KZH-Fold: Accountable Voting from Sublinear Accumulation

George Kadianakis[1], Arantxa Zapico[2], Hossein Hafezi[3], and Benedikt Bünz[4]

[1,2]Ethereum Foundation
[3,4]New York University

Abstract

Accumulation schemes are powerful primitives that enable distributed and incremental verifiable computation with less overhead than recursive SNARKs. However, existing schemes with constant-size accumulation verifiers, suffer from linear-sized accumulators and deciders, leading to linear-sized proofs that are unsuitable in distributed settings. Motivated by the need for bandwidth efficient *accountable* voting protocols, (I) We introduce KZH, a novel polynomial commitment scheme, and (II) KZH-fold, the first sublinear accumulation scheme where the verifier only does 3 group scalar multiplications and $O(n^{1/2})$ accumulator size and decider time. Our scheme generalizes to achieve accumulator and decider complexity of $k \cdot n^{1/k}$ with verifier complexity k. Using the BCLMS compiler, (III) we build an IVC/PCD scheme with sublinear proof and decider. (IV) Next, we propose a new approach to non-uniform IVC, where the cost of proving a step is proportional only to the size of the step instruction circuit, and unlike previous approaches, the witness size is not linear in the number of instructions. (V) Leveraging these advancements, we demonstrate the power of KZH-fold by implementing an *accountable voting scheme* using a novel signature aggregation protocol supporting millions of participants, significantly reducing communication overhead and verifier time compared to BLS-based aggregation. We implemented and benchmarked our protocols and KZH-fold achieves a 2000x reduction in communication and a 50x improvement in decider time over Nova when proving 2000 Poseidon hashes, at the cost of 3x the prover time.

{asn, arantxa}@ethereum.org, {h.hafezi, bb}@nyu.edu
The order of author is arbitrary.

Contents

1 Introduction

Distributed verifiable computation (also known as proof-carrying-data or PCD) [CT10] is a powerful primitive that allows a distributed set of parties to jointly perform a computation such that each intermediate step and the result can be efficiently verified. This is useful for computations that are performed by distributed and distrusting parties. Each party can perform a step of the computation and then forward their output, along with a PCD proof that asserts the correctness of the computation up to this point. Examples of these computations are map-reduce systems [CTV15], where nodes jointly compute on large data sets, distributed rollups [Her24; KB23] where nodes compute joined transaction blocks, and distributed voting, where participants jointly compute a quorum certificate on a set of nodes. PCD can be built from recursive SNARKs; however, this approach has high overhead, requiring implementing the SNARK verifier inside of the proof circuit [Ben+14]. A promising recent line of work [KST22; Bün+21; KS24; BC23] showed how to construct PCD from so-called accumulation schemes. An accumulation scheme iteratively accumulates proofs into an accumulator. The accumulator is valid if and only if the input proofs are all valid. Importantly, checking the validity of the accumulation is much simpler than checking all the original proofs. PCD built from the most efficient accumulation schemes [KST22; Bün+21; KS24; BC23] only requires proving a constant number of group operations as overhead.

Unfortunately, these accumulation schemes have significant limitations. The accumulator, which must be communicated among distributed provers, is linear in the size of a computation step and takes linear time to decide. Each computation step is lower-bounded by the so-called recursive overhead, i.e., the cost of verifying the accumulation step. Even for the most efficient schemes, this is in the tens of thousands of gates, which results in accumulators that are at least 1 MB in size, and the decider scales linearly in the underlying computation. While this overhead may be acceptable if a single prover performs the computation, this is a significant bottleneck to applying PCD to distributed and peer-to-peer settings. In these settings, every node needs to receive and fully decide, i.e., check the accumulator when it wants to take over a computation.

In this work, we present KZH, a polynomial commitment scheme. KZH, like the famous KZG polynomial commitment [KZG10], is secure in a pairing-friendly group, leverages a universal, upgradable structured reference string, and is secure in the algebraic group model. We also design KZH-fold, an efficient accumulation scheme for KZH. Unlike prior accumulation schemes, KZH-fold has a sublinear-sized accumulator that can be efficiently verified, while retaining the constant size and concretely efficient accumulation verifier of prior works, like Nova or Protostar. Concretely, we present an accumulation scheme where the accumulator for computations of size n is $O(n^{\frac{1}{2}})$, compared to $O(n)$ for Nova, and the accumulation verifier performs 3 group scalar multiplications to add a proof to the accumulator (compared to 2 for Nova, 3 for Protostar and 1 for HyperNova). The decider runtime is dominated by a pairing product of size $n^{\frac{1}{2}}$. We also generalize the scheme to KZH-k such that the accumulator and decider are only $O(k \cdot n^{\frac{1}{k}})$, with an $O(k)$

accumulation verifier for an arbitrary constant k. Using the compilers from [Bün+20; Bün+21; BC23], we construct an accumulation scheme for KZH which, if combined with a variant of Spartan [Set20], yields an accumulation scheme for R1CS. We then show that our scheme has significant benefits in an important application: accountable voting for large-scale consensus.

1.1 KZH polynomial commitment scheme

Extending ideas from Hyrax, we design a similar multilinear polynomial commitment scheme named KZH with following appealing features: Given a polynomial of degree n, KZH has a linear-time commitment phase, primarily dominated by an MSM of size n. The commitment is a single $\mathbb{G}_1$ element. The opening and verification times are both $O(n^{\frac{1}{2}})$, and the proof size is also $O(n^{\frac{1}{2}})$. The core idea of KZH is to represent polynomial evaluation as a matrix operation, reducing it to matrix-vector multiplication. We extend this concept to tensors, where a matrix is a tensor of dimension 2, and introduce KZH-k. KZH-k has linear commitment time, and the opening time remains $O(n^{\frac{1}{2}})$ through preprocessing, but the verifier time and proof size are both changed to $O(k \cdot n^{\frac{1}{k}})$.

KZH-log(n) is of independent interest as a standalone commitment scheme. Like the multivariate commitment scheme from [PST13], it has $O(\log(n))$ group elements proof size, and verifier time. Its key advantage is that computing an opening proof can be done using only $n^{\frac{1}{2}}$ group operations (not including polynomial evaluation). KZH-log(n) unlike similar schemes like Dory [Lee21], the proof does not consist of target group elements and subsequently the verifier does not do target group operations which limits its application to smart contracts not supporting such operations. In Table 1 we compare various polynomial commitment schemes with KZH-log(n).

Scheme	Supports	Opening time	Proof size	Verifier time
KZH-log(n)	Multilinear	$n^{\frac{1}{2}}\,\mathbb{G}_1$	$2\log(n)\,\mathbb{G}_1$	$2\log(n)$ P
[PST13]	Multivariate	MSM(n)	$\log(n)\,\mathbb{G}_1$	$\log(n)$ P
[KZG10]	Univariate	MSM(n)	$1\mathbb{G}_1$	2 P
[Lee21]	Multilinear	$n^{\frac{1}{2}}$ P	$3\log(n)\,\mathbb{G}_T$	$\log(n)\,\mathbb{G}_T$, P

Table 1: Comparison for KZH-log(n), KZG, PST and Dory. For the multivariate scheme log(n) is the number of variables, for KZG n is the degree. P stands for pairing operations $\mathbb{G}_1$ and $\mathbb{G}_T$ for base and target group operations respectively. Commitment time for all schemes is dominated by an MSM of size n.

