FlexProofs: A Vector Commitment with Flexible Linear Time for Computing All Proofs

Jing Liu and Liang Feng Zhang(✉)

ShanghaiTech University, Shanghai, China
{liujing1,zhanglf}@shanghaitech.edu.cn

Abstract. In this paper, we introduce FlexProofs, a new *vector commitment (VC)* scheme that achieves two key properties: (1) the prover can generate all individual opening proofs for a vector of size N in optimal time $\mathcal{O}(N)$, and there is a flexible batch size parameter b that can be increased to further reduce the time to generate all proofs; and (2) the scheme is directly compatible with a family of zkSNARKs that encode their input as a multi-linear polynomial. As a critical building block, we propose the first *functional commitment (FC)* scheme for multi-exponentiations with batch opening. Compared with HydraProofs, the only existing VC scheme that computes all proofs in optimal time $\mathcal{O}(N)$ and is directly compatible with zkSNARKs, FlexProofs may speed up the process of generating all proofs, if the parameter b is properly chosen. Our experiments show that for $N = 2^{16}$ and $b = \log^2 N$, FlexProofs can be 6× faster than HydraProofs. Moreover, when combined with suitable zkSNARKs, FlexProofs enable practical applications such as verifiable secret sharing and verifiable robust aggregation.

Keywords: Vector Commitment · Functional Commitment · zkSNARKs.

1 Introduction

Zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) [36,30] allow a prover to convince a verifier with input $\mathbf{m}$ that $\mathbf{y}$ is the output of a computation $\mathcal{C}(\mathbf{m}, \mathbf{w})$, where $\mathcal{C}$ is typically represented as an arithmetic circuit and $\mathbf{w}$ may be a private input held by the prover. While zkSNARKs have been widely used to secure real-world applications [45,56,46,1,31] and are well known for enabling efficient verification by end-users, it is less clear how to apply them in a *multi-user* setting where N participants $\{\mathcal{V}_i\}_{i=0}^{N-1}$, each holding a private input m_i, delegate the computation of $\mathcal{C}((m_0, \ldots, m_{N-1}), \mathbf{w})$ to a service provider. Such multi-user settings underlie numerous applications such as distributed machine learning [50], crowdsourcing [32], secret sharing [61], and collaborative filtering [57], where it may be crucial to address the risk of a dishonest service provider [47,24]. The traditional zkSNARKs are not directly suitable for this setting because they assume each verifier knows the entire input $\mathbf{m} = (m_0, \ldots, m_{N-1})$, which however is not true in the multi-user setting.

Recently, Pappas, Papadopoulos, and Papamanthou [54] proposed a method of adapting zkSNARKs to the multi-user setting by combining them with *vector*

commitment (VC) [17]. In their solution, the prover uses a VC scheme to commit to the input vector $\mathbf{m} = (m_0, \ldots, m_{N-1})$ and runs a zkSNARK for $\mathcal{C}(\mathbf{m}, \mathbf{w})$ (similar to commit-and-prove zkSNARKs [25,13]). Since VCs support openings of the vector at chosen indices, the prover can additionally provide each verifier $\mathcal{V}_i$ with a proof that its input m_i is indeed the i-th element of the committed vector used in the zkSNARK, ensuring both computation correctness and individual data inclusion. This scenario requires a VC scheme that can *efficiently generate all proofs* and is *directly compatible with zkSNARKs*.

Among the existing VC schemes, the Merkle trees [51] support generating all proofs in $\mathcal{O}(N)$ time, but are inefficient when integrated into zkSNARKs. Some modern VC schemes, e.g., those based on elliptic curves [62,67,65,28,44] or error-correcting codes [74,76], are directly compatible with many zkSNARKs [71,59,27], yet require $\mathcal{O}(N^2)$ time to naively produce all proofs. Some constructions [62,65,44,74,76] may reduce the time cost to $\mathcal{O}(N \log N)$. To the best of our knowledge, HydraProofs [54] is the only existing VC scheme that can generate all opening proofs in $\mathcal{O}(N)$ time and is directly compatible with zkSNARKs based on multi-linear polynomials [71,59,19]. In particular, the $\mathcal{O}(N)$ time cost is incurred by $\mathcal{O}(N)$ field operations and $\mathcal{O}(N)$ cryptographic operations (e.g., group exponentiations or pairings). In this paper, we propose FlexProofs, a new VC scheme that preserves zkSNARK compatibility and generates all proofs using only $\mathcal{O}(N)$ field operations and $\mathcal{O}(N/b + \sqrt{N} \log N)$ heavy cryptographic operations, where b is the batch size (ranging from 1 to $\sqrt{N}$). Compared with HydraProofs, FlexProofs requires fewer heavy cryptographic operations for larger b, enabling more efficient generation of all proofs.

1.1 Our Contributions

Functional Commitment Scheme for Multi-Exponentiations with Batch Opening. To construct FlexProofs, we propose the first *functional commitment (FC)* scheme for multi-exponentiations with batch opening. In the scheme, the prover commits to a vector and later generates a batch proof for multiple computations over the vector; any verifier can check computation results against the commitment. The scheme achieves constant-size commitments and logarithmic-size proofs, and is proven secure in the algebraic group model (AGM) and the random oracle model (ROM) under the n-ASDBP and q-SDH assumptions; our implementation confirms the high efficiency of batch opening.

FlexProofs. We introduce FlexProofs, a new *vector commitment (VC)* scheme that achieves two key properties. First, the prover can generate all proofs for a vector of size N in optimal time $\mathcal{O}(N)$, which is incurred by $\mathcal{O}(N)$ field operations and $\mathcal{O}(N/b + \sqrt{N} \log N)$ cryptographic operations, where batch size b ranges from 1 to $\sqrt{N}$. Second, the scheme is directly compatible with a family of zkSNARKs that encode their input as a multi-linear polynomial [71,59,19]. FlexProofs builds on the proposed FC scheme and an existing polynomial commitment (PC) scheme, and is proven correct and secure (see below for the detailed design rationale). Compared with HydraProofs, the only existing VC scheme

that computes all proofs in time $\mathcal{O}(N)$ (incurred by $\mathcal{O}(N)$ field operations and $\mathcal{O}(N)$ cryptographic operations) and is directly compatible with zkSNARKs, FlexProofs requires fewer cryptographic operations for larger b, enabling more efficient proof generation. Our experiments show that for $N = 2^{16}$ and $b = \log^2 N$, FlexProofs generates all proofs about $6\times$ faster than HydraProofs, while achieving similar verification time and slightly larger proofs. Finally, combining our VC scheme with suitable zkSNARKs enables practical applications such as verifiable secret sharing and verifiable robust aggregation.

Design Rationale for FlexProofs. FlexProofs adopts a clear two-layer structure: each sub-vector is first committed using an existing PC scheme, and the resulting PC commitments are then committed with our proposed FC scheme. Correspondingly, the proof generation process also naturally splits into an FC layer and a PC layer, which allow us to do optimization independently: the FC layer employs our batch-opening FC scheme to lower the cost, while the PC layer applies ideas similar to HydraProofs to further reduce the cost. Overall, the two-layer structure combining FC and PC, together with our batch-opening FC scheme, enables FlexProofs to outperform purely PC-based HydraProofs.

1.2 Applications

Verifiable Secret Sharing. A real-world application involving multiple users' data is *verifiable secret sharing (VSS)*. At a high level, a secret sharing scheme [61] allows a dealer to split a secret value s into N shares such that any $t+1$ of them can reconstruct s, while any subset of at most t reveals nothing about it. Verifiability protects the receivers of shares against a dishonest dealer who may issue malformed or inconsistent shares [24,21]. The combination of VCs and zkSNARKs yields a VSS scheme where each receiver can verify that all shares form a valid sharing of s and that its own share is consistent with this sharing.

Verifiable Robust Aggregation. Another application is *federated learning (FL)* [4,50,37], where multiple clients train local models and send gradients to an *aggregator*, who combines them into a global model. This process is repeated iteratively until the model converges. Due to its decentralized nature, FL is vulnerable to misbehaving clients that submit poisoned or low-quality gradients, motivating a long line of work on *robust aggregation* to mitigate the impact of such adversarial inputs [8,34,15,49,26]. However, the security of robust aggregation relies on an honest aggregator, and these guarantees collapse when the aggregator is untrusted. In practice, aggregators often have strong incentives to misuse their power for personal benefit. For instance, in federated recommendation systems [73,63], an aggregator can tamper with the model to promote its products, influence markets, or push its political agendas [23,64,33]; in FL-as-a-Service [38], it may delay convergence for monetary gain [72]. To safeguard FL against both a misbehaving aggregator and malicious clients, Pappas et al. [54] introduce the notion of *verifiable robust aggregation (VRA)*. And by applying the combination of VCs and zkSNARKs to existing robust aggregation algorithms such as FLTrust [15], one can obtain such a VRA scheme.

1.3 Related Work

Functional Commitments. Existing FC schemes target polynomials [18,69], linear functions [41,39,48,14,22], monotone span programs [18], or semi-sparse polynomials [43]. Prior FC schemes for Boolean or arithmetic circuits cannot handle multi-exponentiation $\prod_{i=0}^{n-1} \mathbf{A}[i]^{\mathbf{b}[i]} (\mathbf{A} \in \mathbb{G}_1^n, \mathbf{b} \in \mathbb{F}_p^n)$, where $\mathbb{G}_1$ is the source group of a bilinear group. Some schemes fail because they cannot commit to vectors over $\mathbb{G}_1$: for instance, [55,16] commit to vectors over $\mathbb{Z}_q$; [3] commits to vectors over a commutative ring; [70] commits to vectors over finite rings; and [11] commits to arithmetic circuits in a field. Other schemes fail for a different reason: FC schemes [69,68,60] targeting Boolean functions $f : \{0,1\}^l \to \{0,1\}$ operate over bitstrings and fixed Boolean predicates, which cannot represent or efficiently compute exponentiations over $\mathbb{G}_1$.

Vector Commitments. Merkle trees [51] support generating all proofs in $\mathcal{O}(N)$ time. However, they are not well suited for zkSNARKs, since incorporating this VC pre-image into existing zkSNARKs would essentially require reconstructing the entire tree within the zkSNARK arithmetic circuit, which is highly inefficient even with SNARK-friendly hash functions [29]. Some modern VC schemes, such as those based on elliptic curves [62,67,65,28,44] or error-correcting codes [74,76], are directly compatible with many zkSNARKs [71,59,27] since they use the same data encoding. However, unlike Merkle trees, generating all N proofs is inefficient: the naive cost is $\mathcal{O}(N^2)$, and even improved constructions [62,65,44,74,76] require $\mathcal{O}(N \log N)$ time. To the best of our knowledge, HydraProofs [54] is the only existing VC scheme that can generate all proofs in $\mathcal{O}(N)$ time while being directly compatible with zkSNARKs based on multi-linear polynomials [71,59,19]. We do not consider lattice-based VC techniques [53,55,69], as the corresponding lattice-based zkSNARKs remain largely impractical.

2 Preliminaries

Notation. For any integer $n > 0$, let $[0, n) = \{0, 1, \ldots, n-1\}$. Let $\mathbf{m}$ be a vector of length n. For any $i \in [0, n)$, we denote by $\mathbf{m}[i]$ the i-th element of $\mathbf{m}$. For any $I \subseteq [0, n)$, we denote $\mathbf{m}[I] = (\mathbf{m}[i])_{i \in I}$. Besides, we denote

$$\mathbf{m}_L = (\mathbf{m}[0], \ldots, \mathbf{m}[n/2 - 1]), \quad \mathbf{m}_R = (\mathbf{m}[n/2], \ldots, \mathbf{m}[n-1]),$$

We denote by $\mathbb{G}$ a multiplicative group, and by $\mathbb{F}_p$ the finite field of prime order p. For a vector $\mathbf{A} \in \mathbb{G}^n$, a vector $\mathbf{b} \in \mathbb{F}_p^n$ and a scalar $x \in \mathbb{F}_p$, we denote

$$\mathbf{A}^x = (\mathbf{A}[0]^x, \ldots, \mathbf{A}[n-1]^x), x\mathbf{b} = (x\mathbf{b}[0], \ldots, x\mathbf{b}[n-1]), \langle \mathbf{A}, \mathbf{b} \rangle = \prod_{i \in [0,n)} \mathbf{A}[i]^{\mathbf{b}[i]}.$$

Here, $\langle \mathbf{A}, \mathbf{b} \rangle$ compactly denotes the multi-exponentiation of $\mathbf{A}$ by $\mathbf{b}$, which our FC scheme is designed to support. For two vectors $\mathbf{A}, \mathbf{A}' \in \mathbb{G}^n$, we denote $\mathbf{A} \circ \mathbf{A}' = (\mathbf{A}[0]\mathbf{A}'[0], \ldots, \mathbf{A}[n-1]\mathbf{A}'[n-1])$. Let $\mathbf{u}_i$ be the i-th unit vector, with

1 at the i-th position and zeros elsewhere. We define $\mathsf{Bin}(i) = (i_{\ell-1}, \ldots, i_0)$ as the *bit decomposition* of an ℓ-bit integer i if $i = \sum_{k\in[0,\ell)} i_k 2^k$. Any vector $\mathbf{m} \in \mathbb{F}_p^n$ can be encoded as a multilinear polynomial $f_{\mathbf{m}} : \mathbb{F}_p^{\log n} \to \mathbb{F}_p$ as below:

$$f_{\mathbf{m}}(\mathbf{x}) = \sum_{i\in[0,n)} \mathbf{m}[i] \prod_{k\in[0,\log n)} (i_k x_k + (1-i_k)(1-x_k)), \tag{1}$$

where $\mathbf{x} = (x_{\log n-1}, \ldots, x_0)$, $\mathsf{Bin}(i) = (i_{\log n-1}, \ldots, i_0)$. We refer to $f_{\mathbf{m}}$ as the *multi-linear extension* of $\mathbf{m}$. For any finite set S, we denote by $s \leftarrow S$ the process of choosing s uniformly from S. We denote by $y \leftarrow \mathsf{Alg}(x)$ the process of running an algorithm Alg on an input x and assigning the output to y. We say that a function $\epsilon(\lambda)$ is *negligible* in λ and denote $\epsilon(\lambda) = \mathsf{negl}(\lambda)$, if $\epsilon(\lambda) = o(\lambda^{-c})$ for all $c > 0$. Our security proofs are in the *random oracle model* (ROM), formalized in [5]: we model a cryptographic hash function as a truly random function, accessible to all parties only via oracle queries. Specifically, we use two random oracles $H, H' : \{0,1\}^* \to \mathbb{F}_p$.

