ and $\alpha_x - \omega_n^i = r_\rho(\alpha - \langle d \rangle_0)$, the following equations are equivalent each other:

$e(\mathfrak{r}'(\alpha_x)P - \mathfrak{r}(\alpha_x)P, Q) = \prod_{c=0}^{\rho-1} e(W_{d,c} - W'_{d,c}, (\alpha_{s_c} - \langle d \rangle_c)Q) \cdot \frac{e(\pi_{d,\rho,I}, A_I(\alpha_x)Q)}{e(\pi'_{d,\rho,I'}, A'_I(\alpha_x)Q)}$

Since $\mathfrak{r}(\omega_n^i) = v$ and $\mathfrak{r}'(\omega_n^i) = v'$, We can rewrite $\mathfrak{r}(x)$ using the polynomial remainder theorem as $\mathfrak{r}(x) = q^*_{d,i}(x)(x - \omega_n^i) + v$. Similarly, $\mathfrak{r}'(x) = \mathfrak{q}'^*_{d,i}(x)(x - \omega_n^i) + v'$.

$$e(q^*_{d,i}(\alpha_x)P - \mathfrak{q}'^*_{d,i}(\alpha_x)P, Q)^{\alpha_x - \omega_n^i} \cdot e(P,Q)^{v-v'} = \prod_{c=0}^{\rho-1} e(W_{d,c} - W'_{d,c}, (\alpha_{s_c} - \langle d \rangle_c)Q) \cdot \frac{e(\pi_{d,\rho,I}, A_I(\alpha_x)Q)}{e(\pi'_{d,\rho,I'}, A'_I(\alpha_x)Q)}$$

Let $a_I(x) = \frac{A_I(x)}{x - \omega_n^i}$ and $a'_I(x) = \frac{A'_I(x)}{x - \omega_n^i}$, then:

$$\frac{e(\pi_{d,\rho,I}, A_I(\alpha_x)Q)}{e(\pi'_{d,\rho,I'}, A'_I(\alpha_x)Q)} = \left(\frac{e(\pi_{d,\rho,I}, a_I(\alpha_x)Q)}{e(\pi'_{d,\rho,I'}, a'_I(\alpha_x)Q)}\right)^{(\alpha_x - \omega_n^i)}$$

Considering that $\alpha_{s_c} - \langle d \rangle_c = r_c(\alpha - \langle d \rangle_0)$ and $\alpha_x - \omega_n^i = r_\rho(\alpha - \langle d \rangle_0)$, then:

$$e(P,Q)^{\frac{1}{\alpha - \langle d \rangle_0}} = \prod_{c=0}^{\rho-1} e(W_{d,c} - W'_{d,c}, Q)^{\frac{r_c}{v-v'}} \cdot \left(\frac{e(\pi_{d,\rho,I}, a_I(\alpha_x)Q)}{e(q^*_{d,i}(\alpha_x)P - \mathfrak{q}'^*_{d,i}(\alpha_x)P, Q) \cdot e(\pi'_{d,\rho,I'}, a'_I(\alpha_x)Q)}\right)^{\frac{\rho}{v-v'}}$$

Thus, $\mathcal{B}$ can output $(\langle d \rangle_0, \prod_{c=0}^{\rho-1} e(W_{d,c} - W'_{d,c}, Q)^{\frac{r_c}{v-v'}} \cdot \left(\frac{e(\pi_{d,\rho,I}, a_I(\alpha_x)Q)}{e(q^*_{d,i}(\alpha_x)P - \mathfrak{q}'^*_{d,i}(\alpha_x)P, Q) \cdot e(\pi'_{d,\rho,I'}, a'_I(\alpha_x)Q)}\right)^{\frac{\rho}{v-v'}})$ and break $(\rho+n)-$BSDH.

5. **Aggregation proof soundness** (cross-subvectors). Our aggregated proofs from 5.5 are sound if the IPA scheme is secure (a zero knowledge argument for the relation $\mathcal{R}_{IPA}$ is secure if it satisfies correctness, soundness, zero-knowledge and extractable. The definition for these properties can be found in [3]).