1.2 Sublinear accumulation schemes

As our core technical contribution, we design the first accumulation scheme[a] with a sublinear accumulator and decider (including the accumulator witness) and a constant-sized accumulation verifier. Concretely, the accumulation verifier performs only 3 group scalar multiplications and has an accumulator consisting of $O(n^{\frac{1}{2}})$ elements. We generalize the scheme to KZH-k, a scheme with an accumulator of size $k \cdot n^{\frac{1}{k}}$ and an accumulation verifier of size $O(k)$. Through the compilers of [Bün+21; KS23a], we get a PCD scheme with minimal overhead and significantly smaller and faster to verify PCD proofs compared to any prior accumulation-based PCD.

Our new accumulation scheme KZH-fold is based on our polynomial commitment scheme KZH. We construct KZH-fold, a highly efficient accumulation scheme for KZH in which the accumulation verifier only performs 3 $\mathbb{G}_1$ operations. The accumulator witness consists of $n^{\frac{1}{2}}$ group elements and $5 \times n^{\frac{1}{2}}$ field elements and is thus significantly smaller than the original polynomial, and subsequently the decider runs in time $O(n^{\frac{1}{2}})$, dominated by a pairing-product of that size. We combine KZH-fold with a *polynomial interactive oracle proof* (PIOP) [BFS20; Set20], to get an accumulation scheme for R1CS (an encoding of arithmetic circuits). This R1CS accumulation scheme yields a PCD scheme through the BCLMS [Bün+21] and Cyclefold [KS22a] compilers. The accumulator size directly corresponds to the PCD proof size and the decider to the PCD verifier time. KZH-fold-based PCD is, therefore, the first accumulation-based PCD scheme with constant recursive overhead and a sublinear proof size and verification time.

The sublinear proof is advantageous in distributed applications of PCD, such as prover networks or accountable voting, as proofs must be passed from node to node. For this reason, prior applications of accumulation-based PCD were mainly limited to single-prover scenarios. Nova [KST22], previously proposed compressing the proof via outsourcing the decider. However, computing this proof has a 24x prover overhead and a 2x decider overhead, compared to the accumulation step without compression. Even in the compressed setting, the decider remains linear and inefficient. Using KZH-fold, this additional compression step is unnecessary, as the decider is already efficient. Concurrent work, MicroNova [ZSC24], modifies Nova's compression phase to achieve sublinear decider time. However, this improvement comes at an even higher prover cost than Nova's compression phase. Note that the compressed proofs in Nova and MicroNova are no longer incrementally updateable with an accumulation scheme. While this may be acceptable in a single-prover setting for applications like rollups, it is problematic in distributed settings or applications with an ongoing computation such as building light clients from IVC [Che+20], constant-sized blockchains [Bon+20] or verifiable virtual machines (e.g. zkVM).

KZH-fold retains many of the benefits of previous accumulation schemes, shown in Table 2. It only requires a single commitment to the witness and can take advantage of

[a] Technically, any SNARK yields an accumulation scheme. However, no SNARK exists where the verification only requires basic group operations.

sparse and small-weight witnesses. The accumulation verifier only performs a small (3) number of group scalar multiplications in group one of a pairing-friendly group. Finally, we generalize KZH-fold to variants called KZH-k fold. For any constant $k \in \{2, \log_2(n)\}$, KZH-k fold yields an $O(k \cdot n^{\frac{1}{k}})$ accumulator and decider with an $O(k)$ accumulation verifier. Unlike prior accumulation constructions, KZH-fold requires a trusted setup, but that setup is universal and updatable. It is possible to reuse a powers-of-τ [Gro+18] setup, commonly used for the KZG polynomial commitment. We implement KZH2-fold and KZH3-fold, and show that even our unoptimized implementation has a 200-2000x times smaller accumulator than Nova for reasonable computations.

Scheme	**Recursive Overhead**	**Decider**	**|acc|**
Nova	2 group ops	MSM(n)	$O(n)$
KZH2-fold	3 group ops	$n^{\frac{1}{2}}$ P	$O(n^{\frac{1}{2}})$
KZH-k fold	$k+1$ group ops	$k \cdot n^{\frac{1}{k}}$ P	$O(k \cdot n^{\frac{1}{k}})$
Halo	$O(\log n)$ group ops	MSM(n)	$O(\log n)$

Table 2: Comparison for Nova, KZH-fold, and Halo, MSM(n) is a multi-scalar multiplication of size n, and P is a pairing operation.

1.3 Signature aggregation in consensus

Voting protocols, especially accountable open-ballot systems, require robust yet efficient methods to aggregate and verify votes from a large participant base. These protocols depend on mechanisms that ensure each participant's vote is accurately recorded and that the overall results remain transparent and verifiable. A prominent application of accountable voting principles can be found in blockchain consensus protocols. In decentralized networks like Ethereum, over 1 million validators must attest to each block by signing it, collectively establishing consensus on network state changes[b]. With such a high volume of participants, bandwidth efficiency becomes critical, not only due to the redundancy and overhead inherent in decentralized P2P networking but also to support the participation of low-end machines as validators.

In consensus protocols, a block is considered *finalized* when a supermajority of validators vote for it, ensuring it cannot be reverted. Currently, Ethereum limits itself to aggregating 32,768 signatures per slot [Resb], which delays finality to 15 minutes. Reducing the cost of signature aggregation could simultaneously improve finality time to just a few seconds, which refers to the speed at which transactions become irreversible [Sin], while also enhancing Ethereum's decentralization by enabling a higher validator count [But]. Furthermore, efficient aggregation could facilitate the adoption of advanced consensus protocols, designed

[b] https://beaconcha.in/charts/validators

explicitly for Ethereum [D'A+24b; DZ23; D'A+24a], which aim to provide faster finality, strengthen protocol security [D'A+22], and improve resilience to MEV.

Current state-of-the-art consensus protocols, such as Ethereum, Chia, Algorand [Gil+17] and Hotstuff [Yin+19] employ aggregatable signature schemes like BLS signatures [BLS04]. BLS signatures allow multiple validators to merge their signatures on the same message, e.g. a block, into a single, compact, aggregated message, reducing both transmission and verification overhead. To achieve accountability, Blockchain protocols [BDN18] pair BLS aggregates with a bit vector indicating who has signed. This enables slashing validators who sign conflicting proposals. To add redundancy and improve communication, consensus protocols can use multiple layers of *recursive aggregation*, aggregating already aggregated signatures into one. In proposed designs [Resa], the signatures are aggregated in multiple layers, and each aggregation layer has between $r = 16$ and $r = 32$ aggregators. Figure 1 depicts such a tree-based aggregation mechanism.

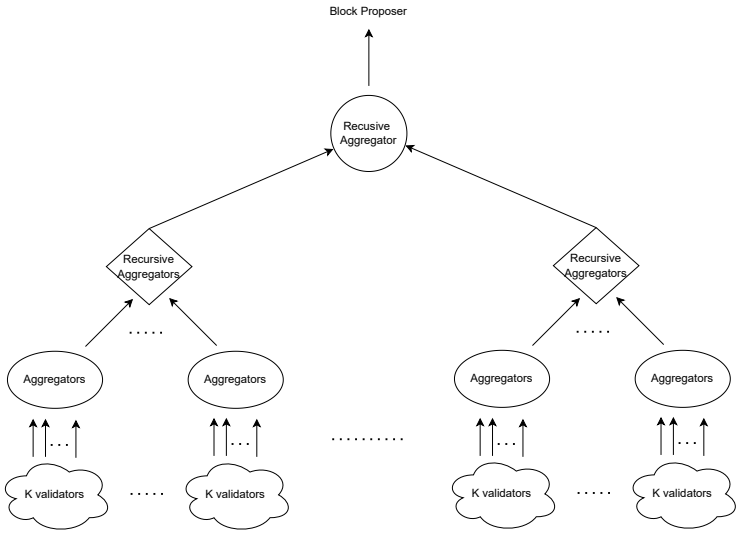

Figure 1: A tree-based aggregation scheme with a single layer of recursive aggregation.