Bilinear Group. We denote by $\mathsf{BG}(1^\lambda)$ a bilinear group generator that takes a security parameter λ as input and outputs a bilinear group context $\mathbf{bg} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e, g_1, g_2)$, where $\mathbb{G}_1 = \langle g_1 \rangle, \mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_\mathrm{T}$ are groups of prime order p, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathrm{T}$ is a *pairing* such that $e(u^\alpha, v^\beta) = e(u, v)^{\alpha\beta}$ for all $u \in \mathbb{G}_1, v \in \mathbb{G}_2$ and $\alpha, \beta \in \mathbb{Z}_p$. We assume *Type-3* bilinear groups where *no* efficiently computable homomorphisms exist between $\mathbb{G}_1$ and $\mathbb{G}_2$. For any $\mathbf{A} \in \mathbb{G}_1^n, \mathbf{B} \in \mathbb{G}_2^n$, we denote $\mathbf{A} * \mathbf{B} = \prod_{i\in[0,n)} e(\mathbf{A}[i], \mathbf{B}[i])$.

2.1 Functional Commitments with Batch Openings

Libert et al. [41] introduced a *functional commitment* (FC) model for linear functions over finite fields. Multi-exponentiations over bilinear groups naturally extends these linear functions: for $\mathbf{A} \in \mathbb{G}_1^n$ and $\mathbf{b} \in \mathbb{F}_p^n$, $\langle \mathbf{A}, \mathbf{b} \rangle = \prod_{i=0}^{n-1} \mathbf{A}[i]^{\mathbf{b}[i]}$. We extend the FC model of [41] to support multi-exponentiations and batched evaluations on multiple field vectors. An FC scheme $\mathsf{FC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{BOpen}, \mathsf{BVerify})$ in our model consists of four algorithms:

- $\mathsf{FC.Setup}(1^\lambda, 1^n) \to \mathsf{pp}$*: Given the security parameter λ and the vector size n, outputs public parameters* pp*, an implicit input to all remaining algorithms.*
- $\mathsf{FC.Commit}(\mathbf{A}) \to C$*: Given a vector $\mathbf{A} \in \mathbb{G}_1^n$, outputs a commitment C.*
- $\mathsf{FC.BOpen}(C, \mathbf{A}, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)}) \to \pi_y$*: Given a commitment C (to $\mathbf{A}$), a vector $\mathbf{A}$, and vectors $\mathbf{b}^{(0)}, \ldots, \mathbf{b}^{(t-1)} \in \mathbb{F}_p^n$, outputs a batch proof π_y for $\{y_i = \langle \mathbf{A}, \mathbf{b}^{(i)} \rangle\}_{i\in[0,t)}$.*
- $\mathsf{FC.BVerify}(C, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)}, \pi_y) \to \{0,1\}$*: Verifies the batch proof π_y for $\{y_i = \langle \mathbf{A}, \mathbf{b}^{(i)} \rangle\}_{i\in[0,t)}$ against C and outputs 1 (accept) or 0 (reject).*

An FC scheme is *correct* if $\mathsf{FC.BVerify}$ always outputs 1, provided that all algorithms are correctly executed.

Definition 1 (Correctness). *For any security parameter λ, any integer $n > 0$, any vector $\mathbf{A} \in \mathbb{G}_1^n$, any integer $t > 0$, any vectors $\mathbf{b}^{(0)}, \ldots, \mathbf{b}^{(t-1)} \in \mathbb{F}_p^n$, define $y_i = \langle \mathbf{A}, \mathbf{b}^{(i)} \rangle$ for all $i \in [0, t)$, then*

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{FC.Setup}(1^\lambda, 1^n), C \leftarrow \mathsf{FC.Commit}(\mathbf{A}), \\ \pi_y \leftarrow \mathsf{FC.BOpen}(C, \mathbf{A}, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)}) : \\ \mathsf{FC.BVerify}(C, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)}, \pi_y) = 1 \end{array}\right] = 1.$$

Referring to [69], an FC scheme is *function binding* if no probabilistic polynomial time (PPT) adversary can open the commitment C to two distinct values for the same field vector.

Definition 2 (Function binding). *For any security parameter λ, any integer $n > 0$, and any PPT adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{FC.Setup}(1^\lambda, 1^n), (C, \{\mathbf{b}^{(i)}, y_i, \hat{y}_i\}_{i\in[0,t)}, \pi_y, \pi_{\hat{y}}) \leftarrow \mathcal{A}(\mathsf{pp}) : \\ (\mathsf{FC.BVerify}(C, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)}, \pi_y) = 1) \wedge \\ (\mathsf{FC.BVerify}(C, \{\mathbf{b}^{(i)}\}_{i=0}^{t-1}, \{\hat{y}_i\}_{i=0}^{t-1}, \pi_{\hat{y}}) = 1) \wedge (\exists j \in [0, t) : y_j \neq \hat{y}_j) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

2.2 Polynomial Commitments

A *polynomial commitment* (PC) scheme [35,7,52,66,75] enables a prover to commit to an n-variate polynomial of maximum degree d (per variable), and later open its evaluation at any point by generating an evaluation proof. A PC scheme $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Eval}, \mathsf{Verify})$ consists of four algorithms:

- $\mathsf{PC.Setup}(1^\lambda, 1^d, 1^n) \rightarrow \mathsf{pp}$*: Given the security parameter λ, the maximum degree per variable d and the number of variables n, outputs the public parameters* pp*, an implicit input to all remaining algorithms.*
- $\mathsf{PC.Commit}(f) \rightarrow C$*: Outputs the commitment of f.*
- $\mathsf{PC.Eval}(f, \mathbf{r}) \rightarrow (y, \pi)$*: Generates a proof π showing that $f(\mathbf{r}) = y$.*
- $\mathsf{PC.Verify}(C, \mathbf{r}, y, \pi) \rightarrow \{0, 1\}$*: Returns 1 if for the committed polynomial f it holds that $f(\mathbf{r}) = y$.*

Informally, a PC scheme is *complete* if the verifier always accepts the proof for a correctly evaluated point.

Definition 3 (Completeness). *A PC scheme is complete if for any λ, n, d,*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, 1^d, 1^n), C \leftarrow \mathsf{PC.Commit}(f), \\ (y, \pi) \leftarrow \mathsf{PC.Eval}(f, \mathbf{r}) : \mathsf{PC.Verify}(C, \mathbf{r}, y, \pi) = 1 \end{array}\right] = 1.$$

A PC scheme is *knowledge sound* if for any PPT adversary that produces an accepting proof, there exists an extractor $\mathcal{E}_{PC}$ that extracts the committed polynomial f, such that the probability that $f(\mathbf{r}) \neq y$ is negligible.

Definition 4 (Knowledge Soundness). *A PC scheme is knowledge sound, if for any λ, n, d and PPT adversary $\mathcal{A}_{PC}$ there exists an extractor $\mathcal{E}_{PC}$ having access to $\mathcal{A}_{PC}$ such that:*

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, 1^d, 1^n), (C, y, \pi, \mathbf{r}) \leftarrow \mathcal{A}_{PC}(\mathsf{pp}), \\ f \leftarrow \mathcal{E}_{PC}^{\mathcal{A}_{PC}}(\mathsf{pp}) : (\mathsf{PC.Verify}(C, \mathbf{r}, y, \pi) = 1) \wedge \\ ((f(\mathbf{r}) \neq y) \vee (C \neq \mathsf{PC.Commit}(f))) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

For our construction, we use the PST multivariate polynomial commitment scheme [52], which has commitment and evaluation complexities *linear* in the size of the polynomial, and proof size and verification time logarithmic in it.

HyperEval Algorithm. In some scenarios, the prover needs to generate 2^n evaluation proofs of a multi-linear polynomial $f : \mathbb{F}_p^n \to \mathbb{F}_p$ at all hypercube points $\mathbf{i} \in \{0,1\}^n$. This is formalized by the $\mathsf{PC.HyperEval}$ [54]:

- $\mathsf{PC.HyperEval}(f) \to \{y_i, \pi_i\}_{i\in[0,2^n)}$*: Generate* 2^n *evaluation proofs showing that* $y_i = f(\mathsf{Bin}(i))$ *for all* $i \in [0, 2^n)$.

The $\mathsf{HyperEval}$ algorithm for the PST scheme achieves $\mathcal{O}(n2^n)$ complexity [62].

2.3 Vector Commitments

A *vector commitment* (VC) scheme [17,42] enables a prover to commit to a vector $\mathbf{m} \in \mathbb{F}_p^N$ and later prove that an element m_i is the i-th element of the committed vector. A VC scheme $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ consists of the following algorithms:

- $\mathsf{VC.Setup}(1^\lambda, 1^N) \to \mathsf{pp}$*: Given the security parameter* λ *and vector size* N*, outputs the public parameters, an implicit input to all remaining algorithms.*
- $\mathsf{VC.Commit}(\mathbf{m}) \to (C, \mathsf{aux})$*: Outputs the vector commitment of* $\mathbf{m}$ *and an auxiliary information* aux*.*
- $\mathsf{VC.Open}(\mathsf{aux}, i, \mathbf{m}) \to \pi_i$*: Outputs an opening proof* π_i *showing that* m_i *is the* i*-th element of* $\mathbf{m}$*.*
- $\mathsf{VC.Verify}(C, i, m_i, \pi_i) \to \{0,1\}$*: Returns 1 if* m_i *is the* i*-th element of the committed vector.*

A VC scheme is *correct* if $\mathsf{VC.Verify}$ always outputs 1, provided that all algorithms are correctly executed.

Definition 5 (Correctness). *For any security parameter* λ*, any integer* $N > 0$*, any vector* $\mathbf{m} \in \mathbb{F}_p^N$ *and any index* $i \in [0, N)$*, a VC scheme is correct if*

$$\Pr\left[\begin{matrix}\mathsf{pp} \leftarrow \mathsf{VC.Setup}(1^\lambda, 1^N), (C, \mathsf{aux}) \leftarrow \mathsf{VC.Commit}(\mathbf{m}),\\ \pi_i \leftarrow \mathsf{VC.Open}(\mathsf{aux}, i, \mathbf{m}) : \mathsf{VC.Verify}(C, i, \mathbf{m}[i], \pi_i) = 1\end{matrix}\right] = 1.$$

A VC scheme is *position binding* if the probability that a PPT adversary generates valid proofs for two different elements at the same index i is negligible.

Definition 6 (Position binding). *For any security parameter* λ*, any integer* $N > 0$*, and any PPT adversary* $\mathcal{A}$*,*

$$\Pr\left[\begin{matrix}\mathsf{pp} \leftarrow \mathsf{VC.Setup}(1^\lambda, 1^N), (C, i, m_i, m_i', \pi_i, \pi_i') \leftarrow \mathcal{A}(\mathsf{pp}) :\\ (\mathsf{VC.Verify}(C, i, m_i, \pi_i) = 1) \wedge\\ (\mathsf{VC.Verify}(C, i, m_i', \pi_i') = 1) \wedge (m_i \neq m_i')\end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

OpenAll Algorithm. Recent VC schemes [62,67,65,44,54] support an $\mathsf{OpenAll}$ algorithm that *efficiently* generates all opening proofs in an offline pre-processing phase and later reply to open queries with no additional computation. The formal definition of the $\mathsf{VC.OpenAll}$ algorithm is as follows.

- $\mathsf{VC.OpenAll}(\mathsf{aux}, \mathbf{m}) \to \{\pi_i\}_{i\in[0,N)}$: Generate N opening proofs, one for each element of the vector.

3 A Functional Commitment with Batch Opening

Bünz et al. [12] proposed a known-exponent *multi-exponentiation inner product argument* (MIPA) which employs structured-key variants of the commitment in [2] to commit to $(\mathbf{A}, \mathbf{b})$ and allows one to prove that a value $U \in \mathbb{G}_1$ is equal to $\langle \mathbf{A}, \mathbf{b}\rangle$ against this commitment, where $\mathbf{A} \in \mathbb{G}_1^n$ is hidden and $\mathbf{b} \in \mathbb{F}_p^n$ is publicly known and structured of the form $(1, b, \ldots, b^{n-1})$ for some $b \in \mathbb{F}_p$.

We observe that in the MIPA, the vector $\mathbf{b}$ can be any vector in $\mathbb{F}_p^n$ and extend the argument to the *first* FC scheme with batch openings for multi-exponentiations. To develop the final FC scheme, we first transform the MIPA (which originally verified $\langle \mathbf{A}, \mathbf{b}\rangle$ for fixed $\mathbf{A}$ and $\mathbf{b}$) into an FC scheme that commits to $\mathbf{A}$, enabling verification of $\langle \mathbf{A}, \mathbf{b}\rangle$ for *any* $\mathbf{b}$. Opening the commitment of vector $\mathbf{A}$ to $y_0 = \langle \mathbf{A}, \mathbf{b}^{(0)}\rangle, \ldots, y_{t-1} = \langle \mathbf{A}, \mathbf{b}^{(t-1)}\rangle$ is equivalent to proving:

$$\{\langle \mathbf{A}, \mathbf{b}^{(i)}\rangle = y_i\}_{i\in[0,t)}. \tag{2}$$

As shown in Lemma 1 (Appendix B), this task can be reduced to showing that a randomized linear combination of these equations holds:

$$\langle \mathbf{A}, \sum_{i\in[0,t)} r_i \mathbf{b}^{(i)}\rangle = \prod_{i\in[0,t)} y_i^{r_i}, \tag{3}$$

where each r_i is random and may be computed with a hash function. We then show that this FC scheme gives a VC scheme in which $\mathsf{VC.OpenAll}$ algorithm runs in linear time (Section 4). Below we give a formal description of our FC scheme, assuming without loss of generality that the vector length n is a power of 2, i.e., $n = 2^\ell$ for some integer ℓ.