Suppose there exists $\mathcal{A}$ that outputs $(C_{\mathbf{v}}, \mathbf{I}, \mathbf{I}', \mathbf{v}_{\mathbf{I}}, \mathbf{v}'_{\mathbf{I}}, \Pi_{\mathbf{I}}, \Pi'_{\mathbf{I}})$, including a commitment $C_{\mathbf{v}}$ and two

inconsistent batch position proofs $\Pi_{\mathbf{I}} = \{[\pi_{d,c}]_{c\in[0,\rho)}, \pi_{d,\rho,I}\}$ for the index set $\mathbf{I}$, and $\Pi'_{\mathbf{I}} = \{[\pi'_{d,c}]_{c\in[0,\rho)}, \pi'_{d,\rho,I}\}$ for the index set $\mathbf{I}'$. Specially, $(d,i) \in \mathbf{I}\cap\mathbf{I}'$ and $\Pi_{\mathbf{I}}$ proves that $v_{d,i} = v$, but $\Pi'_{\mathbf{I}}$ proves that $v_{d,i} = v'$. Since the IPA scheme is extractable, there exist extractors that extract the individual $\rho+1$-sized proofs. And there exist $\Pi_{d,i} = \{[\pi_{d,c}]_{c\in[0,\rho)}, \pi_{d,\rho,i}\}$ for $v_{d,i} = v$ (extracted from $\Pi_{\mathbf{I}}$) and $\Pi'_{d,i} = \{[\pi'_{d,c}]_{c\in[0,\rho)}, \pi'_{d,\rho,i}\}$ for $v_{d,i} = v'$ (extracted from $\Pi'_{\mathbf{I}}$).

By the same argument for the soundness of an individual position, this breaks $(\rho+n)$-SDH.

□

F Update the commitment and all proofs

When $\mathcal{P}_u$ updates his vector $\mathbf{v}_u$ to $\mathbf{v}'_u$ by updating $v'_{u,j} = v_{u,j} + \delta$, the HLE commitment need to be updated to $C'_{\mathbf{v}}$. All partial proofs and position proofs need to be updated correspondingly. The update algorithm is formalized as follow:

- $\mathsf{UpdComProof}(C_{\mathbf{v}}, \overline{upk_d}, [\overline{\Psi_d}]_{d\in[0,M)}, \overline{\eta_u}, v'_{u,j}, [\Pi_{d,i}]_{d\in[0,M),i\in[0,n)}) \to (C'_{\mathbf{v}}, [\overline{\Psi_d}']_{d\in[0,M)}, [\Pi'_{d,i}]_{d\in[0,M),i\in[0,n)})$.

This algorithm involves three phases: **(1)** $\mathcal{P}_u$ updates his vector $\mathbf{v}_u$ to $\mathbf{v}'_u$ by updating $v'_{u,j} = v_{u,j} + \delta$, updates the HLE commitment $C_{\mathbf{v}}$ to $C'_{\mathbf{v}}$, and updates his auxdata η_u to η'_u. **(2)** Update all partial proofs. **(3)** Update all position proofs.

∘ **Phase 1:** $\mathcal{P}_u$ updates $\mathbf{v}_u$ to $\mathbf{v}'_u$ by updating $v'_{u,j} = v_{u,j} + \delta$. Correspondingly, the sub-HLE polynomial is updated by $\mathcal{F}'_u(\mathbf{s},x) = \mathcal{F}_u(\mathbf{s},x) + \delta\cdot\mathsf{eq}(\mathbf{s},\langle u\rangle)\mathcal{L}_j(x)$. Similarly, the HLE is updated by $\mathcal{F}'(\mathbf{s},x) = \mathcal{F}(\mathbf{s},x) + \delta\cdot\mathsf{eq}(\mathbf{s},\langle u\rangle)\mathcal{L}_j(x)$. Thus the new commitment $C'_{\mathbf{v}}$ can be computed via:

$$\begin{aligned} C'_{\mathbf{v}} &= \mathsf{PST}(\mathcal{F}'(\mathbf{s},x)) = \mathsf{PST}(\mathcal{F}(\mathbf{s},x) + \delta\cdot\mathsf{eq}(\mathbf{s},\langle u\rangle)\mathcal{L}_j(x)) \\ &= \mathsf{PST}(\mathcal{F}(\mathbf{s},x)) + \delta\cdot\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\mathcal{L}_j(x)) \end{aligned}$$

Note that $\overline{\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\mathcal{L}_j(x))}$ has been included in the update key $\overline{upk_u}$, thus the update for the commitment costs $\mathrm{O}(1)$.