Limitations of signature aggregation. In this work, we are interested in accountable signature aggregation schemes with *minimal communication and verification overhead*. Regarding communication, an accountable aggregation scheme such as the one depicted in Figure 1, requires nodes to transmit, at minimum, a compact bit vector (i.e. a bitfield) indicating who signed. For verification, nodes must scan the bitfield to identify the signers while ensuring that verification complexity remains independent of the number of signers. BLS aggregation, while yielding a small aggregate signature, does not achieve this optimal

design in either dimension: Each *recursive aggregation* layer incurs an $O(\log r)$ communication overhead, where r is the total number of aggregators. This overhead arises because aggregating bitfields with overlapping signers requires each aggregator to track multiplicities, i.e. the number of times each signature appears across overlapping bitfields. If r aggregators combine their bitfields then $\log(r)$ bits are necessary to indicate the multiplicity for each signer. BLS aggregate verification is dominated by the computation of the aggregate public key. This is done through a multi-scalar multiplication (MSM) between the multiplicities and the list of public keys. For n signers the MSM has length n and $\log(r)$-bit scalars. For $n = 2$ million and $r = 16$ the verification time of a BLS aggregate signature, is 0.7 seconds (See Table 4). Every validator incurs this cost before checking the correctness of a consensus proposal and moving on to the next block.

Additionally, transmitting these multiplicity lists introduces a $k \cdot \log r$ multiplicative factor communication costs, where k is the number of aggregation layers. Even with a single layer of recursion and $r = 16$, the multiplicities have a 1 MB representation. For 4 million validators, 4 levels of recursion and $r = 32$, the multiplicities require 10MB to represent. Today, the average Ethereum block is less than 1MB, and is published every 12 seconds. This demonstrates how even the constant factor overhead of aggregate BLS can become a significant bottleneck in consensus.

Signature aggregation using PCD. Our work examines the use of distributed verifiable computing for signature aggregation, leveraging their expressiveness to union bitfields directly within the proof. Unlike BLS, which requires multiplicities, SNARKs enable a compact bitfield to represent validator participation, reducing communication overhead. Effectively, the PCD proves that the signature was correctly aggregated *and* that the bitfields were correctly unioned. This approach offers a more bandwidth-efficient solution for accountable voting in P2P protocols with a large number of participants.

1.4 Contributions

We make theoretical and systems contributions that advance the state of the art for distributed proving:

KZH polynomial commitment scheme. In Section 4 we introduce KZH, a polynomial commitment scheme that has sublinear verification, sublinear proof size, and efficient opening proof generation and the commitment consists of a single group element.

KZH-fold with sublinear accumulator/decider. Building on KZH, we design KZH-fold in Section 4.1, the first accumulation scheme with a sublinear accumulator and decider. It sits between Nova and Halo in the trade-off space as can be seen in Table 2. Using KZH-fold, we built the first IVC/PCD scheme with a sublinear accumulator and decider, as

described in Section 4.2. In Appendix C we generalize KZH and KZH-fold to polynomial commitments of dimension k.

New approach to non-uniform IVC. In Section 4.3 we propose a new non-uniform IVC approach using a PCS accumulation scheme with a sublinear verifier such that, the per-step cost matches the step circuit cost. However, unlike prior methods, the witness size of the step function does not grow linearly with the number of instructions. For a zkVM with 64 instructions, this achieves a 64x reduction in witness size and communication.

Signature aggregation protocol. We design and optimize an accountable signature aggregation protocol in Section 5, using KZH-fold. The protocol supports unlimited aggregations with optimal communication and verification time. The communication is dominated by a bitvector indicating the signers. The verification is dominated by just one field multiplication per signer.

Implementation and evaluation. We implement[c] and evaluate KZH-2-fold and KZH-3-fold and compare it against Nova in Section 6. We show that KZH-3-fold, at the expense of only 3x prover cost, achieves a 2000x reduction in communication overhead and a 50x slimmer verifier time. We also implement our accountable voting protocol using KZH-3-fold, demonstrating its effectiveness and practical applicability. For four million signatures, our scheme reduces the communication cost by more than 10x and the verifier time by 4x compared to an accountable voting protocol using BLS signatures.

1.5 Additional related work

Accumulation. The Halo protocol [BGH19] was the first accumulation protocol; it has a succinct accumulator of $O(\log n)$ size and an accumulation verifier with $O(\log n)$ group operations, but the decider runs in time $O(n)$. In contrast, KZH-fold, only has a constant size accumulation verifier, as well as an accumulator and decider of size $O(k \cdot n^{\frac{1}{k}})$.

Distributed proving. Most prior works on distributed zkSNARKs [Wu+18; Liu+23; Ros+24; Wan+24] rely on a coordinator that receives the circuit C, the public input x, and the witness w, then distributes these to worker nodes and aggregates their outputs into a single proof. This centralized approach introduces a significant vulnerability: the coordinator represents a single point of failure and carries substantial responsibility, making it unsuitable for fully decentralized systems. Another limitation is that aggregated proofs cannot be further aggregated. An alternative approach involves using Proof-Carrying Data (PCD), where each node performs a portion of the computation, and these partial proofs are aggregated (accumulated) hierarchically in a tree-like structure. However, decentralized

[c] https://github.com/asn-d6/kzh_fold

PCD requires each aggregator to verify the validity of incoming partial proofs using a decider, which cannot trust other nodes by default. In previous schemes, this verification step was a bottleneck because the decider's complexity is linear with respect to the original statement. Additionally, the witness size also grows linearly with the statement, leading to substantial peer-to-peer (P2P) communication overhead. KZH-fold addresses these issues by reducing both communication complexity and witness size to $O(k \cdot n^{\frac{1}{k}})$, significantly enhancing the efficiency and scalability of decentralized proving systems.

Signature aggregation. Several works have studied signature aggregation for consensus. Handel shows how to build aggregation structures, in the face of adversarial corruptions [Bé+19]. Their work is largely orthogonal and should be compatible with arbitrary aggregate signature schemes. [Kha+21; Aar+24] provide aggregation schemes for hash and lattice-based signature schemes, respectively. However, their constructions only support one level of aggregation, whereas our construction is focused on aggregating already aggregated signatures.

2 Technical Overview

At the core of our construction is our new KZH polynomial commitment scheme, combining ideas from the Hyrax polynomial commitment [Wah+18] and KZG [KZG10]. Similar to Hyrax, we are building a commitment for a matrix $M \in \mathbb{F}^{m \times n}$, such that we can open a bilinear vector matrix-vector product, for vectors $\vec{a} \in \mathbb{F}^m$ and $\vec{b} \in \mathbb{F}^n$, i.e. prove that $y = \vec{a}^T \times M \times \vec{b}$. This is general enough to construct both univariate and multilinear polynomial commitments, i.e. $\vec{a}$ and $\vec{b}$ are extensions of the evaluation point.