- $\mathsf{FC.Setup}(1^\lambda, 1^n)$: *Choose* $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e, g_1, g_2) \leftarrow \mathsf{BG}(1^\lambda)$ *and* $\beta \leftarrow \mathbb{F}_p$. *Compute* $\mathbf{v} = \left((g_2)^{\beta^{2i}}\right)_{i\in[0,n)}$ *and output* $\mathsf{pp} = \mathbf{v}$. *Note that* β *must never be known to the adversary, and thus our scheme requires a trusted setup.*
- $\mathsf{FC.Commit}(\mathbf{A})$: *Given the implicit input* $\mathsf{pp} = \mathbf{v} \in \mathbb{G}_2^n$ *and a vector* $\mathbf{A} \in \mathbb{G}_1^n$, *output a commitment* $C = \mathbf{A} * \mathbf{v}$.
- $\mathsf{FC.BOpen}(C, \mathbf{A}, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)})$: *Parse the implicit input* pp *as* $\mathbf{v}$. *Set*

$$\mathbf{b} = \sum_{i\in[0,t)} r_i \mathbf{b}^{(i)}, \tag{4}$$

where $r_i = H(\mathbf{b}^{(i)}, C, \{\mathbf{b}^{(k)}\}_{k\in[0,t)}, \{y_k\}_{k\in[0,t)}), \forall i \in [0,t)$, *and* $H : \{0,1\}^* \to \mathbb{F}_p$ *is a hash function. Set* $\mathbf{A}_0 = \mathbf{A}, \mathbf{v}_0 = \mathbf{v}, \mathbf{b}_0 = \mathbf{b}$, *and let* x_0 *be the empty string. For* $j = 1, 2, \ldots, \ell$, *compute*

$$\begin{aligned}
\mathbf{L}_j &= ((\mathbf{A}_{j-1})_R * (\mathbf{v}_{j-1})_L, \langle (\mathbf{A}_{j-1})_R, (\mathbf{b}_{j-1})_L \rangle), \\
\mathbf{R}_j &= ((\mathbf{A}_{j-1})_L * (\mathbf{v}_{j-1})_R, \langle (\mathbf{A}_{j-1})_L, (\mathbf{b}_{j-1})_R \rangle), \\
x_j &= H'(x_{j-1}, \mathbf{L}_j, \mathbf{R}_j), \\
\mathbf{A}_j &= (\mathbf{A}_{j-1})_L \circ ((\mathbf{A}_{j-1})_R)^{x_j}, \\
\mathbf{v}_j &= (\mathbf{v}_{j-1})_L \circ ((\mathbf{v}_{j-1})_R)^{1/x_j}, \\
\mathbf{b}_j &= (\mathbf{b}_{j-1})_L + (1/x_j) \cdot (\mathbf{b}_{j-1})_R,
\end{aligned} \tag{5}$$

where $H' : \{0,1\}^* \to \mathbb{F}_p$ *is a hash function. Output* $\pi_y = \left\{\{\mathbf{L}_j, \mathbf{R}_j\}_{j=1}^{\ell}, \mathbf{A}_\ell\right\}$.

– $\mathsf{FC.BVerify}(C, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)}, \pi_y)$*: Parse the implicit input* pp *as* $\mathbf{v}$. *Compute* $\{r_i\}_{i\in[0,t)}$ *and* $\mathbf{b}$ *as in algorithm* $\mathsf{FC.BOpen}$, *and then compute*

$$y = \prod_{i\in[0,t)} y_i^{r_i}. \tag{6}$$

Let $\mathbf{C}_0 = (C, y)$ *and let* x_0 *be the empty string. Parse* π_y *as* $\{\{\mathbf{L}_j, \mathbf{R}_j\}_{j=1}^{\ell}, \mathbf{A}_\ell\}$. *For* $j = 1, 2, \ldots, \ell$, *compute* $x_j, \mathbf{v}_j$ *and* $\mathbf{b}_j$ *as per Eq. (5), and compute*

$$\mathbf{C}_j = \mathbf{L}_j^{x_j} \circ \mathbf{C}_{j-1} \circ (\mathbf{R}_j)^{1/x_j}.$$

Finally, if $\mathbf{C}_\ell = (\mathbf{A}_\ell * \mathbf{v}_\ell, \langle \mathbf{A}_\ell, \mathbf{b}_\ell \rangle)$, *output 1; otherwise, output 0.*

Correctness and security. The correctness of our FC scheme is proven below, while its security (function binding) is proven in Appendix B.1.

Theorem 1. *The FC scheme satisfies the correctness property (Definition 1).*

Proof. To prove the correctness of the FC scheme, it suffices to show that for all $k \in \{0, 1, \ldots, \ell\}$, $\mathbf{C}_i = (\mathbf{A}_i * \mathbf{v}_i, \langle \mathbf{A}_i, \mathbf{b}_i \rangle)$. We begin by proving this statement for the base case $k = 0$. Since for each $i \in [0,t)$, $y_i = \langle \mathbf{A}, \mathbf{b}^{(i)} \rangle$ and r_i is computed as in algorithm $\mathsf{FC.BOpen}$, we obtain

$$\prod_{i\in[0,t)} y_i^{r_i} = \prod_{i\in[0,t)} \langle \mathbf{A}, \mathbf{b}^{(i)} \rangle^{r_i} = \left\langle \mathbf{A}, \sum_{i\in[0,t)} r_i \mathbf{b}^{(i)} \right\rangle.$$

By equations (4) and (6), it follows that $y = \langle \mathbf{A}, \mathbf{b} \rangle$. Hence, it is evident that $\mathbf{C}_0 = (C, y) = (\mathbf{A}_0 * \mathbf{v}_0, \langle \mathbf{A}_0, \mathbf{b}_0 \rangle)$, as required.

By mathematical induction, it remains to show the statement is true for $k = j$ when it is true for $k = j - 1$. For simplicity, we denote $\bar{\mathbf{A}} = \mathbf{A}_{j-1}, \bar{\mathbf{v}} = \mathbf{v}_{j-1}$, and $\bar{\mathbf{b}} = \mathbf{b}_{j-1}$. According to the algorithm $\mathsf{FC.BVerify}$, $\mathbf{C}_j = \mathbf{L}_j^{x_j} \circ \mathbf{C}_{j-1} \circ (\mathbf{R}_j)^{1/x_j}$. By Eq. (5) and the induction hypothesis,

$$\begin{aligned}
\mathbf{C}_j &= (\bar{\mathbf{A}}_R * \bar{\mathbf{v}}_L, \langle \bar{\mathbf{A}}_R, \bar{\mathbf{b}}_L \rangle)^{x_j} \circ (\bar{\mathbf{A}} * \bar{\mathbf{v}}, \langle \bar{\mathbf{A}}, \bar{\mathbf{b}} \rangle) \circ (\bar{\mathbf{A}}_L * \bar{\mathbf{v}}_R, \langle \bar{\mathbf{A}}_L, \bar{\mathbf{b}}_R \rangle)^{1/x_j} \\
&= ((\bar{\mathbf{A}}_R^{x_j} * \bar{\mathbf{v}}_L) \cdot (\bar{\mathbf{A}} * \bar{\mathbf{v}}) \cdot (\bar{\mathbf{A}}_L^{1/x_j} * \bar{\mathbf{v}}_R), \langle \bar{\mathbf{A}}_R^{x_j}, \bar{\mathbf{b}}_L \rangle \cdot \langle \bar{\mathbf{A}}, \bar{\mathbf{b}} \rangle \cdot \langle \bar{\mathbf{A}}_L^{1/x_j}, \bar{\mathbf{b}}_R \rangle).
\end{aligned}$$

Since $\bar{\mathbf{A}} * \bar{\mathbf{v}} = (\bar{\mathbf{A}}_L * \bar{\mathbf{v}}_L) \cdot (\bar{\mathbf{A}}_R * \bar{\mathbf{v}}_R)$ and $\langle \bar{\mathbf{A}}, \bar{\mathbf{b}} \rangle = \langle \bar{\mathbf{A}}_L, \bar{\mathbf{b}}_L \rangle \cdot \langle \bar{\mathbf{A}}_R, \bar{\mathbf{b}}_R \rangle$, we have

$$\mathbf{C}_j = (((\bar{\mathbf{A}}_R^{x_j} \circ \bar{\mathbf{A}}_L) * \bar{\mathbf{v}}_L) \cdot ((\bar{\mathbf{A}}_R \circ \bar{\mathbf{A}}_L^{1/x_j}) * \bar{\mathbf{v}}_R), \langle \bar{\mathbf{A}}_R^{x_j} \circ \bar{\mathbf{A}}_L, \bar{\mathbf{b}}_L \rangle \cdot \langle \bar{\mathbf{A}}_R \circ \bar{\mathbf{A}}_L^{1/x_j}, \bar{\mathbf{b}}_R \rangle).$$

Since $\bar{\mathbf{A}}_R \circ \bar{\mathbf{A}}_L^{1/x_j} = (\bar{\mathbf{A}}_R^{x_j} \circ \bar{\mathbf{A}}_L)^{1/x_j}$, by replacing the left-hand side of this equality with the right-hand side in $\mathbf{C}_j$, we have

$$\mathbf{C}_j = ((\bar{\mathbf{A}}_R^{x_j} \circ \bar{\mathbf{A}}_L) * (\bar{\mathbf{v}}_L \circ \bar{\mathbf{v}}_R^{1/x_j}), \langle \bar{\mathbf{A}}_R^{x_j} \circ \bar{\mathbf{A}}_L, \bar{\mathbf{b}}_L + 1/x_j \cdot \bar{\mathbf{b}}_R \rangle).$$

According to Eq. (5), $\mathbf{C}_j = (\mathbf{A}_j * \mathbf{v}_j, \langle \mathbf{A}_j, \mathbf{b}_j \rangle)$. □

Faster verification. The FC scheme uses the same approach as the known-exponent MIPA in Bünz et al. [12], which allows the verifier to outsource the computation of $\mathbf{v}_\ell$ to the untrusted prover and reduces verification time. However, this comes at the cost of additionally relying on the n-SDH assumption (**Assumption** 1). We implicitly assume the faster verifier.

Efficiency. The efficiency of the FC scheme with faster verification is analyzed below and summarized in Table 1. In our analysis of group operations, we focus on pairings and exponentiations, ignoring the cheaper multiplications.

- *Proof size.* The original proof contains $2 \log n$ elements in $\mathbb{G}_\mathrm{T}$ and $2 \log n + 1$ elements in $\mathbb{G}_1$. To enable faster verification, two additional elements in $\mathbb{G}_2$ (i.e., $\mathbf{v}_\ell$ and its proof) were included. Hence, the proof size is $\mathcal{O}(\log n)$.
- *Commit.* Committing to an n-sized vector requires n pairings.
- *Batch opening.* Computing vector $\mathbf{b}$ requires tn field operations. Computing $\{\mathbf{L}_j, \mathbf{R}_j\}_{j=1}^{\ell}$ requires $2n$ pairings and $2n$ exponentiations in $\mathbb{G}_1$. Rescaling $\mathbf{A}, \mathbf{v}, \mathbf{b}$ requires n exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$, n field operations. Moreover, to achieve faster verification, the proof of $\mathbf{v}_\ell$ must be computed, and this computation requires performing $2n$ exponentiations in $\mathbb{G}_2$.
- *Batch verification.* Computing $\mathbf{b}$ requires tn field operations, and computing y requires t exponentiations in $\mathbb{G}_1$. Rescaling the commitments and $\mathbf{b}$ requires $2 \log n$ exponentiations in $\mathbb{G}_\mathrm{T}$ and $\mathbb{G}_1$, and n field operations. The final check equation requires 1 pairing and an exponentiation in $\mathbb{G}_1$. With faster verification, the verifier no longer needs to compute $\mathbf{v}_l$, but needs to verify its correctness; this requires two pairings and $\log n$ field operations.

Table 1: The efficiency of our FC scheme with faster verification.

Algorithms	**Field operations**	**Exponentiations**	**Pairings**
FC.Commit	-	-	$\mathcal{O}(n)$
FC.BOpen	$\mathcal{O}(tn)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
FC.BVerify	$\mathcal{O}(tn)$	$\mathcal{O}(\log n + t)$	$\mathcal{O}(1)$

4 A VC Scheme with Linear-Time OpenAll

Based on the scheme FC from Section 3, we construct FlexProofs (denoted by VC), a VC scheme that achieves linear-time VC.OpenAll with a flexible batch size parameter b, where b provides a tradeoff and the running time of VC.OpenAll can be continuously reduced by increasing b. Our scheme is directly compatible with the zkSNARKs based on multi-linear polynomials.

4.1 FlexProofs

FlexProofs is structured in two layers. We set $\mu = \sqrt{N}$ and partition the vector $\mathbf{m} \in \mathbb{F}_p^N$ into μ subvectors of length μ:

$$\mathbf{m} = \big(\mathbf{m}_0, \mathbf{m}_1, \ldots, \mathbf{m}_{\mu-1}\big).$$

In the first layer, each $\mathbf{m}_j$ $(j \in [0, \mu))$ is encoded using its multilinear extension $f_j : \mathbb{F}_p^{\log \mu} \to \mathbb{F}_p$ and committed via a commitment scheme PC [52] for multilinear polynomials. In the second layer, the vector of the commitments of all $\mathbf{m}_j (j \in [0, \mu))$ is committed using the scheme FC from Section 3. Specifically, the algorithms VC.Setup and VC.Commit in our VC scheme are as follows.

- $\mathsf{VC.Setup}(1^\lambda, 1^N)$*: Generate public parameters for* PC *and* FC *by invkoing*

$$\mathsf{pp}_{PC} \leftarrow \mathsf{PC.Setup}(1^\lambda, 1, \log \mu), \ \ \mathsf{pp}_{FC} \leftarrow \mathsf{FC.Setup}(1^\lambda, 1^\mu). \tag{7}$$

  Output the public parameters $\mathsf{pp} = \{\mathsf{pp}_{PC}, \mathsf{pp}_{FC}\}$.
- $\mathsf{VC.Commit}(\mathbf{m})$*: For each* $j \in [0, \mu)$*, the subvector* $\mathbf{m}_j$ *is encoded using its multilinear extension* f_j *and then committed as*

$$C_j \leftarrow \mathsf{PC.Commit}(f_j). \tag{8}$$

  For the vector $\mathbf{C} = (C_0, \ldots, C_{\mu-1})$ *of* μ *polynomial commitments, generate a functional commitment*

$$C \leftarrow \mathsf{FC.Commit}(\mathbf{C}). \tag{9}$$

  Output the commitment C *and an auxiliary information string* $\mathsf{aux} = \mathbf{C}$.

Given the commitment process defined by (8) and (9), there is a *two-step* method of proving the correctness of $\mathbf{m}[i]$ for each $i \in [0, N)$. First, use our FC scheme to prove the correctness of $C_{\lfloor i/\mu \rfloor}$ with respect to C, i.e., $C_{\lfloor i/\mu \rfloor} = \langle \mathbf{C}, \mathbf{u}_{\lfloor i/\mu \rfloor} \rangle$. Second, use the PC scheme to prove the correctness of $\mathbf{m}[i]$ with respect to $C_{\lfloor i/\mu \rfloor}$, i.e., $f_{\lfloor i/\mu \rfloor}(\mathsf{Bin}(i \bmod \mu)) = \mathbf{m}[i]$. Since both steps have time complexity $\mathcal{O}(\sqrt{N})$, the computational cost of the two-step method is $\mathcal{O}(\sqrt{N})$. Consequently, proving the correctness of all elements of $\mathbf{m}$ one after the other with the two-step method requires $\mathcal{O}(N\sqrt{N})$ time. By examining the two-step proving process, we observe that: (1) proving the correctness of C_j with respect to C takes place μ times and can be done only once for all elements in $\mathbf{m}_j$; (2) for all $j \in [0, \mu)$, the μ processes of proving the correctness of all elements in $\mathbf{m}_j$ with respect to C_j share a similar structure and thus can be combined. Based on these observations, we propose an *OpenAll* protocol, in which the prover can prove the correctness of all elements in $\mathbf{m}$ in $\mathcal{O}(N)$ time.