∘ **Phase 2:** Update all partial proofs $[\overline{\Psi_d}]_{d\in[0,M)}$.

$\mathcal{P}_u$ updates his auxdata $\overline{\eta_u}' = \{\eta'_{u,0},\ldots,\eta'_{u,\rho-1}\}$, where

$$\begin{aligned} \eta'_{u,c} &= \mathsf{PST}\left(\frac{\mathcal{F}'_u(\mathbf{s},x)}{s_c - (1-\langle u\rangle_c)}\right) = \mathsf{PST}\left(\frac{\mathcal{F}_u(\mathbf{s},x) + \delta\cdot\mathsf{eq}(\mathbf{s},\langle u\rangle)\mathcal{L}_j(x)}{s_c - (1-\langle u\rangle_c)}\right) \\ &= \mathsf{PST}\left(\frac{\mathcal{F}_u(\mathbf{s},x)}{s_c - (1-\langle u\rangle_c)}\right) + \delta\cdot\mathsf{PST}\left(\frac{\mathsf{eq}(\mathbf{s},\langle u\rangle)\mathcal{L}_j(x)}{s_c - (1-\langle u\rangle_c)}\right) \end{aligned}$$

Note that $\mathsf{PST}(\frac{\mathcal{F}_u(\mathbf{s},x)}{s_c-(1-\langle u\rangle_c)})$ is the old auxdata $\eta_{u,c}$, and $\mathsf{PST}(\frac{\mathsf{eq}(\mathbf{s},\langle u\rangle)\mathcal{L}_j(x)}{s_c-(1-\langle u\rangle_c)})$ has been provided in the update key $\overline{upk_u}$. So the update for auxdata $\overline{\eta_u}'$ costs $\mathrm{O}(\log M)$ time.

Then the master node $\mathcal{P}_0$ needs to update the ρ HLE-Aux trees $[\mathtt{T}_c]_{c\in[0,\rho)}$: For $c\in[0,\rho)$, update $\mathtt{T}_c.\mathtt{Node}'_{b_0\ldots b_c} = \mathtt{T}_c.\mathtt{Node}_{c,b_0\ldots b_c} - \eta_{u,c} + \eta'_{u,c}$, where $b_0 = \langle u\rangle_0,\ldots,b_c = \langle u\rangle_c$. Thus c HLE-Aux trees are updated in $\mathrm{O}(\log M)$ time.

$\mathcal{P}_u$'s partial proof remains unchanged. Other machines' partial proof can be updated according to the new HLE-Aux trees without repeating computation.

∘ **Phase 3:** update all position proofs.

Step 1: Machine $\mathcal{P}_d$ ($d\neq u$) updates the proofs to $[\Pi'_{d,i} = \{\overline{\Psi_d}', \pi_{d,\rho,i}\}]_{i\in[0,n)}$ by updating his partial proof.

Step 2: Machine $\mathcal{P}_u$ updates the proofs $[\Pi'_{u,i} = \{\overline{\Psi_u}, \pi'_{u,\rho,i}\}]_{i\in[0,n)}$: If $i=j$, $\pi'_{u,\rho,j}$ can be computed by: $\pi'_{u,\rho,j} = \mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{\mathfrak{f}'_u(x)-v'_{u,j}}{x-\omega_n^j}) = \mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{\mathfrak{f}_u(x)+\delta\cdot\mathcal{L}_j(x)-v_{u,j}-\delta}{x-\omega_n^j}) = \pi_{u,\rho,j} + \delta\cdot\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{\mathcal{L}_j(x)-1}{x-\omega_n^j})$. Since $\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{\mathcal{L}_j(x)-1}{x-\omega_n^j})$ has been included in the update key $\overline{upk_u}$, this computation costs $\mathrm{O}(1)$.

If $i\neq j$, $\pi'_{u,\rho,i}$ can be computed by: $\pi'_{u,\rho,i} = \mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{\mathfrak{f}'_u(x)-v_{u,i}}{x-\omega_n^i}) = \pi_{u,\rho,i} + \delta\cdot\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{\mathcal{L}_j(x)}{x-\omega_n^i})$, where $\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{\mathcal{L}_j(x)}{x-\omega_n^i})$ can be computed with $\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{\mathcal{L}_j(x)}{x-\omega_n^j})$ and $\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{\mathcal{L}_i(x)}{x-\omega_n^i})$ in $\mathrm{O}(1)$ time [5] (method from [13]).