In Hyrax, each row of M is committed using a Pedersen commitment [Ped92]. The resulting commitment vector, $\vec{\mathsf{D}} = [\mathsf{D}_1, \ldots, \mathsf{D}_m] \in \mathbb{G}^m$, consists of m group elements, with each element corresponding to a row of M. Due to the homomorphic properties of the commitment scheme, the verifier can validate an opening efficiently. To verify, the verifier first computes $\mathsf{C} = \langle \vec{a}, \vec{\mathsf{D}} \rangle$. The prover then opens C to a vector $\vec{r} \in \mathbb{F}^n$ and claims $y = \langle \vec{r}, \vec{b} \rangle$. The verifier checks two conditions: $\mathsf{C} = \mathsf{Commit}(\vec{r})$, and $y = \langle \vec{r}, \vec{b} \rangle$. These checks involve two inner products, one of size n and the other of size m. For a matrix with ℓ entries, setting $n = m = \ell^{\frac{1}{2}}$ yields a verification time of $O(\ell^{\frac{1}{2}})$.

Moreover, efficient accumulation schemes for inner products are known [Bün+21; BC23], suggesting that it may be feasible to construct an accumulation scheme for Hyrax. However, Hyrax has a significant limitation: its commitment consists of m group elements, and while the commitment is homomorphic, performing homomorphic operations requires m group additions. The primary method for constructing accumulation schemes relies on homomorphically combining commitments. In the case of Hyrax, this approach leads to an accumulation verifier with size $O(m) = O(\ell^{\frac{1}{2}})$, which is notably larger than the constant-size accumulation verifiers achievable with other group-based constructions.

Our key insight is that we can modify Hyrax, such that the commitment only consists of a single group element, and the homomorphism can be performed efficiently, using only a single group operation. A strawman approach here, would be to commit to $\vec{\mathsf{D}} \in \mathbb{G}^m$ using a *structure-preserving* commitment to group elements [Abe+16]. While these commitments, preserve the homomorphism, the commitment to a vector of group elements will be a target group element in a pairing-based group. Target group operations are significantly more expensive, especially when implemented as an arithmetic circuit. A single target group scalar multiplication takes tens of thousands of R1CS constraints.

We aim to design a commitment scheme where the homomorphism only requires a single $\mathbb{G}_1$ operation, in a pairing-friendly group. To do this, we utilize a common reference string, similar to KZG. Let $\vec{\mathsf{G}} = (\mathsf{G}_1, \ldots, \mathsf{G}_n)$ be the generators for the Pedersen commitment. We now construct $\vec{\mathsf{H}}^{(i)} = \tau^{(i)} \times \vec{\mathsf{G}}$, for each $i \in [m]$ and a secret trapdoor $\tau^{(i)}$. We first commit to the matrix M by computing the commitment $\mathsf{C} = \sum_{i\in[m],j\in[n]} M_{i,j} \times \mathsf{H}_j^{(i)}$, where $\mathsf{C} \in \mathbb{G}_1$ is a single group element. Next, we compute commitments to each individual row of the matrix using the vector $\vec{\mathsf{G}}$, where each row commitment is given by $\mathsf{D}_i = \sum_j M_{i,j} \times \mathsf{G}_j$. During the opening phase, the prover sends $[\mathsf{D}_i]_{i=1}^m$, and the verifier ensures consistency between the D_is commitments and C using a pairing-based check. Specifically, a generator $\mathsf{V} \in \mathbb{G}_2$ is sampled, and the verifier computes $\mathsf{V}_i = \tau^{(i)} \times \mathsf{V}$ for all $i \in [m]$. The correctness of the decomposition is verified by checking the equality $e(\mathsf{C}, \mathsf{V}) = \sum_{i=1}^m e(\mathsf{D}_i, \mathsf{V}_i)$. Here, $[\mathsf{D}_i]_{i=1}^m$ corresponds exactly to the Hyrax commitment, allowing us to reuse Hyrax's opening algorithm with the added decomposition check. Furthermore, C is a homomorphic commitment to both the D_i values and the matrix M, represented compactly as a single element in $\mathbb{G}_1$.

KZH-fold. Next, we leverage the Protostar [BC23] compiler to design an efficient accumulation scheme for KZH, referred to as KZH-fold. The verifier's structure in KZH enables a highly efficient scheme. Specifically, the Hyrax checks, namely multi-scalar multiplications (MSMs), can be efficiently accumulated using existing techniques. Additionally, the KZH verifier validates a pairing product: $e(\mathsf{C}, \mathsf{V}) = \sum_{i=1}^m e(\mathsf{D}_i, \mathsf{V}_i)$. Notably, one side of all pairings remains fixed. When combining two equations, we derive: $e(\mathsf{C} + X \times \mathsf{C}', \mathsf{V}) = \sum_{i=1}^m e(\mathsf{D}_i + X \times \mathsf{D}'_i, \mathsf{V}_i)$, with an overwhelming probability for a random $X \in \mathbb{F}$, if and only if the pairing check holds for both $(\mathsf{C}, [\mathsf{D}_i]_{i=1}^m)$ and $(\mathsf{C}', [\mathsf{D}'_i]_{i=1}^m)$. The linearity of this check ensures no additional error terms (which would belong to the target group $\mathbb{G}_T$) are introduced. The final accumulation verifier thus performs only 3 $\mathbb{G}_1$ operations when combining an accumulator with a fresh proof, or 4 operations when merging two accumulators. The accumulator size matches that of a PCS proof, with a size of $O(\ell^{\frac{1}{2}})$. The decider, equivalent to the PCS verifier, is dominated by $O(\ell^{\frac{1}{2}})$ pairings. In order to go from accumulation for a polynomial commitment to accumulation for all of NP, we leverage the Spartan PIOP for R1CS into the scheme. In the accumulation setting, evaluations of multilinear extensions of the R1CS matrices can be efficiently deferred, requiring only a

logarithmic number of additional field operations for the accumulation verifier.

KZH-k. We generalize the scheme to further improve the proof size and verifier efficiency, which yields a smaller accumulator and more efficient decider. The key insight is that instead of committing to a two-dimensional matrix we can commit to a k-dimensional tensor. The matrix vector-matrix product turns into a k-dimensional tensor product, between the tensor and k matrices of size $n^{\frac{1}{k}}$. We first commit to the entire tensor using a structured reference string. Then we use the same technique to open commitments to all $k-1$ dimensional slices of the tensor. If the full tensor has n entries, then there are $n^{\frac{1}{k}}$ such slices. We use a pairing check and the reference string to check the consistency between the slices and the full commitment. Then we can evaluate the slices homomorphically with the first of k vectors. This yields a single $k-1$ dimensional commitment. At this point, we proceed recursively, until we reach a one-dimensional vector that can be opened in $O(n^{\frac{1}{k}})$. Since each dimension consists of $n^{\frac{1}{k}}$ slices, and there are k slices, the overall proof size and verification time is $O(k \cdot n^{\frac{1}{k}})$. The resulting accumulation verifier increases slightly to $k+1$ $\mathbb{G}_1$ operations.

Signature aggregation. We use a KZH-based PCD scheme in order to construct an aggregate accountable signature scheme. The IVC aggregates the public key signatures and signer bitfields of a BLS accountable aggregate signature. The key challenge is ensuring accountability by proving that the output bitfield is equivalent to the or of the input bitfields. Let $\vec{b}^{(1)}, \vec{b}^{(2)}$ and $\vec{c}$ be three bitfields such that $\vec{c} = \vec{b}^{(1)} \vee \vec{b}^{(2)}$. Naively putting this statement into the circuit would make the IVC circuit linear in the number of signers and possibly increase the verification cost. Instead at each aggregation step, we define multilinear extensions of these vectors $\tilde{b}_1(\vec{X}), \tilde{b}_2(\vec{X})$, and $c(\vec{X})$ and commit to them using a multilinear polynomial commitment. In order to prove that $\vec{c} = \vec{b}^{(1)} \vee \vec{b}^{(2)}$, we can show that $\tilde{b}_1(\vec{x}) + \tilde{b}_2(\vec{x}) - \tilde{b}_1(\vec{x}) \cdot \tilde{b}_2(x) = \tilde{c}(\vec{x}) \, \forall \vec{x} \in \{0,1\}^{\mu}$. This is a zerocheck and can be proven using a simple sumcheck protocol as in HyperPlonk [Che+23]. The sumcheck requires evaluating the polynomial commitments to $\tilde{b}_1, \tilde{b}_2, \tilde{c}$ at a random point. We can instantiate the PCS with KZH and accumulate the opening proofs as part of the IVC. We detail the scheme and further optimizations in Section 5.