4.2 Linear-Time OpenAll

In this section, we first present *OpenAll* as an interactive protocol and then make it non-interactive with the Fiat-Shamir variant of [74], yielding the algorithms VC.OpenAll and VC.Verify. In the non-interactive version of our VC scheme, the executions of the algorithms VC.Commit and VC.OpenAll form a *pre-processing* phase and the latter algorithm generates N opening proofs for any vector $\mathbf{m}$ of length N. In particular, the N opening proofs can be used to directly reply any query to VC.Open, which is an algorithm different from VC.OpenAll and every time generates an opening proof for any *single* element of the vector.

The interactive protocol *OpenAll* is a protocol between a prover holding a vector $\mathbf{m}$ and N verifiers $\{\mathcal{V}_i\}_{i\in[0,N)}$, where each verifier $\mathcal{V}_i$ holds a commitment C (to $\mathbf{m}$) and a value m_i, and verifies that m_i is the i-th element of the vector committed in C by interacting with the prover.

Intuition of our protocol. Our *OpenAll* protocol consists of two steps: (1) proving the correctness of C_j with respect to C for all $j \in [0, \mu)$, and (2) combining the μ processes of proving the correctness of all elements in $\mathbf{m}_j$ with respect to C_j. For the first step, if the prover uses the scheme FC from Section 3 to generate a proof for each C_j, then it needs to generate μ proofs and thus performs $\mathcal{O}(N)$ field operations and $\mathcal{O}(N)$ cryptographic operations (e.g., group exponentiations or pairings) according to Table 1. To reduce this cost, the prover partitions the vector $\mathbf{C}$ into blocks and uses FC to generate one batch proof per block. With a batch size of b, the prover only needs to generate $\lceil \mu/b \rceil$ batch proofs, which require $\mathcal{O}(N)$ field operations and $\mathcal{O}(N/b)$ cryptographic operations. For the second step, we combine the μ processes by folding μ polynomials into a single polynomial [27,10], called *folded* polynomial, which is essentially a linear combination of the μ polynomials, such that if an evaluation claim at a hypercube point is wrong in one of the μ polynomials, then it will be wrong for the folded polynomial with overwhelming probability. For the folded polynomial, we invoke PC.HyperEval.

Protocol description. Formally our interactive *OpenAll* protocol is shown in **Protocol 1** and consists of the following two steps.

- *The first step:* The prover partitions the vector $\mathbf{C}$ into $\lceil \mu/b \rceil$ blocks of size b such that
$$\mathbf{C} = (\mathbf{C}_0, \ldots, \mathbf{C}_{\lceil \mu/b \rceil - 1}),$$
where $\mathbf{C}_i = (C_{ib}, \ldots, C_{(i+1)b-1})$. For each $k \in [0, \lceil \mu/b \rceil)$, the prover invokes FC.BOpen to generate a batch proof for the correctness of $\mathbf{C}_k$, and sends $\mathbf{C}_k$ together with this proof to the verifiers $\{\mathcal{V}_i\}_{i\in[kb\mu,(k+1)b\mu)}$. These verifiers then validate using FC.BVerify.
- *The second step:* At this point, $\mathcal{V}_i$ assures that $C_{\lfloor i/\mu \rfloor}$ is the $\lfloor i/\mu \rfloor$-th element of the vector committed in C. It remains to verify that $f_{\lfloor i/\mu \rfloor}(\mathsf{Bin}(i \bmod \mu)) = \mathbf{m}[i]$. The prover has to prove this identity for all $i \in [0, N)$ or equivalently generate proofs for the evaluations of the μ polynomials $\{f_j\}_{j=0}^{\mu-1}$ at all boolean hypercube points in $\{0,1\}^{\log \mu}$. If the prover invokes PC.HyperEval on

Protocol 1: Efficient OpenAll. We assume a prover holding a vector $\mathbf{m} \in \mathbb{F}_p^N$, the commitment C and an auxiliary information string $\mathsf{aux}(= \mathbf{C})$. We assume N verifiers $\{\mathcal{V}_i\}_{i\in[0,N)}$, where each verifier $\mathcal{V}_i$ holds the commitment C and an element m_i and wants to verify whether m_i is the i-th element of $\mathbf{m}$.

- **Step 1: Prove the correctness of $\mathbf{C}_k$ for all $k \in [0, \lceil \mu/b \rceil)$.**
  1) For each $k \in [0, \lceil \mu/b \rceil)$, the prover sets $S_k = [kb, (k+1)b)$, generates a batch proof
  $$\pi_{\mathbf{C}_k} \leftarrow \mathsf{FC.BOpen}(C, \mathbf{C}, \{\mathbf{u}_a\}_{a\in S_k}, \{C_a\}_{a\in S_k}), \tag{10}$$
  and sends $(\mathbf{C}_k, \pi_{\mathbf{C}_k})$ to the verifiers $\{\mathcal{V}_i\}_{i\in[k\mu b,(k+1)\mu b)}$.
  2) The verifier $\mathcal{V}_i$ sets $k = \lfloor i/(\mu b) \rfloor$ and $S_k = [kb, (k+1)b)$, and checks the correctness of $\mathbf{C}_k$ by executing
  $$\mathsf{FC.BVerify}(C, \{\mathbf{u}_a\}_{a\in S_k}, \{C_a\}_{a\in S_k}, \pi_{\mathbf{C}_k}).$$
- **Step 2: Fold all μ polynomials $\{f_j\}_{j=0}^{\mu-1}$ into a single polynomial g^* and then invoke $\mathsf{PC.HyperEval}$ on the folded polynomial g^*.**
  1) The prover receives random values $\{r_j\}_{j\in[0,\mu)}$ from the verifiers and computes
  $$g_j = r_j f_j, \qquad D_j = C_j^{r_j}. \tag{11}$$
  For each $i \in [0, N)$, the verifier $\mathcal{V}_i$, holding m_i and $C_{\lfloor i/\mu \rfloor}$, computes
  $$y_{\lfloor i/\mu \rfloor,(i \bmod \mu)} = r_{\lfloor i/\mu \rfloor}\, m_i, \qquad D_{\lfloor i/\mu \rfloor} = C_{\lfloor i/\mu \rfloor}^{r_{\lfloor i/\mu \rfloor}} \tag{12}$$
  The prover folds the randomized polynomials $\{g_j\}_{j\in[0,\mu)}$ into a single polynomial in a tree-like, bottom-up fashion. At the leaf level, the state of the j-th leaf is initialized as $\big(\mathcal{P} : \{g_j\},\ \boldsymbol{\mathcal{V}}_j : \{(y_{j,a}, D_j)\}_{a\in[0,\mu)}\big)$, where the prover $\mathcal{P}$ holds only the polynomial g_j, and $\boldsymbol{\mathcal{V}}_j = (\mathcal{V}_{j\mu}, \ldots, \mathcal{V}_{(j+1)\mu-1})$ is the vector of verifiers, with each $\boldsymbol{\mathcal{V}}_j[a]$ holding $y_{j,a}$ claimed to be $g_j(\mathsf{Bin}(a))$ together with the commitment D_j. For each internal node w, the state $\mathcal{S}_w = (\mathcal{P} : \{g_w\},\ \boldsymbol{\mathcal{V}}_w : \{(y_{w,a}, D_w)\}_{a\in[0,\mu)})$ is computed from its left child $\mathcal{S}_{w_l}$ and right child $\mathcal{S}_{w_r}$ using the **Sub-protocol**. This process is repeated recursively up the tree until the root.
  2) At the root, the final state is $\mathcal{S}^* = \big(\mathcal{P} : \{g^*\},\ \boldsymbol{\mathcal{V}}^* : \{(y_a^*, D^*)\}_{a\in[0,\mu)}\big)$. The prover follows the $\mathsf{PC.HyperEval}$ algorithm to generate μ proofs $\{\pi_a^*\}_{a\in[0,\mu)}$ proving that each $y_a^* = g^*(\mathsf{Bin}(a))$, and sends π_a^* to verifiers $\{\mathcal{V}_{j\mu+a}\}_{j\in[0,\mu)}$. Verifiers $\{\mathcal{V}_{j\mu+a}\}_{j\in[0,\mu)}$ validates by running $\mathsf{PC.Verify}(D^*, \mathsf{Bin}(a), y_a^*, \pi_a^*)$.

the μ polynomials one by one, the total time complexity will be $\mathcal{O}(N \log \sqrt{N})$. To be more efficient, we first fold the μ polynomials into a single polynomial g^* and then invoke $\mathsf{PC.HyperEval}$ on the folded polynomial g^*.

Next, for clarity, each verifier step is presented immediately after its corresponding prover step to illustrate all participants' actions. To efficiently compute the folded polynomial g^*, the verifiers send a set $\{r_j\}_{j\in[0,\mu)}$ of random values to the prover. The prover then uses each random value r_j to compute a randomized polynomial $g_j = r_j f_j$ and a commitment $D_j = C_j^{r_j}$. Meanwhile, for each $i \in [0, N)$, the verifier $\mathcal{V}_i$, holding m_i and $C_{\lfloor i/\mu \rfloor}$, computes $y_{\lfloor i/\mu \rfloor,(i \bmod \mu)} = r_{\lfloor i/\mu \rfloor} m_i$ and $D_{\lfloor i/\mu \rfloor} = C_{\lfloor i/\mu \rfloor}^{r_{\lfloor i/\mu \rfloor}}$. The prover folds the randomized polynomials $\{g_j\}_{j\in[0,\mu)}$ into a single polynomial in a tree-like, bottom-up fashion. At the leaf level, the state of the j-th leaf is initialized as $\big(\mathcal{P} : \{g_j\},\ \boldsymbol{\mathcal{V}}_j : \{(y_{j,a}, D_j)\}_{a\in[0,\mu)}\big)$, where the prover $\mathcal{P}$ holds only the polynomial g_j, and $\boldsymbol{\mathcal{V}}_j = (\mathcal{V}_{j\mu}, \ldots, \mathcal{V}_{(j+1)\mu-1})$ is the vector of verifiers, with each $\boldsymbol{\mathcal{V}}_j[a]$ holding $y_{j,a}$ claimed to be $g_j(\mathsf{Bin}(a))$ together with the commitment D_j. For each internal node w, the state

$$\mathcal{S}_w = (\mathcal{P} : \{g_w\},\ \boldsymbol{\mathcal{V}}_w : \{(y_{w,a}, D_w)\}_{a\in[0,\mu)})$$

is computed from its left child $\mathcal{S}_{w_l}$ and right child $\mathcal{S}_{w_r}$ using the **Sub-protocol**. This process is repeated recursively up the tree until the root.

Sub-protocol: Computing the parent state from the two child states. For each $z \in \{l, r\}$, the child node w_z is associated with a state

$$\mathcal{S}_{w_z} = (\mathcal{P} : \{g_{w_z}\},\ \boldsymbol{\mathcal{V}}_{w_z} : \{(y_{w_z,a}, D_{w_z})\}_{a\in[0,\mu)}),$$

where the prover $\mathcal{P}$ holds the polynomial g_{w_z}, and for each $a \in [0, \mu)$, the verifiers $\boldsymbol{\mathcal{V}}_{w_z}[a]$ hold a point $y_{w_z,a}$ claimed to be $g_{w_z}(\mathsf{Bin}(a))$ together with the commitment D_{w_z}. At the end of the folding, the parent node w has a state

$$\mathcal{S}_w = (\mathcal{P} : \{g_w\},\ \boldsymbol{\mathcal{V}}_w : \{(y_{w,a}, D_w)\}_{a\in[0,\mu)}),$$

where the prover holds the polynomial g_w, and $\boldsymbol{\mathcal{V}}_w = \big(\boldsymbol{\mathcal{V}}_{w_l}[0] \cup \boldsymbol{\mathcal{V}}_{w_r}[0],\ \ldots,\ \boldsymbol{\mathcal{V}}_{w_l}[\mu - 1] \cup \boldsymbol{\mathcal{V}}_{w_r}[\mu - 1]\big)$ collects the verifier vectors. For each $a \in [0, \mu)$, both $\boldsymbol{\mathcal{V}}_{w_l}[a]$ and $\boldsymbol{\mathcal{V}}_{w_r}[a]$ now hold the same claimed evaluation $y_{w,a}$ and the commitment D_w. Our protocol works as follows.

1. The prover computes the folded polynomial $g_w(\mathbf{x}) = g_{w_l}(\mathbf{x}) + g_{w_r}(\mathbf{x})$.
2. For every $\boldsymbol{\mathcal{V}}_{w_l}[a]$, the prover provides D_{w_r} and $y_{w_r,a}$ and for every $\boldsymbol{\mathcal{V}}_{w_r}[a]$, the prover provides D_{w_l} and $y_{w_l,a}$.
3. Next, every $\boldsymbol{\mathcal{V}}_w[a]$ computes $y_{w,a} = y_{w_l,a} + y_{w_r,a}$ and $D_w = D_{w_l} D_{w_r}$.

At the root, the final state is $\mathcal{S}^* = \big(\mathcal{P} : \{g^*\},\ \boldsymbol{\mathcal{V}}^* : \{(y_a^*, D^*)\}_{a\in[0,\mu)}\big)$, where the prover holds the final folded polynomial g^*, and each verifier $\mathcal{V}_{j\mu+a}$ (for $j \in [0, \mu), a \in [0, \mu)$) holds the claimed evaluation $y_a^* = g^*(\mathsf{Bin}(a))$ together with the commitment D^*. Finally, the prover runs the $\mathsf{PC.HyperEval}$ algorithm for g^* to generate proofs $\{\pi_a^*\}_{a\in[0,\mu)}$ proving that each $y_a^* = g^*(\mathsf{Bin}(a))$,

and sends π_a^* to verifiers $\{\mathcal{V}_{j\mu+a}\}_{j\in[0,\mu)}$. Verifiers $\{\mathcal{V}_{j\mu+a}\}_{j\in[0,\mu)}$ validates by running $\mathsf{PC.Verify}(D^*, \mathsf{Bin}(a), y_a^*, \pi_a^*)$.

Making our protocol non-interactive. Up to this point, our protocol has been presented in the interactive setting: in *Step 1*, the prover sends a message to the verifiers; in *Step 2*, the prover receives their challenges and responds accordingly. To make it non-interactive, we adopt the Fiat-Shamir variant of [74], which is designed for the multi-verifier setting and proven secure in ROM. Our protocol becomes non-interactive as follows. For *Step 2*, the prover builds a Merkle tree by hashing the coefficients of all polynomials to be folded and their corresponding commitments. Each verifier then receives a proof that includes a Merkle tree membership path, showing that its data is correctly included.

The correctness and security of FlexProofs are proven in Appendix C. A zero-knowledge variant of *OpenAll* can also be obtained using standard techniques [20,40] (see Appendix A for the formal definition). In this case, the PC scheme is replaced with its zero-knowledge version, whereas the FC scheme does not require zero-knowledge since it only operates on already hidden commitments.