In summary, the vector commitment can be updated in $\mathrm{O}(1)$ time and all proofs can be updated in $\mathrm{O}(\frac{N}{M}+\log M)$ time with $\mathrm{O}(1)$ communication cost per machine.

G Aggregating proof

G.1 Preliminary: partial fraction decomposition [13]

Denote the set of all i_j involved in $\mathbf{I}$ as $I = [i_j]_{j\in[0,|\mathbf{I}|-1),(d,i_j)\in\mathbf{I}}$. Recall that $A_I(x) = \prod_{i\in I}(x-\omega_n^i)$. Partial fraction decomposition lemma shows that $\frac{1}{A_I(x)} = \frac{1}{\prod_{i\in I}(x-\omega_n^i)} = \sum_{i\in[0,M)} c_i\cdot\frac{1}{x-\omega_n^i}$

G.2 Single-subvector proof aggregation

Given an index set $\mathbf{I} = [(d_j,i_j)]_{j\in[0,|\mathbf{I}|-1)}$, and a claimed $\mathbf{I}$-subvector $\mathbf{v}^*_{\mathbf{I}} = [v^*_{d,i_j}]_{(d,i_j)\in\mathbf{I}}$, one can efficiently aggregate several individual proofs $[\Pi_{d_j,i_j}]_{j\in[0,|\mathbf{I}|-1)}$ (the j-th proof Π_{d_j,i_j} is a proof for $v_{d_j,i_j} = v^*_j$) into a single, succinct proof $\Pi_{\mathbf{I}}$.

The partial fraction decomposition can be used to aggregate position proofs. Denote the set of all i_j involved in $\mathbf{I}$ as $I = [i_j]_{(d,i_j)\in\mathbf{I}}$. Recall that $A_I(x) = \prod_{i\in I}(x-\omega_n^i)$. Let $A'_I(x) = \sum_{j\in I} A_I(x)/(x-\omega_n^j)$ be the derivative of $A_I(x)$. Extend $\mathbf{v}^*_{\mathbf{I}}$ to a univariate polynomial $\mathfrak{r}(x)$, s.t. for all $i\in I$, $\mathfrak{r}(\omega_n^i) = v^*_{d,i}$. Let $\mathfrak{q}^*_{d,I}(x) = \frac{\mathfrak{f}_d(x)-\mathfrak{r}(x)}{A_I(x)}$ and for $i\in I$, $\mathfrak{q}^*_{d,i}(x) = \frac{\mathfrak{f}_d(x)-v^*_{d,i}}{x-\omega_n^i}$.

[5] $\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{\mathcal{L}_j(x)}{x-\omega_n^j})$ can be computed with $\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{\mathcal{L}_j(x)}{x-\omega_n^j})$ and $\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{\mathcal{L}_i(x)}{x-\omega_n^i})$ in $\mathrm{O}(1)$ time as follows:

Denote $A(x) = \prod_{i=0}^{n-1}(x-\omega^i) = x^n-1$ and let $A'(x)$ be the derivative of $A(x)$. Compute $W_{u,i,j} = \frac{\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{A(x)}{x-\omega_n^i})}{\omega_n^i-\omega_n^j} + \frac{\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{A(x)}{x-\omega_n^j})}{\omega_n^j-\omega_n^i}$ Then $\mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle u\rangle)\frac{\mathcal{L}_j(x)}{x-\omega_n^i}) = \frac{W_{u,i,j}}{A'(\omega_n^i)}$. The correctness of this method can be seen in aSVC [13].