3 Preliminaries

3.1 Notation

Let $\mathbb{F}$ be a finite field and $\mathbb{G}$ a group with scalars in $\mathbb{F}$, with additive notation. For $a \in \mathbb{F}$ and $\mathsf{G} \in \mathbb{G}$, the scalar multiplication of G by a is denoted as $a \times \mathsf{G}$. For an asymmetric pairing group, we define it as a tuple $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where p is the order of groups $\mathbb{G}_1$ and $\mathbb{G}_2$, and e is an efficiently computable, non-degenerate bilinear map. Let GGen be

a deterministic polynomial-time algorithm that takes as input a security parameter λ and outputs a such a group description. A function $f(x)$ is negligible if, for any polynomial $p(x)$, there exists a positive integer N such that for all $x > N$, we have $f(x) < \frac{1}{p(x)}$. When we say an event happens with overwhelming probability, we mean it occurs with a probability of $1 - \epsilon(\lambda)$, where $\epsilon(\lambda)$ is a negligible function. We use $\langle \vec{a}, \vec{b} \rangle$ to denote the inner product between two vectors $\vec{a}, \vec{b} \in \mathbb{F}^n$, and extend this notation for $\vec{a} \in \mathbb{F}^n$ and $\vec{\mathsf{G}} \in \mathbb{G}^n$ as $\langle \vec{a}, \vec{\mathsf{G}} \rangle = \sum_{i=1}^{n} a_i \times \mathsf{G}_i$. Additionally, we use $\mathsf{MLP}(\mathbb{F}, d)$ to denote the set of multilinear polynomials with d variables over the field $\mathbb{F}$, which we abbreviate as $\mathsf{MLP}(d)$ when $\mathbb{F}$ is understood from the context. Operator $\|$ represents concatenation, e.g. $\vec{x} \| \vec{y}$ is the concatenation of vectors $\vec{x}$ and $\vec{y}$. To indicate equality on vectors, we define $\mathsf{eq}(\vec{X}, \vec{Y})$ as $\mathsf{eq}(\vec{X}, \vec{Y}) = \prod_{i=1}^{k} \big((1 - X_i) \cdot (1 - Y_i) + X_i \cdot Y_i\big)$, so that for $\vec{x}, \vec{y} \in \{0,1\}^k$, $\mathsf{eq}(\vec{x}, \vec{y}) = 1$ if and only if $\vec{x} = \vec{y}$. We denote a complete binary tree of depth n with node values in $\mathbb{F}$ as $\mathcal{T}(\mathbb{F}, n)$. For brevity, we will refer to complete binary trees simply as trees. Given a vector $\vec{x} \in \mathbb{F}^k$, the equality tree $\mathsf{EqTree}(\vec{x})$ is a tree of depth k. The root node is initialized with 1. At depth i, the left child of a node v is $v \cdot (1 - x_i)$, and the right child is $v \cdot x_i$. The leaves of the tree correspond to the equality function $\mathsf{eq}(\vec{x}, \vec{y})$ for all $\vec{y} \in \{0,1\}^k$. For any tree $\mathcal{T}$, we denote the set of its leaf nodes by $\mathcal{T}$.leaves.

Cryptographic Assumptions. We prove security of our protocols in the Algebraic Group Model (AGM) of Fuchsbauer et al. [FKL18], using the *bilinear* version of the q-dlog assumption and quadratic CDH. In the AGM, adversaries are restricted to be *algebraic* algorithms, namely, whenever $\mathcal{A}$ outputs a group element $[y]$ in a cyclic group $\mathbb{G}$ of order p, it also outputs its representation as a linear combination of all previously received group elements. In other words, if $[y] \leftarrow \mathcal{A}([x_1], \ldots, [x_m])$, $\mathcal{A}$ must also provide $\vec{z}$ such that $[y] = \sum_{j=1}^{m} z_j [x_j]$. This definition generalizes naturally in asymmetric bilinear groups with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where for $i \in 1, 2$, the adversary must construct $\mathbb{G}_i$ elements as linear combinations of received $\mathbb{G}_i$ elements.

We denote a random oracle using H that we assume is randomly sampled from the random oracle space $\mathcal{H}$ and initialize it in practice using a secure hash function.

3.2 Polynomial commitment schemes

Definition 1. *[Multilinear Polynomial Commitment Scheme] A* Multilinear Polynomial Commitment Scheme *is a tuple of algorithms* $(\mathsf{Setup}_{\mathsf{PC}}, \mathsf{Commit}_{\mathsf{PC}}, \mathsf{Open}_{\mathsf{PC}}, \mathsf{Verify}_{\mathsf{PC}})$ *such that:*

- $\mathsf{srs}_{\mathsf{PC}} \leftarrow \mathsf{Setup}_{\mathsf{PC}}(\mathsf{pp}_{\mathsf{PC}}, k)$*: On input the system parameters and number of variables* k*, it outputs a structured reference string.*
- $\mathsf{C} \leftarrow \mathsf{Commit}_{\mathsf{PC}}(\mathsf{srs}_{\mathsf{PC}}, p(\vec{X}))$*: On input* $\mathsf{srs}_{\mathsf{PC}}$ *and a polynomial* $p(\vec{X}) \in \mathit{MLP}(\mathbb{F}, k)$*, it outputs a commitment* C*.*

- $\pi_{\mathsf{PC}} \leftarrow \mathsf{Open}_{\mathsf{PC}}\big(\mathsf{srs}_{\mathsf{PC}}, p(\vec{X}), \vec{x}\big)$*: On input* $\mathsf{srs}_{\mathsf{PC}}$*,* $p(\vec{X})$*,* k*, and vector* $\vec{x} \in \mathbb{F}^k$*, outputs an evaluation proof* π_{PC} *that* $y = p(\vec{x})$*.*
- $1/0 \leftarrow \mathsf{Verify}_{\mathsf{PC}}\big(\mathsf{srs}_{\mathsf{PC}}, \mathsf{C}, \vec{x}, \pi_{\mathsf{PC}}, y\big)$*: On input* $\mathsf{srs}_{\mathsf{PC}}$*, the commitment* C*,* k*, vector of evaluations* $\vec{x}$*,* $y \in \mathbb{F}$*, and the proof of the correct evaluation, it outputs a bit indicating acceptance or rejection.*

and satisfies the following properties:

Completeness. *It captures the fact that an honest prover will always convince the verifier. Formally, for any efficient adversary* $\mathcal{A}$*, we have:*