4.3 Efficiency

Efficiency improvements in the first step of Protocol 1. Note that the working process of the first step in the protocol implies the *first* subvector commitment scheme over group elements. Below, we describe some efficiency improvements for this scheme. In the FC model, the vector length is denoted by n; in our setting this length is μ, so we set $\ell = \log \mu$. As a starting point, we consider the case where $\mathbf{b}$ in Eq. (5) is the i-th *unit vector* $\mathbf{u}_i$. Note that $\mathsf{Bin}(i) = (i_{\ell-1}, \ldots, i_0)$. In the first folding round:

$$(\mathbf{u}_i)_1 = (\mathbf{u}_i)_L + 1/x_1 \cdot (\mathbf{u}_i)_R.$$

If $i_{\ell-1} = 0$, the only non-zero entry 1 appears in $(\mathbf{u}_i)_L$, resulting in a single non-zero entry of 1 in $(\mathbf{u}_i)_1$. Conversely, if $i_{\ell-1} = 1$, the only non-zero 1 entry appears in $(\mathbf{u}_i)_R$, which leads to a single non-zero entry of $1/x_1$ in $(\mathbf{u}_i)_1$. In summary, the only non-zero entry in $(\mathbf{u}_i)_1$ is $(1/x_1)^{i_{\ell-1}}$, located at the index $(i_{\ell-2}, \ldots, i_0)$. In general, at the j-th round, the only non-zero entry in $(\mathbf{u}_i)_j$ is $\prod_{k=1}^{j} (1/x_k)^{i_{\ell-k}}$, with index $(i_{\ell-j-1}, \ldots, i_0)$. Hence, after ℓ rounds, we have:

$$(\mathbf{u}_i)_\ell = \prod_{k=1}^{\ell} (1/x_k)^{i_{\ell-k}}. \tag{13}$$

Let $I \subseteq [0, \mu)$ be a set of indices. For batch opening over I, $\mathbf{b} = \sum_{i\in I} r_i \mathbf{u}_i$. Let $\hat{\mathbf{u}}_i = r_i \mathbf{u}_i (i \in I)$, thus $\mathbf{b} = \sum_{i\in I} \hat{\mathbf{u}}_i$. According to the folding of $\mathbf{b}$ in Eq. (5), at the first round: $\mathbf{b}_1 = \mathbf{b}_L + 1/x_1 \mathbf{b}_R = (\sum_{i\in I} \hat{\mathbf{u}}_i)_L + 1/x_1 (\sum_{i\in I} \hat{\mathbf{u}}_i)_R = \sum_{i\in I} (\hat{\mathbf{u}}_i)_L + 1/x_1 (\hat{\mathbf{u}}_i)_R = \sum_{i\in I} (\hat{\mathbf{u}}_i)_1$; at the j-th round: $\mathbf{b}_j = \sum_{i\in I} ((\hat{\mathbf{u}}_i)_{j-1})_L +$

$1/x_j((\hat{\mathbf{u}}_i)_{j-1})_R = \sum_{i\in I}(\hat{\mathbf{u}}_i)_j$. After ℓ rounds and by Eq. (13):

$$\mathbf{b}_\ell = \sum_{i\in I}(\hat{\mathbf{u}}_i)_\ell \text{ and } (\hat{\mathbf{u}}_i)_\ell = r_i \prod_{k=1}^{\ell}(1/x_k)^{i_{\ell-k}}.$$

Therefore, FC.BOpen can run faster, and FC.BVerify can run in $\mathcal{O}(|I|\log\mu)$ time.

Complexity analysis of FlexProofs. The underlying homomorphic PC scheme for multi-linear polynomials achieves commitment and evaluation complexity of $\mathcal{O}(N)$, proof size of $\mathcal{O}(\log N)$, verification time of $\mathcal{O}(\log N)$, and PC.HyperEval complexity of $\mathcal{O}(N\log N)$. Next, we analyze the complexity of FlexProofs.

- *Commitment complexity.* Algorithm VC.Commit makes μ calls to PC.Commit, each on a polynomial of size μ, and then makes one call to FC.Commit on a vector of size μ. Due to the linear commit complexity of both the PC scheme and our FC scheme, the overall commitment complexity is $\mathcal{O}(N)$.
- *Prover complexity.* In *Step 1*, the prover creates proofs for $\lceil\mu/b\rceil$ batches, each containing b elements. According to Table 1, this step takes $\mathcal{O}(N)$ field operations and $\mathcal{O}(N/b)$ cryptographic operations. In *Step 2*, the prover has to (1) randomize polynomials and their commitments, (2) fold polynomials in a tree-like structure, (3) call PC.HyperEval. Due to the underlying PC scheme achieves PC.HyperEval complexity of $\mathcal{O}(N\log N)$, both (1) and (2) require $\mathcal{O}(N)$ field operations and $\mathcal{O}(\mu)$ cryptographic operations, (3) has a complexity of $\mathcal{O}(\sqrt{N}\log N)$. Overall, the prover runs in time $\mathcal{O}(N)$, with $\mathcal{O}(N)$ field operations and $\mathcal{O}(N/b+\sqrt{N}\log N)$ cryptographic operations.
- *Proof size.* In *Step 1*, each $\mathcal{V}_i$ receives $\mathbf{C}_{\lfloor i/\mu b\rfloor}$ of size $\mathcal{O}(b)$ and the batch proof $\pi_{\mathbf{C}_{\lfloor i/\mu b\rfloor}}$ of size $\mathcal{O}(\log N)$. In *Step 2*, it receives $\log N$ commitments, $\log N$ claimed evaluations and an opening proof of the PC scheme. Since the PC scheme has a proof size of $\mathcal{O}(\log N)$, the proof size would be $\mathcal{O}(\log N+b)$ for the interactive version and $\mathcal{O}(\log N+b)$ for the non-interactive version.
- *Verification time.* In *Step 1*, each $\mathcal{V}_i$ runs the FC.BVerify algorithm to verify the correctness of $\mathbf{C}_{\lfloor i/\mu b\rfloor}$, which takes $\mathcal{O}(b\log\mu)$ time. In *Step 2*, it needs to (1) compute the final evaluation and commitment, and (2) verify the final evaluation. Since the underlying PC scheme has a verification time of $\mathcal{O}(\log N)$, the complexities of (1) and (2) are $\mathcal{O}(\log\mu)$ and $\mathcal{O}(\log N)$, respectively. Thus, the verification time would be $\mathcal{O}(b\log N)$ for both the interactive and non-interactive versions.

In conclusion, FlexProofs is a VC scheme with $\mathcal{O}(N)$ Commit and OpenAll complexity, $\mathcal{O}(\log N+b)$ proof size and $\mathcal{O}(b\log N)$ verification time.

4.4 Compatible with a family of zkSNARKs

Paper [54] proposes a construction that combines a VC scheme with a zkSNARK in order to prove the correctness of computations over data originating from multiple sources. To adapt FlexProofs to this construction, we first extend our

OpenAll for sub-arrays and then show how FlexProofs is directly compatible with a family of zkSNARKs.

Extending *OpenAll* for sub-arrays. The above *OpenAll* protocol proves the correctness of all individual elements of $\mathbf{m}$. We now extend this to proving the correctness of all *consecutive sub-arrays*. More formally, for each verifier $\mathcal{V}_j$ with $j \in [0, M)$, we aim to prove that its array $\mathbf{m}_j \in \mathbb{F}^{N/M}$ is indeed the j-th sub-array of the committed vector $\mathbf{m} \in \mathbb{F}^N$. This can be achieved by showing that the commitment C_j of $\mathbf{m}_j$ is the j-th element of the committed vector $\mathbf{C}$, as established in *Step 1* of *OpenAll*. To generate all opening proofs in this setting, all parties follow the $\mathsf{VC.Commit}$ algorithm and *Step 1*, with the following modifications. In the $\mathsf{VC.Commit}$ algorithm, partition $\mathbf{m}$ into M segments of size N/M such that $\mathbf{m} = (\mathbf{m}_0, \ldots, \mathbf{m}_{M-1})$; for each $\mathbf{m}_j$, compute its multi-linear extension commitment C_j; then compute the commitment of the vector $\mathbf{C} = (C_0, \ldots, C_{M-1})$. Upon receiving C_j, verifier $\mathcal{V}_j$ checks whether the polynomial committed in C_j corresponds to the multi-linear extension of $\mathbf{m}_j$.

Compatibility with a family of zkSNARKs. According to [54], if a VC scheme encodes the committed vector $\mathbf{m}$ as a multi-linear polynomial and supports generation of an evaluation proof for the polynomial at a random point, it is directly compatible with a family of zkSNARKs that encode inputs as multi-linear polynomials [71,59,19]. Since FlexProofs is built on PC and FC schemes, it can be viewed as encoding the committed vector $\mathbf{m}$ into a multi-linear polynomial and supports generating an evaluation proof for the committed polynomial at a random point. We next show how FlexProofs produces such a proof. By Eq. (1), the multi-linear extension of a vector $\mathbf{m} \in \mathbb{F}_p^N$ is

$$f_{\mathbf{m}}(\mathbf{x}) = \sum_{i \in [0,N)} \mathbf{m}[i] \prod_{k \in [0, \log N)} \left(i_k x_k + (1 - i_k)(1 - x_k)\right), \tag{14}$$

where $\mathbf{x} = (x_{\log N - 1}, \ldots, x_0)$ and $\mathsf{Bin}(i) = (i_{\log N - 1}, \ldots, i_0)$; and for $j \in [0, \mu)$, the multi-linear extension of $\mathbf{m}_j \in \mathbb{F}_p^{\mu}$ is

$$f_j(\mathbf{x}_R) = \sum_{a \in [0,\mu)} \mathbf{m}_j[a] \prod_{k \in [0, \log \mu)} \left(a_k x_k + (1 - a_k)(1 - x_k)\right), \tag{15}$$

where $\mathsf{Bin}(a) = (a_{\log \mu - 1}, \ldots, a_0)$. From Eq. (14) and (15), we have

$$f_{\mathbf{m}}(\mathbf{x}) = \sum\nolimits_{j \in [0,\mu)} f_j(\mathbf{x}_R) \mathcal{T}_{j, \log \mu}(\mathbf{x}_L), \tag{16}$$

where $\mathcal{T}_{j,\log\mu}(\mathbf{x}_L) = \prod_{a \in [0, \log \mu)} (j_a x_{a + \log \mu} + (1 - j_a)(1 - x_{a + \log \mu}))$ and $\mathsf{Bin}(j) = (j_{\log \mu - 1}, \ldots, j_0)$. To prove the evaluation of $f_{\mathbf{m}}(\mathbf{x})$ at $\mathbf{r} = (r_{\log N - 1}, \ldots, r_0) \in \mathbb{F}_p^{\log N}$, it suffices to prove the evaluation of $f_{\mathbf{m}}(\mathbf{r}_L, \mathbf{x}_R)$ at $\mathbf{r}_R$. Let us first define $F(\mathbf{x}_R) = f_{\mathbf{m}}(\mathbf{r}_L, \mathbf{x}_R)$; using Eq. (16), we then obtain the expression:

$$F(\mathbf{x}_R) = \sum\nolimits_{j \in [0,\mu)} f_j(\mathbf{x}_R) \mathcal{T}_{j, \log \mu}(\mathbf{r}_L).$$

Thus, the original problem reduces to proving the evaluation of $F(\mathbf{x}_R)$ at $\mathbf{r}_R$, which can be accomplished through the following steps:

1. Compute the commitment to $F(\mathbf{x}_R)$ as $C_F = \prod_{j\in[0,\mu)} C_j^{\mathcal{T}_{j,\log\mu}(\mathbf{r}_L)}$.
2. Prove the correctness of C_F using our FC scheme as C_F can be expressed as $\langle \mathbf{C}, (\mathcal{T}_{0,\log\mu}(\mathbf{r}_L), \ldots, \mathcal{T}_{\mu-1,\log\mu}(\mathbf{r}_L))\rangle$, where $\mathbf{C}$ is already committed in C.
3. Generate a proof for the evaluation of F at $\mathbf{r}_R$ with the PC scheme.

Therefore, to prove the evaluation of $f(\mathbf{x})$ at a random point $\mathbf{r}$, the prover sends C_F along with proofs generated by the FC and PC schemes to the verifier, who then checks both proofs. The above process can be adapted into a zero-knowledge version via standard techniques [20,40]. Here, the underlying PC is replaced with its zero-knowledge variant, while the FC component does not require zero-knowledge, as it only operates on hidden commitments.

5 Performance Evaluation

In this section, we implement the non-interactive variants of FlexProofs [1] in the single-thread and evaluate their performance. For field and group operations, we use the mcl library and choose BN_SNARK1 elliptic curve to achieve approximately 100 bits of security. Our codes are run on an Intel(R) Xeon(R) E-2286G CPU @ 4.0GHz with 6 cores and 64GB memory. Unless otherwise stated, we run each experiment 3 times and report their average.

5.1 Evaluation of our FC scheme

We evaluate our FC scheme for $n \in \{2^8, 2^9, 2^{10}, 2^{11}, 2^{12}\}$, and show the results in Table 2.

Table 2: Single-threaded running time of our FC scheme.

n	2^8	2^9	2^{10}	2^{11}	2^{12}
$\lvert\pi_y\rvert$ (KiB)	6.66	7.47	8.28	9.09	9.91
FC.Commit (s)	0.03	0.07	0.14	0.27	0.54
FC.BOpen($t=1$) (s)	0.12	0.23	0.45	0.88	1.70
FC.BVerify($t=1$) (s)	0.004	0.005	0.005	0.006	0.007
FC.BOpen($t=32$) (s)	0.13	0.23	0.46	0.89	1.73
FC.BVerify($t=32$) (s)	0.007	0.01	0.02	0.03	0.05

Proof size. The proof size grows with $\log n$, and is about 9.91 KiB for $n = 2^{12}$.

Commit. As shown in Table 2, the time costs of FC.Commit are roughly proportional to n. For $n = 2^{12}$, committing needs 0.54s.

Batch opening. For $n = 2^{12}$, the running time of FC.BOpen($t = 32$) is approximately 1.73 s, which is only 3% of the total time required to run FC.BOpen($t = 1$) 32 times. Figure 2 compares the running time of a single FC.BOpen call with batch size t (denoted FC.BOpen-t) against t sequential calls with batch size 1 (denoted $t \times$ FC.BOpen-1), for $n = 2^{12}$ and $t \in \{16, 32, 64, 128, 256, 512\}$. And it shows that the efficiency gain of batching increases with the batch size t.

[1] https://github.com/FlexProofs

Batch verification. For $n = 2^{12}$, the running time of $\mathsf{FC.BVerify}(t = 32)$ is approximately 0.05 s, which is only 22% of the total time required to run $\mathsf{FC.BVerify}(t = 1)$ 32 times. This demonstrates the high efficiency of our proposed batching technique. Figure 3 compares the running time of a single $\mathsf{FC.BVerify}$ call with batch size t (denoted $\mathsf{FC.BVerify}$-t) against t sequential calls with batch size 1 (denoted $t \times \mathsf{FC.BVerify}$-1), for $n = 2^{12}$ and $t \in \{16, 32, 64, 128, 256, 512\}$.