According to the partial fraction decomposition, we have:

$$\begin{aligned}
\mathfrak{q}^*_{d,I}(x) &= \mathfrak{f}_d(x)\frac{1}{A_I(x)} - \mathfrak{r}(x)\frac{1}{A_I(x)} \\
&= \mathfrak{f}_d(x)\sum_{i\in I}\frac{1}{A'_I(\omega^i)(x-\omega^i)} - (A_I(x)\sum_{i\in I}\frac{v_i}{A'_I(\omega^i)(x-\omega^i)})\cdot\frac{1}{A_I(x)} \\
&= \sum_{i\in I}\frac{\mathfrak{f}_d(x)}{A'_I(\omega^i)(x-\omega^i)} - \sum_{i\in I}\frac{v_i}{A'_I(\omega^i)(x-\omega^i)} \\
&= \sum_{i\in I}\frac{1}{A'_I(\omega^i)}\cdot\frac{\mathfrak{f}_d(x)-v_i}{x-\omega^i} = \sum_{i\in I}\frac{1}{A'_I(\omega^i)}\cdot\mathfrak{q}^*_{d,i}(x)
\end{aligned}$$

Let $\pi_{d,\rho,I} = \mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle d\rangle)\mathfrak{q}^*_{d,I}(x))$ and for $i\in I$, $\pi_{d,\rho,i} = \mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle d\rangle)\mathfrak{q}^*_{d,i})$. Finally, $\pi_{d,\rho,I} = \prod_{i\in I}\frac{1}{A'_I(\omega^i)}\cdot\pi_{d,\rho,i}$.

Thus the single-subvector aggregation algorithms can be implemented as follows:

- $\mathsf{Agg.Prove}(\mathbf{I},\mathbf{v}^*_{\mathbf{I}},[\Pi_{d_j,i_j}]_{(d_j,i_j)\in\mathbf{I}}) \to \Pi_{Agg,\mathbf{I}}$. For $i\in I$, parse the position proofs as $\Pi_{d,i} = (\pi_{d,0},...,\pi_{d,\rho-1},\pi_{d,\rho,i})$. One can aggregate all proofs by $\pi_{d,\rho,I} = \prod_{i\in I}\frac{\pi_{d,\rho,i}}{A'_I(\omega^i_n)}$. Finally, the single-subvector aggregation proof is set as $\Pi_{Agg,\mathbf{I}} = (\pi_{d,0},...,\pi_{d,\rho-1},\pi_{d,\rho,I})$

Essentially, $\Pi_{Agg,\mathbf{I}}$ is equivalent to the output of the batch proof for $\mathbf{I}$. Thus the verification for $\Pi_{Agg,\mathbf{I}}$ is the same as the verification for batch proof.

- $\mathsf{Agg.Verify}(C_{\mathbf{v}},\mathbf{I},\mathbf{v}^*_{\mathbf{I}},\Pi_{Agg,\mathbf{I}}) \to \{0,1\}$. Run $b\in\{0,1\} \leftarrow \mathsf{BatchVerify}(C_{\mathbf{v}},d,\mathbf{I},\mathbf{v}^*_{\mathbf{I}},\Pi_{Agg,\mathbf{I}},\overline{vk_d})$. Output b.

G.3 Cross-subvectors proof aggregation

Given an index set $\mathbf{I} = [(d_j,i_j)]_{j\in[0,|\mathbf{I}|-1)}$, a claimed subvector $\mathbf{v}^*_{\mathbf{I}} = [v^*_{d_j,i_j}]_{(d_j,i_j)\in\mathbf{I}}$, a set of corresponding individual position proofs $[\Pi_{d_j,i_j} = (\psi_{d_j},\beta_{d_j,i_j})]_{(d_j,i_j)\in\mathbf{I}}$, The individual proofs can be aggregated via a two-stage aggregation process (w.l.o.g., we assume that the set $\mathbf{I}$ involves all M subvectors.):

1. The proofs to be aggregated are first grouped according to their associated subvector indices. $\mathbf{I}$ is partitioned into $\mathbf{I} = \mathbf{I}_0\cup\mathbf{I}_1\cup\cdots\cup\mathbf{I}_{M-1}$ according to the subvector index each position belongs to, where for all $d\in[0,M)$, $\mathbf{I}_d = [(d,i_j)]_{(d,i_j)\in\mathbf{I}}$. Correspondingly, we have $\mathbf{v}^*_{\mathbf{I}} = \mathbf{v}^*_{\mathbf{I}_0}\cup\mathbf{v}^*_{\mathbf{I}_1}\cup\cdots\cup\mathbf{v}^*_{\mathbf{I}_{M-1}}$. For each $d\in[0,M)$, we run the single-subvector aggregation algorithm $\Pi_{Agg,\mathbf{I}_d} \leftarrow \mathsf{Agg.Prove}(\mathbf{I}_d,\mathbf{v}^*_{\mathbf{I}_d},[\Pi_{d,i_j}]_{(d,i_j)\in\mathbf{I}_d})$.