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}_{\mathsf{PC}}\big(\mathsf{srs}_{\mathsf{PC}}, \mathsf{C}, \vec{x}, \pi_{\mathsf{PC}}, y\big) = 1 & \begin{array}{l} \mathsf{srs}_{\mathsf{PC}} \leftarrow \mathsf{Setup}_{\mathsf{PC}}\big(\mathsf{pp}_{\mathsf{PC}}, k\big) \\ p(\vec{X}) \leftarrow \mathcal{A}(\mathsf{srs}_{\mathsf{PC}}) \\ \mathsf{C} \leftarrow \mathsf{Commit}\big(\mathsf{srs}_{\mathsf{PC}}, p(\vec{X})\big) \\ \pi_{\mathsf{PC}} \leftarrow \mathsf{Open}\big(\mathsf{srs}_{\mathsf{PC}}, p(\vec{X}), \vec{x}\big) \end{array} \end{array}\right] = 1$$

Extractability: *Captures the fact that whenever the prover provides a valid opening, it knows a valid pair* $(p(\vec{X}), y) \in \mathbb{F}[\vec{X}] \times \mathbb{F}$*, where* $p(\vec{x}) = y$*. Formally, for all PPT adversaries* $\mathcal{A}$ *there exists an efficient extractor* $\mathcal{E}$ *such that the probability of the following event is negligible:*

$$\Pr\left[\begin{array}{l|l} \begin{array}{c} \mathsf{Verify}_{\mathsf{PC}}\big(\mathsf{srs}_{\mathsf{PC}}, \mathsf{C}, \vec{x}, y, \pi_{\mathsf{PC}}\big) = 1 \\ \wedge\, p(\vec{x}) \neq y \end{array} & \begin{array}{l} \mathsf{srs}_{\mathsf{PC}} \leftarrow \mathsf{Setup}_{\mathsf{PC}}\big(\mathsf{pp}_{\mathsf{PC}}, k\big) \\ \mathsf{C} \leftarrow \mathcal{A}\big(\mathsf{srs}_{\mathsf{PC}}\big) \\ p(\vec{X}) \leftarrow \mathcal{E}\big(\mathsf{srs}_{\mathsf{PC}}, \mathsf{C}, k\big) \\ \vec{x} \leftarrow \mathcal{A}\big(\mathsf{srs}_{\mathsf{PC}}, \mathsf{C}\big) \\ (y, \pi_{\mathsf{PC}}) \leftarrow \mathcal{A}\big(\mathsf{srs}_{\mathsf{PC}}, p(\vec{X}), \vec{x}\big) \end{array} \end{array}\right]$$

3.3 Accumulators

Definition 2. *[Accumulation Scheme] An accumulation scheme [Bün+21; BC23] for a predicate* $\phi : X \to \{0,1\}$ *is a tuple of algorithms* $\Pi_{\mathsf{acc}} = (\mathsf{Setup}_{\mathsf{acc}}, \mathsf{P}_{\mathsf{acc}}, \mathsf{V}_{\mathsf{acc}}, \mathsf{D}_{\mathsf{acc}})$*, all of which have access to the same random oracle* $\mathcal{O}_{\mathsf{acc}}$*, such that:*

$\mathsf{Setup}_{\mathsf{acc}}(1^\lambda) \to \mathsf{srs}_{\mathsf{acc}}$ *: On input the security parameter, outputs public parameters* $\mathsf{srs}_{\mathsf{acc}}$*. For simplicity, we assume that all functions implicitly take* $\mathsf{srs}_{\mathsf{acc}}$ *as input.*

$\mathsf{P}_{\mathsf{acc}}\big(\mathsf{st}, \pi, \mathsf{acc}_1\big) \to \big(\mathsf{acc}, \mathsf{pf}\big)$**:** *The accumulation prover implicitly given* $\mathsf{srs}_{\mathsf{acc}}$*, statement* st*, predicate inputs* $\pi = (\pi.x, \pi.w)$*, and an accumulator* $\mathsf{acc}_1 = (\mathsf{acc}_1.x, \mathsf{acc}_1.w)$*, outputs a new accumulator* acc *and corrections terms* pf*.*

$\mathsf{V}_{\mathsf{acc}}\big(\mathsf{acc}_1.x, \mathsf{acc}_2.x, \mathsf{pf}\big) \to \mathsf{acc}.x$**:** *The accumulation verifier implicitly given* $\mathsf{srs}_{\mathsf{acc}}$*, and on input the instances of two accumulators, and the accumulation proof outputs a new accumulator instance* $\mathsf{acc}.x$*.*

$\mathsf{D}_{\mathsf{acc}}(\mathsf{acc}) \to 1/0$**:** *The decider takes as input* acc *and accepts or rejects.*

and satisfies completeness and soundness as defined below:

Completeness. *For all fresh proofs* π *such that* $\phi(\pi) = 1$ *and accumulator* acc *such that* $\mathsf{D}_{\mathsf{acc}}(\mathsf{acc}) = 1$*, the following holds:*

$$\Pr\left[\begin{array}{c} \mathsf{V}_{\mathsf{acc}}(\mathsf{acc}_1.x, \mathsf{acc}_2.x, \mathsf{pf}) = \mathsf{acc}.x \\ \wedge\ \mathsf{D}_{\mathsf{acc}}(\mathsf{acc}) = 1 \end{array} \middle| \begin{array}{r} \mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{acc}, \mathsf{pf}) \leftarrow \mathsf{P}_{\mathsf{acc}}(\mathsf{st}, \pi, \mathsf{acc}_1) \end{array}\right] = 1$$

Knowledge-soundness. *For every PT adversary* $\mathcal{A}$*, there exists a polynomial-time extractor* Ext *such that the following probability is negligible:*

$$\Pr\left[\begin{array}{c} (D_{\mathsf{acc}}(\mathsf{acc}_1) \neq 1 \vee\ D_{\mathsf{acc}}(\mathsf{acc}_2) \neq 1) \wedge \\ \mathsf{V}_{\mathsf{acc}}(\mathsf{srs}, \mathsf{acc}_1.x, \mathsf{acc}_2.x, \mathsf{pf}) = \mathsf{acc}.x \\ \wedge\ \mathsf{D}_{\mathsf{acc}}(\mathsf{acc}) = 1 \end{array} \middle| \begin{array}{r} \mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{acc}, \mathsf{acc}_1.x, \mathsf{acc}_2.x, \mathsf{pf}) \leftarrow \mathcal{A}(\mathsf{srs}) \\ (\mathsf{acc}_1.w, \mathsf{acc}_2.w) \leftarrow \mathsf{Ext}(\mathsf{acc}, \mathsf{acc}_1.x, \mathsf{acc}_2.x, \mathsf{pf}) \end{array}\right] = 1$$

3.4 Incrementally verifiable computation

Definition 3. *(IVC) An IVC scheme is a tuple of efficient algorithms* $(\mathsf{Setup}_{\mathsf{IVC}}, \mathsf{P}_{\mathsf{IVC}}, \mathsf{V}_{\mathsf{IVC}})$ *with the following interface:*

- $\mathsf{Setup}_{\mathsf{IVC}}(\lambda, S) \to \mathsf{pp}$*: Given a security parameter* λ*, a poly-size bound* $S \in \mathbb{N}$*, outputs public parameters* pp*.*
- $\mathsf{P}_{\mathsf{IVC}}(\mathsf{pp}, F, F.x_i, F.w_i, \pi_i) \to \pi_{i+1}$*: Given public parameters* pp*, a function* $F : \{0,1\}^a \times \{0,1\}^b \to \{0,1\}^a$ *computable by a circuit of size at most* S*, an initial state and claimed output* $F.x_i \in \{0,1\}^a$*, advice* $F.w_i \in \{0,1\}^b$*, and an IVC proof* π_i*, outputs a new IVC proof* π_{i+1}*.*
- $\mathsf{V}_{\mathsf{IVC}}(\mathsf{pp}, F, F.x_i, \pi_i) \to 0/1$*: Given public parameters* pp*, a function* F*, a claimed output* $F.x_i$*, and an IVC proof* π_i*, outputs* 0 *(reject) or* 1 *(accept).*