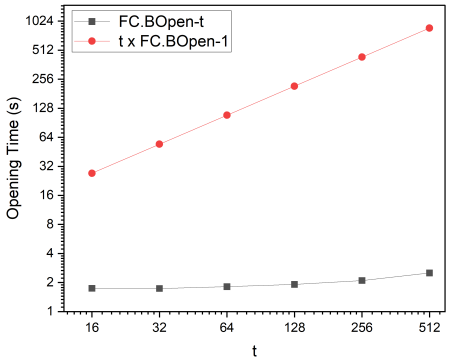

Fig. 2: Opening time. Both axes use logarithmic scales.

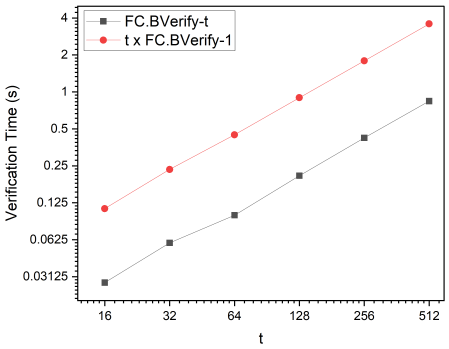

Fig. 3: Verification time. Both axes use logarithmic scales.

5.2 Evaluation of FlexProofs and comparison with HydraProofs

We microbenchmark FlexProofs with batch sizes $b \in \{2\log N, \log^2 N\}$. We also compared it with HydraProofs [54], the state-of-the-art vector commitment scheme. HydraProofs is the only known VC construction that achieves $\mathcal{O}(N)$ time to generate all opening proofs for a vector of size N, while also being directly compatible with a family of zkSNARKs. For fairness, we implement both schemes in C++ using the same elliptic curve and techniques for $N \in \{2^{16}, 2^{18}, 2^{20}, 2^{22}, 2^{24}\}$. The experimental results are shown in Table 3.

Commit. In FlexProofs, the commitment time grows roughly linearly with N, taking about 0.82 seconds when $N = 2^{16}$. HydraProofs shows a similar linear growth in commitment time.

Computing all proofs. In FlexProofs, computing all proofs requires $\mathcal{O}(N)$ field operations and $\mathcal{O}(N/b + \sqrt{N}\log N)$ cryptographic operations, so the proving time decreases as b grows. Experiments confirm this: when $N = 2^{16}$, computing all proofs takes 1.03 seconds for $b = 2\log N$, but only 0.23 seconds for $b = \log^2 N$. In HydraProofs, computing all proofs requires $\mathcal{O}(N)$ field operations and $\mathcal{O}(N)$ cryptographic operations. Thus, FlexProofs requires fewer cryptographic operations as b increases. Consequently, when $N = 2^{16}$, FlexProofs is about 1.3× faster than HydraProofs with $b = 2\log N$, and about 6× faster with $b = \log^2 N$.

Verifying a proof. When $b = 2\log N$, the verification complexity in FlexProofs is $\mathcal{O}(\log^2 N)$, whereas for $b = \log^2 N$ it increases to $\mathcal{O}(\log^3 N)$. Experimental

Table 3: The comparison between FlexProofs (FP) and HydraProofs.

N		2^{16}	2^{18}	2^{20}	2^{22}	2^{24}
Commit (s)	FP ($b = 2\log N$)	0.83	3.02	8.81	31.87	116.61
	FP ($b = \log^2 N$)	0.82	3.01	8.70	31.53	116.41
	HydraProofs	0.82	3.01	8.66	31.71	116.64
Compute all proofs (s)	FP ($b = 2\log N$)	1.03	3.58	12.17	43.64	159.30
	FP ($b = \log^2 N$)	**0.23**	**0.82**	**2.61**	**9.24**	**32.71**
	HydraProofs	1.42	4.82	17.15	58.66	210.07
Verify a proof (s)	FP ($b = 2\log N$)	**0.006**	**0.007**	**0.008**	**0.009**	**0.011**
	FP ($b = \log^2 N$)	0.01	0.012	0.013	0.015	0.017
	HydraProofs	0.01	0.011	0.012	0.013	0.014
Proof size (KiB)	FP ($b = 2\log N$)	8.91	10	11.09	12.19	13.28
	FP ($b = \log^2 N$)	15.91	19	22.34	25.94	29.78
	HydraProofs	**5.53**	**6.63**	**7.81**	**9.09**	**10.47**

results confirm this: when $N = 2^{16}$, verifying a proof takes 0.006 seconds for $b = 2\log N$, and 0.01 seconds for $b = \log^2 N$. In HydraProofs, the verification complexity is also $\mathcal{O}(\log^2 N)$. And the experimental results show that when $b = 2\log N$, FlexProofs verifies proofs slightly faster than HydraProofs.

Proof size. When $b = 2\log N$, the proof size in FlexProofs is $\mathcal{O}(\log N)$, while for $b = \log^2 N$ it grows to $\mathcal{O}(\log^2 N)$. In HydraProofs, the proof size has complexity $\mathcal{O}(\log^2 N)$. For the values of N shown in Table 3, HydraProofs yields smaller proofs; for instance, when $N = 2^{16}$ it requires only 5.53 KiB compared to 8.91 KiB in our scheme. This gap arises because asymptotic bounds capture only the growth trend as N increases, while for moderate N the hidden constants dominate the concrete size. As N becomes larger, the slower asymptotic growth of FlexProofs is expected to eventually result in smaller proofs.

6 Conclusions

We first propose an FC scheme for multi-exponentiations with batch opening. Based on this, we construct FlexProofs, a VC scheme that can generate all proofs for a vector of size N in optimal time $\mathcal{O}(N)$ (incurred by $\mathcal{O}(N)$ field operations and $\mathcal{O}(N/b + \sqrt{N}\log N)$ cryptographic operations) and is directly compatible with zkSNARKs, where batch size b ranges from 1 to $\sqrt{N}$. Finally, FlexProofs, combined with suitable zkSNARKs, enables applications such as VSS and VRA.

Acknowledgments

The authors would like to thank the anonymous shepherd and reviewers for their insightful comments. The corresponding author Liang Feng Zhang's research is partially supported by the National Natural Science Foundation of China (Grant No. 62372299).

References

1. Abbaszadeh, K., Pappas, C., Katz, J., Papadopoulos, D.: Zero-knowledge proofs of training for deep neural networks. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. pp. 4316–4330 (2024)
2. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. Journal of Cryptology **29**(2), 363–421 (2016)
3. Balbás, D., Catalano, D., Fiore, D., Lai, R.W.: Functional commitments for circuits from falsifiable assumptions. IACR Cryptol. ePrint Arch. **2022**, 1365 (2022)
4. Bekkerman, R., Bilenko, M., Langford, J.: Scaling up machine learning: Parallel and distributed approaches. Cambridge University Press (2011)
5. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security. pp. 62–73 (1993)
6. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Theory of Cryptography Conference. pp. 31–60. Springer (2016)
7. Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable delegation of computation over large datasets. In: Annual Cryptology Conference. pp. 111–131. Springer (2011)
8. Biggio, B., Nelson, B., Laskov, P.: Poisoning attacks against support vector machines. arXiv preprint arXiv:1206.6389 (2012)
9. Boneh, D., Boyen, X.: Short signatures without random oracles and the sdh assumption in bilinear groups. Journal of cryptology **21**(2), 149–177 (2008)
10. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo infinite: Proof-carrying data from additive polynomial commitments. In: Annual International Cryptology Conference. pp. 649–680. Springer (2021)
11. Boneh, D., Nguyen, W., Ozdemir, A.: Efficient functional commitments: How to commit to a private function. Cryptology ePrint Archive (2021)
12. Bünz, B., Maller, M., Mishra, P., Tyagi, N., Vesely, P.: Proofs for inner pairing products and applications. In: ASIACRYPT 2021. pp. 65–97. Springer (2021)
13. Campanelli, M., Fiore, D., Querol, A.: Legosnark: Modular design and composition of succinct zero-knowledge proofs. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2075–2092 (2019)
14. Campanelli, M., Nitulescu, A., Ràfols, C., Zacharakis, A., Zapico, A.: Linear-map vector commitments and their practical applications. In: ASIACRYPT 2022. pp. 189–219. Springer (2022)
15. Cao, X., Fang, M., Liu, J., Gong, N.Z.: Fltrust: Byzantine-robust federated learning via trust bootstrapping. arXiv preprint arXiv:2012.13995 (2020)
16. de Castro, L., Peikert, C.: Functional commitments for all functions, with transparent setup and from sis. In: EUROCRYPT 2023. pp. 287–320. Springer (2023)
17. Catalano, D., Fiore, D.: Vector commitments and their applications. In: PKC 2013. pp. 55–72. Springer (2013)
18. Catalano, D., Fiore, D., Tucker, I.: Additive-homomorphic functional commitments and applications to homomorphic signatures. In: ASIACRYPT 2022. pp. 159–188. Springer (2022)
19. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 499–530. Springer (2023)

20. Chiesa, A., Forbes, M.A., Spooner, N.: A zero knowledge sumcheck and its applications. arXiv preprint arXiv:1704.02086 (2017)
21. Chor, B., Goldwasser, S., Micali, S., Awerbuch, B.: Verifiable secret sharing and achieving simultaneity in the presence of faults. In: 26th Annual Symposium on Foundations of Computer Science (sfcs 1985). pp. 383–395. IEEE (1985)
22. Chu, H., Fiore, D., Kolonelos, D., Schröder, D.: Inner product functional commitments with constant-size public parameters and openings. In: SCN 2022. pp. 639–662. Springer (2022)
23. CNBC: Amazon has been promoting its own products at the bottom of competitors' listings (Oct 2018), https://www.cnbc.com/2018/10/02/amazon-is-testing-a-new-feature-that-promotes-its-private-label-brands-inside-a-competitors-product-listing.html
24. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: 28th Annual Symposium on Foundations of Computer Science (sfcs 1987). pp. 427–438. IEEE (1987)
25. Fiore, D., Fournet, C., Ghosh, E., Kohlweiss, M., Ohrimenko, O., Parno, B.: Hash first, argue later: Adaptive verifiable computations on outsourced data. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1304–1316 (2016)
26. Fung, C., Yoon, C.J., Beschastnikh, I.: Mitigating sybils in federated learning poisoning. arXiv preprint arXiv:1808.04866 (2018)
27. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive (2019)
28. Gorbunov, S., Reyzin, L., Wee, H., Zhang, Z.: Pointproofs: Aggregating proofs for multiple vector commitments. In: CCS '20. pp. 2007–2023 (2020)
29. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for {Zero-Knowledge} proof systems. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 519–535 (2021)
30. Groth, J.: On the size of pairing-based non-interactive arguments. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 305–326. Springer (2016)
31. Grubbs, P., Arun, A., Zhang, Y., Bonneau, J., Walfish, M.: {Zero-Knowledge} middleboxes. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 4255–4272 (2022)
32. Howe, J., et al.: The rise of crowdsourcing. Wired magazine **14**(6), 176–183 (2006)
33. Hu, R., Guo, Y., Pan, M., Gong, Y.: Targeted poisoning attacks on social recommender systems. In: 2019 IEEE Global Communications Conference (GLOBECOM). pp. 1–6. IEEE (2019)
34. Kasyap, H., Tripathy, S.: Hidden vulnerabilities in cosine similarity based poisoning defense. In: 2022 56th Annual Conference on Information Sciences and Systems (CISS). pp. 263–268. IEEE (2022)
35. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: ASIACRYPT 2010. pp. 177–194. Springer (2010)
36. Kilian, J.: A note on efficient zero-knowledge proofs and arguments. In: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. pp. 723–732 (1992)
37. Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)

38. Kourtellis, N., Katevas, K., Perino, D.: Flaas: Federated learning as a service. In: Proceedings of the 1st workshop on distributed machine learning. pp. 7–13 (2020)
39. Lai, R.W., Malavolta, G.: Subvector commitments with application to succinct arguments. In: CRYPTO 2019. pp. 530–560. Springer (2019)
40. Lee, J.: Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In: Theory of cryptography conference. pp. 1–34. Springer (2021)
41. Libert, B., Ramanna, S.C., Yung, M.: Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In: ICALP 2016 (2016)
42. Libert, B., Yung, M.: Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In: TCC 2010. pp. 499–517. Springer (2010)
43. Lipmaa, H., Pavlyk, K.: Succinct functional commitment for a large class of arithmetic circuits. In: ASIACRYPT 2020. pp. 686–716. Springer (2020)
44. Liu, J., Zhang, L.F.: Matproofs: Maintainable matrix commitment with efficient aggregation. In: CCS '22. pp. 2041–2054 (2022)
45. Liu, T., Xie, T., Zhang, J., Song, D., Zhang, Y.: Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 1777–1793. IEEE (2024)
46. Liu, T., Xie, X., Zhang, Y.: Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 2968–2985 (2021)
47. Liu, X., Yang, X., Wang, Y., Zhang, X., Yang, X.: Evaluate and guard the wisdom of crowds: Zero knowledge proofs for crowdsourcing truth inference. arXiv preprint arXiv:2308.00985 (2023)
48. Luo, G., Fu, S., Gong, G.: Updatable linear map commitments and their applications in elementary databases. In: PST 2021. pp. 1–6. IEEE (2021)
49. Ma, Z., Ma, J., Miao, Y., Li, Y., Deng, R.H.: Shieldfl: Mitigating model poisoning attacks in privacy-preserving federated learning. IEEE Transactions on Information Forensics and Security **17**, 1639–1654 (2022)
50. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics. pp. 1273–1282. PMLR (2017)
51. Merkle, R.C.: A digital signature based on a conventional encryption function. In: CRYPTO 1987. pp. 369–378. Springer (1987)
52. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: TCC 2013. pp. 222–242. Springer (2013)
53. Papamanthou, C., Shi, E., Tamassia, R., Yi, K.: Streaming authenticated data structures. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 353–370. Springer (2013)
54. Pappas, C., Papadopoulos, D., Papamanthou, C.: Hydraproofs: Optimally computing all proofs in a vector commitment (with applications to efficient zksnarks over data from multiple users). In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 3421–3439. IEEE (2025)
55. Peikert, C., Pepin, Z., Sharp, C.: Vector and functional commitments from lattices. In: TCC 2021. pp. 480–511. Springer (2021)
56. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE symposium on security and privacy. pp. 459–474. IEEE (2014)