2. These M batch proofs $[\Pi_{Agg,\mathbf{I}_d}]_{d\in[0,M)}$ can be further aggregated into a single aggregation proof, $\Pi_{Agg,\mathbf{I}}$ of size $\mathrm{O}(\log M)$ using the IPA protocol [29]:

For $d\in[0,M)$, we set:

$$\begin{aligned}
Z_d &= e(C_{\mathbf{v}} - \mathsf{PST}(\mathsf{eq}(\mathbf{s},\langle d\rangle)\cdot\mathfrak{r}_d(x)),Q) \\
\mathbf{B}_d &= ((\alpha_{s_0}-\langle d\rangle_0)Q,...,(\alpha_{s_{\rho-1}}-\langle d\rangle_{\rho-1})Q,A_{I_d}(\alpha_X)Q) \\
\mathbf{A}_d &= \Pi_{Agg,\mathbf{I}_d} = (\pi_{d,0},...,\pi_{d,\rho-1},\pi_{d,\rho,I_d})
\end{aligned}$$

where $I_d = [i_j]_{(d,i_j)\in\mathbf{I}_d}$, $A_{I_d}(X) = \prod_{i\in I_d}(X-\omega^i_n)$, $\mathfrak{r}_d(X)$ is the univariate LDE of $\mathbf{v}^*_{I_d}$. Considering the batch position proof verification equation, all batch proofs can pass the verification if for all $d\in[0,M)$: $Z_d = \prod_{c=0}^{\rho}\cdot e(A_{d,c},B_{d,c})$ Moreover, we can introduce a random linear combination to combine these M equations into a single one:

$$\prod_{d=0}^{M-1} r_d Z_d = \prod_{d=0}^{M-1} r_d \prod_{c=0}^{\rho}\cdot e(A_{d,c},B_{d,c})$$

This equation is similar to the inner product argument of length $M(\log M+1)$. We give a precise description of its algorithms as follows: ($\mathbf{I},\mathbf{v}^*_{\mathbf{I}}$, and $[\Pi_{d_j,i_j}]_{(d_j,i_j)\in\mathbf{I}}$ have been processed to $[\mathbf{A}_d,\mathbf{B}_d,Z_d]_{d\in[0,M)}$)

- $\mathsf{Agg.Setup}(1^\lambda,(\rho+1)\cdot M) \to \mathsf{pp}_{Agg}$. Output $\mathsf{pp}_{Agg} = (\mathsf{BilGen}(1^\lambda),\mathbf{ck} = (\mathbf{V}\xleftarrow{\$}\mathbb{G}_2^{(\rho+1)\cdot M},\mathbf{W}\xleftarrow{\$}\mathbb{G}_1^{(\rho+1)\cdot M}))$.
- $\mathsf{Agg.Prove}(\mathsf{pp}_{Agg},\mathbf{I},[\mathbf{A}_d,\mathbf{B}_d]_{d\in[0,M)}) \to \Pi_{Agg,\mathbf{I}}$. Let $\mathbf{A} = (\mathbf{A}_0,...,\mathbf{A}_{M-1})$ and $\mathbf{B} = (\mathbf{B}_0,...,\mathbf{B}_{M-1})$. Let $C_1 = \mathsf{IP}(\mathbf{A},\mathbf{V})$. For $d\in[0,M)$, compute $r_d = Hash(C_1,\mathbf{B},d)$. Let $\mathbf{B}' = (r_0\mathbf{B}_0,...,r_{M-1}\mathbf{B}_{M-1})$. Computes $\pi_{IPA} = \mathsf{IPA.Prove}(\mathbf{ck},\mathbf{A},\mathbf{B}')$ and returns $\Pi_{Agg,\mathbf{I}} = (C_1,\pi_{IPA})$.
- $\mathsf{Agg.Verify}(\mathsf{pp}_{Agg},[Z_d,\mathbf{B}_d]_{d\in[0,M)},\Pi_{Agg,\mathbf{I}}) \to \{0,1\}$. Parse the proof $\Pi_{Agg,\mathbf{I}} = (C_1,\pi_{IPA})$. Let $\mathbf{B} = (\mathbf{B}_0,...,\mathbf{B}_{M-1})$ and compute $r_d = Hash(C_1,\mathbf{B},d)$, for $d\in[0,M)$. Let $\mathbf{B}' = (r_0\mathbf{B}_0,...,r_{M-1}\mathbf{B}_{M-1})$ and $Z' = \prod_{d=0}^{M-1} r_d\cdot Z_d$. Let $C_2 = \mathsf{IP}(\mathbf{W},\mathbf{B}')$. Let $\mathbf{C} = (C_1,C_2,Z')$ and return $\mathsf{IPA.verify}(\mathbf{ck},\mathbf{C},\pi_{IPA})$.