An IVC scheme satisfies the following properties:

Completeness. *For every poly-size bound* $S \in \mathbb{N}$*,* pp *in the output space of* $\mathsf{Setup}_{\mathsf{IVC}}(\lambda, S)$*, function* $F : \{0,1\}^a \times \{0,1\}^b \to \{0,1\}^a$ *computable by a circuit within bound* S*, collection of elements* $F.x_i \in \{0,1\}^a$*,* $F.w_i \in \{0,1\}^b$ *and IVC proof* π_i*, the following holds:*

$$\Pr\left[\begin{array}{c} \mathsf{V}_{\mathsf{IVC}}(\mathsf{pp}, F, F.x_i, \pi_i) = 1 \\ \Downarrow \\ \mathsf{V}_{\mathsf{IVC}}(\mathsf{pp}, F, F.x_{i+1}, \pi_{i+1}) = 1 \end{array} \middle| \begin{array}{r} \pi_{i+1} \leftarrow \mathsf{P}_{\mathsf{IVC}}(\mathsf{pp}, F, F.x_i, F.w_i, \pi_i), \\ F.x_{i+1} \leftarrow F(F.x_i, F.w_i) \end{array}\right] = 1$$

Knowledge soundness. *Let $S \in \mathbb{N}$ be a poly-size bound and $\ell(\lambda)$ be a polynomial in the security parameter. Let $\mathcal{F}$ be an efficient function sampling adversary that outputs a function $F : \{0,1\}^a \times \{0,1\}^b \to \{0,1\}^a$ computable by a circuit within the poly-size bound S. We say that an IVC scheme is knowledge sound if there exists an efficient extractor* Ext *such that for every efficient IVC prover $\mathsf{P}^*_{\mathsf{IVC}}$ the probability of the following event is greater than $1 - \mathsf{negl}(\lambda)$:*

$$\Pr\left[\begin{array}{c} \mathsf{V}_{\mathsf{IVC}}(\mathsf{pp}, F, F.x_i, \pi_i) = 1 \wedge \\ F.x_i = F(F.x_{i-1}, F.w_{i-1}) \wedge \\ (i = 1 \implies F.x_{i-1} = F.x_0) \wedge \\ (i > 1 \implies \mathsf{V}_{\mathsf{IVC}}(\mathsf{pp}, F, F.x_{i-1}, \pi_{i-1}) = 1) \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}_{\mathsf{IVC}}(\lambda, S) \\ \rho \leftarrow \{0,1\}^{\ell(\lambda)} \\ F \leftarrow \mathcal{F}(\mathsf{pp}; \rho) \\ (F.x_i, \pi_i) \leftarrow \mathsf{P}^*_{\mathsf{IVC}}(\mathsf{pp}, \rho) \\ (F.x_{i-1}, F.w_{i-1}), \pi_{i-1} \leftarrow \mathsf{Ext}(\mathsf{pp}, \rho) \end{array}\right]$$

3.5 IVC from accumulators

Theorem 1. *Let* NARK *be a non-interactive argument that is $(T - t)$-predicate-efficient with respect to Φ. If $(\Phi, \mathsf{Setup}_{\mathsf{NARK}})$ has an accumulation scheme acc_Φ then* NARK *has an accumulation scheme $\mathsf{acc}_{\mathsf{NARK}}$ with the efficiency properties below:*

- $\mathsf{P}_{\mathsf{acc}}$ *runs in time $\sum_{i=1}^{n} T(N, |x_i|)$ plus the time taken to run $\mathsf{P}_{\mathsf{acc},\Phi}$.*
- $\mathsf{V}_{\mathsf{acc}}$ *runs in time $\sum_{i=1}^{n} T(N, |x_i|)$ plus the time taken to run $\mathsf{V}_{\mathsf{acc},\Phi}$*
- $\mathsf{D}_{\mathsf{acc}}$ *takes time equal to $\mathsf{D}_{\mathsf{acc},\Phi}$*

Theorem 2. *There exists a polynomial time transformation T such that if* $\mathsf{NARK} = (\mathsf{Setup}_{\mathsf{NARK}}, \mathsf{P}_{\mathsf{NARK}}, \mathsf{V}_{\mathsf{NARK}})$ *is a SNARK for circuit satisfiability and AS is an accumulation scheme for* NARK*, then* $\mathsf{IVC} = (\mathsf{Setup}_{\mathsf{IVC}}, \mathsf{P}_{\mathsf{IVC}}, \mathsf{V}_{\mathsf{IVC}}) = T(\mathsf{NARK}, \mathsf{AS})$ *is an IVC scheme for constant-depth compliance predicates, provided $\exists \epsilon \in (0,1)$ and a polynomial α s.t. $v^*(\lambda, m, N, \ell) = O(N^{1-\epsilon}.\alpha(\lambda, m, \ell))$. Moreover, if the size of the predicate $\Phi : \mathbb{F}^{(m+2)\ell} \to \mathbb{F}$ is $f = \omega(\alpha(\lambda, m, \ell)^{1/\epsilon})$, then:*

- *The cost of running* $\mathsf{Setup}_{\mathsf{IVC}}$ *is the cost of running* $\mathsf{Setup}_{\mathsf{NARK}}$ *and* $\mathsf{Setup}_{\mathsf{acc}}$ *on an index of size $f + o(f)$.*
- *The cost of running* $\mathsf{P}_{\mathsf{IVC}}$ *is the cost of accumulating m instance-proof pairs using* $\mathsf{P}_{\mathsf{acc}}$*, and running* $\mathsf{P}_{\mathsf{NARK}}$ *on an index of size $f + o(f)$and instance of size $o(f)$.*
- *The cost of running* $\mathsf{V}_{\mathsf{IVC}}$ *is equal to the cost of running* $\mathsf{V}_{\mathsf{NARK}}$ *and* $\mathsf{D}_{\mathsf{acc}}$ *on an index of size $f + o(f)$ and an instance of size $o(f)$.*

4 KZH: An efficiently aggregatable polynomial commitment

We present KZH in Figure 2. For a multilinear polynomial $f(\vec{X})$, where $\vec{X} \in \mathbb{F}^k$, which interpolates a vector in $\mathbb{F}^{2^k}$, we set $\ell = |f|$. We can select any ν, μ such that $k = \nu + \mu$ and the following costs apply:

- Committing to f costs $O(\ell)$ group operations.
- Proof consists of 2^ν $\mathbb{G}_1$ elements and 2^μ field elements.
- Opening requires 2^ν field operations.
- Verifier requires 2^ν pairings, multi-exponentiations of size $2^\nu + 2^\mu$, and 2^μ field operations.