57. Schafer, J.B., Frankowski, D., Herlocker, J., Sen, S.: Collaborative filtering recommender systems. In: The adaptive web: methods and strategies of web personalization, pp. 291–324. Springer (2007)
58. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. J. ACM **27**(4), 701–717 (1980)
59. Setty, S.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: Annual International Cryptology Conference. pp. 704–737. Springer (2020)
60. Sha, J., Liu, S., Han, S.: Functional commitments for arbitrary circuits of bounded sizes. Designs, Codes and Cryptography pp. 1–35 (2024)
61. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
62. Srinivasan, S., Chepurnoy, A., Papamanthou, C., Tomescu, A., Zhang, Y.: Hyperproofs: Aggregating and maintaining proofs in vector commitments. In: USENIX Security 22. pp. 3001–3018 (2022)
63. Tan, B., Liu, B., Zheng, V., Yang, Q.: A federated recommender system for online services. In: Proceedings of the 14th ACM conference on recommender systems. pp. 579–581 (2020)
64. The Guardian: Revealed: the facebook loophole that lets world leaders deceive and harass their citizens (Apr 2021), https://www.theguardian.com/technology/2021/apr/12/facebook-loophole-state-backed-manipulation
65. Tomescu, A., Abraham, I., Buterin, V., Drake, J., Feist, D., Khovratovich, D.: Aggregatable subvector commitments for stateless cryptocurrencies. In: SCN 2020. pp. 45–64. Springer (2020)
66. Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zksnarks without trusted setup. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 926–943. IEEE (2018)
67. Wang, W., Ulichney, A., Papamanthou, C.: {BalanceProofs}: Maintainable vector commitments with fast aggregation. In: USENIX Security 23. pp. 4409–4426 (2023)
68. Wee, H., Wu, D.J.: Lattice-based functional commitments: Fast verification and cryptanalysis. In: ASIACRYPT 2023. pp. 201–235. Springer (2023)
69. Wee, H., Wu, D.J.: Succinct vector, polynomial, and functional commitments from lattices. In: EUROCRYPT 2023. pp. 385–416. Springer (2023)
70. Wee, H., Wu, D.J.: Succinct functional commitments for circuits from k-lin. In: EUROCRYPT 2024. pp. 280–310. Springer (2024)
71. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Annual International Cryptology Conference. pp. 733–764. Springer (2019)
72. Xu, G., Li, H., Liu, S., Yang, K., Lin, X.: Verifynet: Secure and verifiable federated learning. IEEE Transactions on Information Forensics and Security **15**, 911–926 (2019)
73. Yang, L., Tan, B., Zheng, V.W., Chen, K., Yang, Q.: Federated recommendation systems. In: Federated Learning: Privacy and Incentive, pp. 225–239. Springer (2020)
74. Zhang, J., Xie, T., Hoang, T., Shi, E., Zhang, Y.: Polynomial commitment with a {One-to-Many} prover and applications. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 2965–2982 (2022)
75. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 859–876. IEEE (2020)

76. Zhang, Z., Li, W., Guo, Y., Shi, K., Chow, S.S., Liu, X., Dong, J.: Fast {RS-IOP} multivariate polynomial commitments and verifiable secret sharing. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 3187–3204 (2024)
77. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: EUROSAM 1979. pp. 216–226. Springer (1979)

A Definitions

Definition 7 (Zero-Knowledge of Polynomial Commitment). *A polynomial commitment (PC) scheme is zero-knowledge if for any λ, n, d, adversary $\mathcal{A}$, and simulator $\mathcal{S}$, we have:*

$$\Pr\left(Real_{\mathcal{A},f}(1^\lambda)=1\right) \approx \Pr\left(Ideal_{\mathcal{A},\mathcal{S}}(1^\lambda)=1\right).$$

$\mathbf{Real}_{\mathcal{A},f}(1^\lambda)$:	$\boldsymbol{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$:
1. $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, 1^n, 1^d)$ 2. $C_f \leftarrow \mathsf{PC.Commit}(f, r_f)$ 3. $m \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}, C_f)$ 4. For each step $j \in \{2,3,\ldots,m\}$: (a) $\mathbf{x}_j \leftarrow \mathcal{A}(1^\lambda, C_f, y_1, \ldots, y_{j-1}, \pi_1, \ldots, \pi_{j-1}, \mathsf{pp})$ (b) $y_j, \pi_j \leftarrow \mathsf{PC.Eval}(f, x_j)$ 5. Output $b \leftarrow \mathcal{A}(1^\lambda, C_f, y_1, \ldots, y_m, \pi_1, \ldots, \pi_m, \mathsf{pp})$	*1.* $(C_f, \mathsf{pp}, t) \leftarrow \mathcal{S}(1^\lambda, 1^n)$ *2.* $m \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}, C_f)$ *3. For each step $j \in \{2,3,\ldots,m\}$:* *(a)* $\mathbf{x}_j \leftarrow \mathcal{A}(1^\lambda, C_f, y_1, \ldots, y_{j-1}, \pi_1, \ldots, \pi_{j-1}, \mathsf{pp})$ *(b)* $y_j, \pi_j \leftarrow \mathcal{S}(\mathsf{pp}, f, \mathbf{x}_j)$ *4. Output* $b \leftarrow \mathcal{A}(1^\lambda, C_f, y_1, \ldots, y_m, \pi_1, \ldots, \pi_m, \mathsf{pp})$

Definition 8 (Position Hiding of Vector Commitment). *A vector commitment (VC) scheme is position hiding if for any PPT adversary $\mathcal{A}$ and simulator $\mathcal{S}$, the following holds:*

$$\Pr\left(Real_{\mathcal{A},\mathbf{m}}(1^\lambda)=1\right) \approx \Pr\left(Ideal_{\mathcal{A},\mathcal{S}}(1^\lambda)=1\right).$$

$\mathbf{Real}_{\mathcal{A},\mathbf{m}}(1^\lambda)$:	$\boldsymbol{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$:
1. $\mathsf{pp} \leftarrow \mathsf{VC.Setup}(1^\lambda, 1^N)$ 2. $C \leftarrow \mathsf{VC.Commit}(\mathbf{m})$ 3. $i_1, \ldots, i_n, m_{i_1}, \ldots, m_{i_n} \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}, C)$ (for $n < N$) 4. $\{\pi_i\}_{i=1}^N \leftarrow \mathsf{VC.OpenAll}(\mathbf{m})$. Send $\{\pi_{i_1}, \ldots, \pi_{i_n}\}$ to $\mathcal{A}$ 5. Output $b \leftarrow \mathcal{A}(1^\lambda, C, m_{i_1}, \ldots, m_{i_n}, \pi_{i_1}, \ldots, \pi_{i_n}, \mathsf{pp})$	*1.* $(C, \mathsf{pp}, trap) \leftarrow \mathcal{S}(1^\lambda, N)$ *2.* $i_1, \ldots, i_n, m_{i_1}, \ldots, m_{i_n} \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}, C)$ *(for $n < N$)* *3.* $\{\pi_{i_1}, \ldots, \pi_{i_n}\} \leftarrow \mathcal{S}(i_1, \ldots, i_n, m_{i_1}, \ldots, m_{i_n}, trap, \mathsf{pp})$ *4. Output* $b \leftarrow \mathcal{A}(1^\lambda, C, m_{i_1}, \ldots, m_{i_n}, \pi_{i_1}, \ldots, \pi_{i_n}, \mathsf{pp})$

B Our FC Scheme

Assumption 1 (q-Strong Diffie-Hellman assumption (q-SDH)[9]) *Given a security parameter λ, algorithm BG outputs $\mathbf{bg} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e, g_1, g_2)$. The*

q-SDH assumption holds relative to BG *if for any efficient algorithm* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{c}\mathbf{bg} \leftarrow \mathsf{BG}(1^\lambda), \gamma \xleftarrow{\$} \mathbb{F}_p, \mathsf{pp} = (\mathbf{bg}, g_1^\gamma, \ldots, g_1^{\gamma^q}, g_2^\gamma) : \\ (a, g_1^{1/(\gamma+a)}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp})\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Assumption 2 (*q*-Auxiliary Structured Double Pairing assumption) (*q*-ASDBP) *Given a security parameter* λ*,* BG *output* $\mathbf{bg} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e, g_1, g_2)$*. The q-ASDBP assumption holds relative to* BG *if for any efficient algorithm* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{c}\mathbf{bg} \leftarrow \mathsf{BG}(1^\lambda),\ \beta \xleftarrow{\$} \mathbb{F}_p, (A_0, \ldots, A_{q-1}) \leftarrow \mathcal{A}(\mathbf{bg}, g_1^\beta, \{g_2^{\beta^{2i}}\}_{i=1}^{q-1}) : \\ ((A_0, \ldots, A_{q-1}) \neq \mathbf{1}_{\mathbb{G}_1}) \wedge (1_{\mathbb{G}_\mathrm{T}} = \prod_{i=0}^{q-1} e(A_i, g_2^{\beta^{2i}}))\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Lemma 1. *Let* $\mathbf{A} \in \mathbb{G}_1^n$*. Let* $\mathbf{b}^{(i)} \in \mathbb{F}_p^n$ *and* $y_i \in \mathbb{G}_1$ *for* $i \in [0, t)$*. Assume each* $r_i (i \in [0, t))$ *is chosen uniformly at random from* $\mathbb{F}_p$*. Then with probability at least* $1 - 1/p$*, all Eq.* (2) *are satisfied iff Eq.* (3) *is satisfied.*

Proof. Clearly, if Eq. (2) holds, then Eq. (3) holds. For the other direction, assume the correct value of $\mathbf{A}^{\mathbf{b}^{(i)}}$ is $\hat{y}_i$ for $i \in [0, t)$, then there exists at least one j such that $y_j \neq \hat{y}_j$ and $\prod_{i\in[0,t)} y_i^{r_i} = \prod_{i\in[0,t)} \hat{y}_i^{r_i}$. Define the $a_i = \log_{g_1} y_i/\hat{y}_i$, then there will be at least one j for which $a_j \neq 0$ and $\sum_{i\in[0,t)} a_i r_i = 0 \mod p$. By the Schwartz-Zippel lemma [58,77], this occurs with probability $\leq 1/p$. Therefore, if Eq. (2) is not fully satisfied, Eq. (3) holds with probability at most $1/p$. □

Non-interactive argument of knowledge in the ROM. Bünz et al. [12] define a non-interactive argument of knowledge in the random oracle model (ROM). In ROM, a hash function is replaced by a random function which is sampled from the space of all random functions $\rho \leftarrow \mathcal{U}(1^\lambda)$. The non-interactive argument is an argument system where the prover sends a single message π, and the verifier using the proof accepts or rejects. Both the prover and verifier have access to a random oracle ρ. An argument of knowledge in the ROM has the property that for each convincing prover there exists an extractor which can rewind the prover and reinitialize the random oracle with new randomness.

Definition 9. (Non-interactive argument of knowledge in the ROM, from [12]). *A non-interactive argument is an argument of knowledge for a relation* $\mathcal{R}$ *with knowledge error* $\kappa(\lambda)$ *if for every adversary* $\widetilde{\mathcal{P}}$ *there exists an extractor* $\mathcal{E}$ *such that*

$$\Pr\left[\begin{array}{c}\rho \leftarrow \mathcal{U}(1^\lambda),\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda), (\mathbb{x}, \pi) \leftarrow \widetilde{\mathcal{P}}^\rho(\mathsf{crs}), \\ \mathbb{w} \leftarrow \mathcal{E}^{\widetilde{\mathcal{P}}, \rho}(\mathsf{crs}, \mathbb{x}, \pi) : (\mathcal{V}^\rho(\mathsf{crs}, \mathbb{x}, \pi) = 1) \wedge ((\mathbb{x}, \mathbb{w}) \notin \mathcal{R})\end{array}\right] \leq \kappa(\lambda).$$

The $\mathcal{E}$ *can rewind the prover and reinitialize (but not program) the random oracle.*

B.1 Function Binding

Before proving the function binding of our FC scheme, we first show that our FC scheme is a non-interactive argument of knowledge in the ROM for the relation

$\mathcal{R}_B = \{(C \in \mathbb{G}_\mathrm{T}, \{\mathbf{b}^{(i)}\}_{i\in[0,t)} \subset \mathbb{F}_p^n, \{y_i\}_{i\in[0,t)} \subset \mathbb{G}_1, \mathbf{v} \in \mathbb{G}_2^n; \mathbf{A} \in \mathbb{G}_1^n) : (C = \mathbf{A}*\mathbf{v}) \wedge (\forall i \in [0,t), y_i = \langle \mathbf{A}, \mathbf{b}^{(i)}\rangle)\}$. That is, we show that for any PPT adversary $\mathcal{A}_B$, we can construct a PPT extractor $\mathcal{E}_B$ that extracts the vector $\mathbf{A}$ such that

$$\Pr\left[\begin{array}{c}\rho \leftarrow \mathcal{U}(1^\lambda),\ \mathsf{pp}(=\mathbf{v}) \leftarrow \mathsf{FC.Setup}(1^\lambda, 1^n),\\ (C, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)}, \pi_y) \leftarrow \mathcal{A}_B^\rho(\mathsf{pp}),\\ \mathbf{A} \leftarrow \mathcal{E}_B^{\mathcal{A}_B,\rho}(\mathsf{pp}, C, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)}, \pi_y) :\\ (\mathsf{FC.BVerify}^\rho(C, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)}, \pi_y) = 1) \wedge\\ ((C, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)}, \mathbf{v}; \mathbf{A}) \notin \mathcal{R}_B)\end{array}\right] \leq \mathsf{negl}(\lambda).$$

To prove this, we first prove the knowledge soundness of the FC.BOpen and FC.BVerify algorithms in the single-instance case (specifically, $t = 1$) and without the random value r_0.

Knowledge soundness in the single-instance case and without the random value. For simplicity, we denote the FC.BOpen and FC.BVerify algorithms in the single-instance case (specifically, $t = 1$) and without the random value r_0 as FC.Open and FC.Verify. Note that in FC.Open and FC.Verify, $\mathbf{b} = \mathbf{b}^{(0)}$ and $y = y_0$. We will show that the scheme consisting of algorithms FC.Setup, FC.Commit, FC.Open and FC.Verify is a non-interactive argument of knowledge in the ROM for the relation $\mathcal{R}_S = \{(C \in \mathbb{G}_\mathrm{T}, \mathbf{b} \in \mathbb{F}_p^n, y \in \mathbb{G}_1, \mathbf{v} \in \mathbb{G}_2^n; \mathbf{A} \in \mathbb{G}_1^n) : C = \mathbf{A} * \mathbf{v} \wedge y = \langle \mathbf{A}, \mathbf{b}\rangle\}$. That is, we show that for any PPT adversary $\mathcal{A}_S$, we can construct a PPT extractor $\mathcal{E}_S$ that extracts the vector $\mathbf{A}$ such that

$$\Pr\left[\begin{array}{c}\rho \leftarrow \mathcal{U}(1^\lambda),\ \mathsf{pp}(=\mathbf{v}) \leftarrow \mathsf{FC.Setup}(1^\lambda, 1^n),\\ (C, \mathbf{b}, y, \pi_y) \leftarrow \mathcal{A}_S^\rho(\mathsf{pp}),\ \mathbf{A} \leftarrow \mathcal{E}_S^{\mathcal{A}_S,\rho}(\mathsf{pp}, C, \mathbf{b}, y, \pi_y) :\\ (\mathsf{FC.Verify}^\rho(C, \mathbf{b}, y, \pi_y) = 1) \wedge ((C, \mathbf{b}, y, \mathbf{v}; \mathbf{A}) \notin \mathcal{R}_S)\end{array}\right] \leq \mathsf{negl}(\lambda).$$

The existence of $\mathcal{E}_S$ is guaranteed by [12].