IPA complexity. IPA.Prove takes $\mathrm{O}(\rho)$ time, IPA.Verify takes $\mathrm{O}(\log\rho)$ time and the proof size is $|\pi_{IPA}| = \mathrm{O}(\log\rho)$ (the whole protocol can be seen in [29]).

Cross-subvectors aggregation complexity. Agg.Prove takes $\mathrm{O}(|\mathbf{I}|\log^2(|\mathbf{I}|)+M\log M)$ time, Agg.Verify takes $\mathrm{O}(|\mathbf{I}|\log^2(|\mathbf{I}|)+\log(M\log M))$ time and the proof size is $|\pi_{IPA}| = \mathrm{O}(\log(M\log M))$.

H Extensions

H.1 "LDE+LDE" version HLE-DVC

HLE utilize the "LDE + MLE" paradigm to construct the DVC scheme. If we choose to use the "LDE + LDE" paradigm to construct the DVC scheme, (namely using LDE to extend the subvector to sub-polynomials, and also use LDE to aggregate all sub-polynomials, result in a bivariate polynomial to represent the vector) the "LDE + LDE" version DVC can succinct proof size ($\mathrm{O}(1)$), and $\mathrm{O}(1)$ verification time, with the cost of updating all proofs increasing to $\mathrm{O}(\log N)$. Its overhead complexities are summarized in Table 1.

H.2 How to achieve maintainability and hiding/zero-knowledge

Maintainability. Naturally, the HLE-DVC scheme is aggregatable. However, it is not maintainable because updating all proofs incurs an overhead of $\mathrm{O}(\frac{N}{M}+\log M)$. To achieve maintainability, we present the following two methods:

1. *Using "MLE + MLE" paradigm*: The "MLE+MLE" version HLE-DVC can achieve maintainability with the update cost, $O(\log N)$.

2. *Using BalanceProofs.* The BalanceProofs [27] proposed a compiler that takes as input any naturally aggregatable vector commitment that is not maintainable, and produces another naturally-aggregatable and maintainable vector commitment. By compiling HLE-DVC with the compiler in BalanceProofs, we can also achieve maintainability in $O(\sqrt{\frac{N}{M}}\log\frac{N}{M}+\log M)$ update-all time.

Hiding/zero-knowledge. In the main text, we choose to instantiate HLE-DVC with a non-hiding version PST PCS by default. The hiding property can be achieved by using the hiding version PST PCS.

Furthermore, since the evaluation proofs of the PST PCS possess the zero-knowledge property, and the position proof in HLE-DVC is equivalent to the evaluation proof of the PST PCS, the hiding version of HLE-DVC also inherits the zero-knowledge property.

H.3 How to design a key-value-DVC (KDVC) from HLE-DVC

Key-value vector commitment (KVC) [41] is a type of vector commitment scheme that supports key-value maps. There are two approaches to designing a KDVC scheme based on HLE-DVC:

1. The MLE is based on a binary tree structure. Intuitively, a natural idea is to extend the binary MLE to q-ary Verkle trees [42], thereby constructing a KDVC scheme. Specifically, by adopting an "LDE/MLE + Verkle-MLE" approach, where the subvector is represented by a Verkle-trees, enabling support for key-value maps.

2. Leveraging the 256-bit width of finite field elements to encode both the key and the value: the higher-order bits can be used to encode the key, and the remaining bits to store the value (just like in stateless blockchain).