Constructing $\mathcal{E}_B$. $\mathcal{E}_B$ builds a adversary $\mathcal{A}_S$ against the extractability game of algorithms FC.Verify. $\mathcal{A}_S$ is given $(\mathsf{pp}, C, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)}, \pi_y)$, computes $\mathbf{b} = \sum_{i\in[0,t)} r_i \mathbf{b}^{(i)}$ and $y = \prod_{i\in[0,t)} y_i^{r_i}$, and output $(C, \mathbf{b}, y, \pi_y)$. Then $\mathcal{E}_B$ invokes $\mathcal{E}_S$ which outputs the vector $\mathbf{A}$ such that $C = \mathbf{A} * \mathbf{v}$ and $y = \langle \mathbf{A}, \mathbf{b}\rangle$. The equation $y = \langle \mathbf{A}, \mathbf{b}\rangle$ means that $\prod_{i\in[0,t)} y_i^{r_i} = \langle \mathbf{A}, \sum_{i\in[0,t)} r_i \mathbf{b}^{(i)}\rangle$. According to Lemma 1, with overwhelming probability, all $\{\langle \mathbf{A}, \mathbf{b}^{(i)}\rangle = y_i\}_{i\in[0,t)}$ are satisfied. Therefore, $\mathcal{E}_B$ can output vector $\mathbf{A}$ such that $(C, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)}, \mathbf{v}; \mathbf{A}) \in \mathcal{R}_B$ with overwhelming probability. The probability $\mathcal{E}_B$ fails is only negligible to the security parameter.

Proving Function Binding. From the above, we prove the function binding of our FC scheme as follows. Assuming that an adversary $\mathcal{A}$ outputs C, $\{\mathbf{b}^{(i)}, y_i, \hat{y}_i\}_{i\in[0,t)}, \pi_y, \pi_{\hat{y}}$ such that (1) $\mathsf{FC.BVerify}(C, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{y_i\}_{i\in[0,t)}, \pi_y) = 1$ and (2) $\mathsf{FC.BVerify}(C, \{\mathbf{b}^{(i)}\}_{i\in[0,t)}, \{\hat{y}_i\}_{i\in[0,t)}, \pi_{\hat{y}}) = 1$ with non-negligible probability. By the knowledge soundness of our FC scheme, equation (1) means that we can extract a vector $\mathbf{A}$ such that $C = \mathbf{A} * \mathbf{v}$ and $\{y_i = \langle \mathbf{A}, \mathbf{b}^{(i)}\rangle)\}_{i\in[0,t)}$, equation (2) means that we can extract a vector $\mathbf{A}'$ such that $C = \mathbf{A}' * \mathbf{v}$ and $\{\hat{y}_i =$

$\langle \mathbf{A}', \mathbf{b}^{(i)} \rangle)\}_{i\in[0,t)}$. Since there exists $j \in [0,t)$ such that $y_j \neq \hat{y}_j$, we have $\mathbf{A} \neq \mathbf{A}'$. Since $\mathbf{A} \neq \mathbf{A}'$ and $\mathbf{A} * \mathbf{v} = \mathbf{A}' * \mathbf{v}$, we construct another adversary $\mathcal{B}$ which outputs $(\mathbf{A}[i]/\mathbf{A}'[i])_{i\in[0,n)}$ which satisfy $(\mathbf{A}[0]/\mathbf{A}'[0], \ldots, \mathbf{A}[n-1]/\mathbf{A}'[n-1]) \neq \mathbf{1}_{\mathbb{G}_1} \wedge 1_{\mathbb{G}_T} = \prod_{i=0}^{n-1} e(\mathbf{A}[i]/\mathbf{A}'[i], h^{\beta^{2i}})$ with non-negligible probability, breaking n-ASDBP assumption.

C Correctness and Security Analysis of FlexProofs

Theorem 2. *FlexProofs satisfies the correctness property (Definition 5).*

Proof. We first prove that the interactive version of FlexProofs satisfies the correctness property. Specifically, we show that for any security parameter λ, any integer $N > 0$, any vector $\mathbf{m} \in \mathbb{F}_p^N$, and any index $i \in [0, N)$, it holds that:

$$\Pr\left[\begin{matrix}\mathsf{pp} \leftarrow \mathsf{VC.Setup}(1^\lambda, 1^N), (C, \mathsf{aux}) \leftarrow \mathsf{VC.Commit}(\mathbf{m}), \\ \langle \mathcal{P}_{OpenAll}(\mathsf{pp}, \mathsf{aux}, i, \mathbf{m}),\ \mathcal{V}_i(\mathsf{pp}, C, i, \mathbf{m}[i]) \rangle = 1\end{matrix}\right] = 1.$$

By Eq. (8) and (9), C is a commitment to a vector $\mathbf{C}(= (C_0, \ldots, C_{\mu-1}))$ under the FC scheme. Let $k = \lfloor i/(\mu b) \rfloor$ and $K = [kb, (k+1)b)$. According to Eq. (10), the proof $\pi_{\mathbf{C}_k}$ is a batch proof for $\{C_a = \langle \mathbf{C}, \mathbf{u}_a \rangle\}_{a\in K}$ (i.e., C_a is the a-th element of $\mathbf{C}$). By Definition 1, the correctness property of the FC scheme ensures that $\mathsf{FC.BVerify}(C, \{\mathbf{u}_a\}_{a\in K}, \{C_a\}_{a\in K}, \pi_{\mathbf{C}_k}) = 1$.

As per Eq. (11), the prover $\mathcal{P}_{OpenAll}$ random its values. For convenience, let $[i]_\mu = i \bmod \mu$. As per Eq. (12), the verifier $\mathcal{V}_i$ computes $D_{\lfloor i/\mu \rfloor}$ and $y_{\lfloor i/\mu \rfloor, [i]_\mu}$. As described in Step 2.2, the verifier also receives $\{D_w, y_{w,[i]_\mu}\}_{w\in\mathsf{sib}(\lfloor i/\mu \rfloor)}$, where $\mathsf{sib}(\lfloor i/\mu \rfloor)$ is the set of sibling nodes along the path from the root to the $\lfloor i/\mu \rfloor$-th leaf in the folding tree. Then, the verifier computes:

$$D^* = \sum_{w\in\mathsf{sib}(\lfloor i/\mu \rfloor)} D_w + D_{\lfloor i/\mu \rfloor}, \quad y^*_{[i]_\mu} = \sum_{w\in\mathsf{sib}(\lfloor i/\mu \rfloor)} y_{w,[i]_\mu} + y_{\lfloor i/\mu \rfloor, [i]_\mu}.$$

By the construction of the folding scheme, we have: $D^* = \prod_{j=0}^{\mu-1} C_j^{r_j}$ and $g^* = \sum_{j=0}^{\mu-1} r_j f_j$, so that D^* is a commitment to the folded polynomial g^*, and $y^*_{[i]_\mu} = g^*(\mathsf{Bin}([i]_\mu))$. As per Step 2.3, the proof $\pi^*_{[i]_\mu}$ proves $y^*_{[i]_\mu} = g^*(\mathsf{Bin}([i]_\mu))$. Then, by Definition 3, the correctness property of the underlying PC scheme implies that $\mathsf{PC.Verify}(D^*, \mathsf{Bin}([i]_\mu), y^*_{[i]_\mu}, \pi^*_{[i]_\mu}) = 1$.

Thus, the verifier accepts the proof with probability 1, confirming the correctness of the interactive protocol. And under the random oracle model, the correctness of the non-interactive protocol resulting from the Fiat-Shamir transformation [74] follows directly from the established correctness of the underlying interactive scheme.

Theorem 3. *FlexProofs satisfies the position binding property (Definition 6) in the ROM.*

Proof. We first prove that the interactive version of FlexProofs satisfies the position binding property. Concretely, we show that for any security parameter λ, any integer $N > 0$, and any PPT adversary $\mathcal{A}_{OpenAll}$,

$$\Pr\left[\begin{array}{c}\mathsf{pp} \leftarrow \mathsf{VC.Setup}(1^\lambda, 1^N) : (\langle \mathcal{A}_{OpenAll}, \mathcal{V}_i\rangle(\mathsf{pp}, C, i, m_i) = 1) \wedge \\ (\langle \mathcal{A}_{OpenAll}, \mathcal{V}_i\rangle(\mathsf{pp}, C, i, m_i') = 1) \wedge (m_i \neq m_i')\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Suppose there is a PPT adversary $\mathcal{A}_{OpenAll}$ that breaks the position binding property of the interactive version with a non-negligible probability ϵ. To distinguish the messages, denote all messages and proofs during the second acceptance (for m_i') with primes.

Let $k = \lfloor i/(\mu b) \rfloor$ and $K = [kb, (k+1)b)$. Then either

$$\exists a \in K : C_a \neq C_a', \text{ or} \tag{17}$$
$$\forall a \in K : C_a = C_a'. \tag{18}$$

For the remaining part of the proof, we discuss two cases. In the first case, $\mathcal{A}_{OpenAll}$ breaks the position binding property of the interactive version with two accepting proof transcripts that satisfies (17). In the second case, $\mathcal{A}$ breaks the position binding property of the interactive version with two accepting proof transcripts that satisfies (18). For $\ell \in \{1, 2\}$, let ϵ_ℓ be the probability that case ℓ occurs. Then $\epsilon = \epsilon_1 + \epsilon_2$. We will show that ϵ is negligible by showing that both ϵ_1 and ϵ_2 are negligible.

ϵ_1 is negligible. If ϵ_1 is non-negligible, we construct an adversary $\mathcal{B}_{FC}$ that breaks the function binding of the FC scheme with probability at least ϵ_1 and thus give a contradiction. Given the pp_{FC} in Eq. (7), $\mathcal{B}_{FC}$ generates pp_{PC} as per Eq. (7), and invokes $\mathcal{A}_{OpenAll}$ with $\mathsf{pp} = \{\mathsf{pp}_{FC}, \mathsf{pp}_{PC}\}$. Upon receiving $\mathcal{A}_{OpenAll}$'s two accepting proof transcripts that satisfies (17), $\mathcal{B}_{FC}$ outputs

$$(C, \{\mathbf{u}_a, C_a, C_a'\}_{a \in K}, \pi_{\mathbf{C}_k}, \pi'_{\mathbf{C}_k}).$$

Note that these values satisfies the following properties:

(1) $\mathsf{FC.BVerify}(C, \{\mathbf{u}_a\}_{a \in K}, \{C_a\}_{a \in K}, \pi_{\mathbf{C}_k}) = 1$; (by the transcript for m_i)
(2) $\mathsf{FC.BVerify}(C, \{\mathbf{u}_a\}_{a \in K}, \{C_a'\}_{a \in K}, \pi'_{\mathbf{C}_k}) = 1$; (by the transcript for m_i')
(3) $\exists a \in K : C_a \neq C_a'$. (by Eq. (17))

Therefore, $\mathcal{B}_{FC}$ breaks the function binding property of the FC scheme with probability $\geq \epsilon_1$, which gives a contradiction.

ϵ_2 is negligible. The soundness of the folding scheme for $\mathsf{PC.HyperEval}$ is guaranteed by [27,10]. If ϵ_2 is non-negligible, we construct an adversary $\mathcal{B}_{FH}$ that breaks the soundness of the folding scheme with probability at least ϵ_2 and thus give a contradiction.

For convenience, let $j = \lfloor i/\mu \rfloor$ and $[i]_\mu = i \bmod \mu$. First, recall that the accepting proof transcript contains the following: (1) a random value r_j, (2) $\{D_w, y_{w,[i]_\mu}\}_{w \in \mathsf{sib}(j)}$, where $\mathsf{sib}(j)$ denotes the set of sibling nodes along the path

from the root to the j-th leaf node in the folding tree and (3) an opening proof $\pi^*_{[i]_\mu}$ of the final folded polynomial.

Given the pp_{PC} in Eq. (7), $\mathcal{B}_{FH}$ generates pp_{FC} as per Eq. (7), and invokes $\mathcal{A}_{OpenAll}$ with $\mathsf{pp} = \{\mathsf{pp}_{FC}, \mathsf{pp}_{PC}\}$. Upon receiving $\mathcal{A}_{OpenAll}$'s two accepting proof transcripts that satisfies (17), $\mathcal{B}_{FH}$ computes

$$D_c = \sum_{w \in \mathsf{sib}(j)} D_w,\ y_{[i]_\mu} = \sum_{w \in \mathsf{sib}(j)} y_{w,[i]_\mu},\ D'_c = \sum_{w \in \mathsf{sib}(j)} D'_w,\ y'_{[i]_\mu} = \sum_{w \in \mathsf{sib}(j)} y'_{w,[i]_\mu}.$$

Then it outputs two folding transcripts

$$\rho_1 = (D_c, C_j, [i]_\mu, y_{[i]_\mu}, m_i, r_j, \pi^*_{[i]_\mu}), \rho_2 = (D'_c, C'_j, [i]_\mu, y'_{[i]_\mu}, m'_i, r'_j, \pi^{*\prime}_{[i]_\mu}).$$

Note that these values satisfies the following properties:

(1) ρ_1 passes the verification of the folding scheme; (by the transcript for m_i)
(2) ρ_2 passes the verification of the folding scheme; (by the transcript for m'_i)
(3) $C_j = C'_j$; (by Eq. (18))
(4) $m_i \neq m'_i$.

Therefore, $\mathcal{B}_{FH}$ breaks the soundness of the folding scheme for $\mathsf{PC.HyperEval}$ with probability $\geq \epsilon_2$, which gives a contradiction.

We have shown that the interactive version of FlexProofs satisfies soundness, captured by the *position binding* property. Next, we show that the interactive version of FlexProofs satisfies a stronger notion of soundness called *state restoration soundness* [6]. To prove state restoration soundness, we consider adversaries that are allowed to reset or rewind the verifier, potentially reusing the verifier's randomness or internal state. We note that the above soundness proof is robust to such behaviors. In particular, the reductions used in both cases (function binding and folding soundness) remain valid even if the verifier's challenge randomness is reused across different transcripts. Hence, the soundness proof already implies that the interactive protocol satisfies state restoration soundness.

The work [74] formally proves that as long as an interactive proof protocol is secure against state restoration attacks, then the non-interactive protocol by applying their Fiat-Shamir transformation is sound in the random oracle model. Therefore, we conclude that our non-interactive scheme satisfies soundness (i.e., position binding) in the random oracle model. □