We can set $\nu = \mu = \frac{k}{2}$ so $2^\nu = 2^\mu = 2^{\frac{k}{2}}$, or choose a trade-off between prover and verifier work based on convenience. For example when the verifier workload or proof size is more critical, selecting a lower ν results in fewer pairings and smaller proof size, but at the expense of more multi-exponentiations by the prover to open the commitment. The polynomial commitment scheme has two key properties: The opening proof can be precomputed during the commitment phase and is $O(\ell^{\frac{1}{2}})$ in size. Secondly, it can be accumulated efficiently using only $\mathbb{G}_1$ operation and has an $O(\ell^{\frac{1}{2}})$ accumulator witness. More precisely, the prover commits to the multilinear commitment $f(\vec{X}, \vec{Y})$ with $\vec{X} \in \mathbb{F}^\nu$ and $\vec{Y} \in \mathbb{F}^\mu$ as a matrix of evaluation points. During the opening at point $(\vec{x}_0, \vec{y}_0)$, the prover presents a decomposition of the matrix into row commitments. The correctness of this decomposition can be checked using pairings. Intuitively, KZH is a proof of correct partial evaluation at point $\vec{X} = \vec{x}_0$. Once the verifier is convinced about the correctness of $f^*(\vec{Y}) = f(\vec{x}_0, \vec{Y})$, it can evaluate $f^*(\vec{Y})$ at y_0 on its own. We can then use the technique from Hyrax to evaluate the bivariate commitment as a vector, matrix, or vector product. The technique extends to more variables as presented in Appendix C (which reduces the verification cost). Note that KZH is not inherently hiding; however, by utilizing the general compiler described in [Bü+19] for homomorphic polynomial commitments, it can be transformed to achieve hiding.

Theorem 3. *The protocol in Fig. 2 is a complete and knowledge-sound polynomial commitment scheme as defined in Definition 1, in the AGM under the* dlog *assumption.*

The proof is deferred to Appendix B.1

[d] It is possible to make $\tau^{(i)} = \tau^i$ and $\mathsf{G}^{(j)} = \mu^j \cdot \mathsf{G}$, making the CRS equivalent to the KZG powers-of-tau.

$\mathsf{Setup}_{\mathsf{KZH}}(\lambda, k)$:

- Choose μ, ν such that $k = 2^{\nu+\mu}$. Let $n = 2^{\nu}$ and $m = 2^{\mu}$ and define the boolean cubes as $B_n = \{0,1\}^{\nu}$ and $B_m = \{0,1\}^{\mu}$.
- Sample $\{\mathsf{G}^{(\vec{i})} \leftarrow\$ \mathbb{G}_1\}_{\vec{i} \in B_m}$, $\mathsf{V} \leftarrow\$ \mathbb{G}_2$ and sample trapdoor $\{\tau^{(\vec{j})}\}_{\vec{j} \in B_n}$, $\alpha \leftarrow\$ \mathbb{F}$ [d].
- For $\vec{i} \in B_n$, $\vec{j} \in B_m$, define:
  - $\mathsf{H}^{(\vec{i},\vec{j})} \leftarrow \tau^{(\vec{i})} \times \mathsf{G}^{(\vec{j})}$
  - $\mathsf{H}^{(\vec{j})} \leftarrow \alpha \times \mathsf{G}^{(\vec{j})}$
  - $\mathsf{V}^{(\vec{i})} \leftarrow \tau^{(\vec{i})} \times \mathsf{V} \in \mathbb{G}_2$
  - $\mathsf{V}' \leftarrow \alpha \times \mathsf{V}$
- Let $\mathsf{srs} \leftarrow \left([\mathsf{G}^{(\vec{i})}, \mathsf{H}^{(\vec{i},\vec{j})}, \mathsf{H}^{(\vec{j})}, \mathsf{V}', \mathsf{V}^{(\vec{i})}]_{\vec{i} \in B_n, \vec{j} \in B_m}\right)$.
- Output srs.

$\mathsf{Commit}_{\mathsf{KZH}}(\mathsf{srs}, f(\vec{X}, \vec{Y}))$: For $f \in \mathsf{MLP}(\nu+\mu)$ and $\vec{X} \in \mathbb{F}^{\nu}$, $\vec{Y} \in \mathbb{F}^{\mu}$

- Output $\mathsf{C} \leftarrow \sum_{\vec{x} \in B_n} \sum_{\vec{y} \in B_m} f(\vec{x}, \vec{y}) \times \mathsf{H}^{(\vec{x},\vec{y})}$.
- Let $\mathsf{D}^{(\vec{x})} \leftarrow \sum_{\vec{y} \in B_m} f(\vec{x}, \vec{y}) \times \mathsf{H}^{(\vec{y})}$ $\forall\ \vec{x} \in B_n$.
- Output C and $\mathsf{aux} = \left(\{\mathsf{D}^{(\vec{x})}\}_{\vec{x} \in B_n}\right)$ as cache.

$\mathsf{Open}_{\mathsf{KZH}}(\mathsf{srs}, f(\vec{X}, \vec{Y}), \vec{x_0}, \vec{y_0}, \mathsf{aux})$:

- Let $f^*(\vec{Y}) \leftarrow f(\vec{x_0}, \vec{Y})$, $f^* \in \mathsf{MLP}(\mathbb{F}, \mu)$.
- Let $z_0 \leftarrow f^*(\vec{y_0})$.
- Output $\pi \leftarrow \left(f^*(\vec{Y}), \mathsf{aux} = \{\mathsf{D}^{(\vec{x})}\}_{\vec{x} \in B_n}\right)$, z_0.

$\mathsf{Verify}_{\mathsf{KZH}}(\mathsf{srs}, \mathsf{C}, \vec{x_0}, \vec{y_0}, \pi, z_0)$: Accept if and only if all checks below pass:

1. $e(\mathsf{C}, \mathsf{V}') = \sum_{\vec{x} \in B_n} e(\mathsf{D}^{(\vec{x})}, \mathsf{V}^{(\vec{x})})$,
2. $\sum_{\vec{y} \in B_m} f^*(\vec{y}) \times \mathsf{H}^{(\vec{y})} = \sum_{\vec{x} \in B_n} \mathsf{eq}(\vec{x}, \vec{x_0}) \times \mathsf{D}^{(\vec{x})}$,
3. $f^*(\vec{y_0}) = z_0$.

Figure 2: KZH polynomial commitment scheme

4.1 Accumulator

We build an accumulator for KZH, the polynomial commitment scheme described in Section 4. For the polynomial evaluation predicate, we have that the instance $\pi.x$ and witness $\pi.w$ are as follows:

$$\pi.x = \{\mathsf{C}, \vec{x}_0, \vec{y}_0, z_0\}, \pi.w = \{\vec{\mathsf{D}} := [\mathsf{D}^{(\vec{x})}]_{\vec{x}\in B_n}, f^*(\vec{Y})\}$$

where C is the commitment to $f(\vec{X}, \vec{Y})$, $(\vec{x}_0, \vec{y}_0) \in \mathbb{F}^{\nu+\mu}$ is the opening value, z_0 is claimed to be $z_0 = f(\vec{x}_0, \vec{y}_0)$, and $\pi.w$ is the output of $\mathsf{Open}_{\mathsf{KZH}}$. The accumulator instance and witness are defined as follows, where the red elements only appear in the accumulator and not in a proof:[e]

$$\mathsf{acc}.x = \{\mathsf{C}, T, \vec{x}_0, \vec{y}_0, z_0, E\},$$

$$\mathsf{acc}.w = \{\vec{\mathsf{D}} := [\mathsf{D}^{(\vec{x})}]_{\vec{x}\in B_n}, \vec{f}^*, \mathcal{T}^{(x)}, \mathcal{T}^{(y)}\}$$

for $T \in \mathbb{G}_1$, $\mathcal{T}^{(x)} \in \mathcal{T}(\mathbb{F}, \nu)$, $\mathcal{T}^{(y)} \in \mathcal{T}(\mathbb{F}, \mu)$, and $\vec{f}^*$ which is the vector of the evaluation of $f^*(\vec{Y})$ on the boolean hypercube. We also define the function Dec to represent the checks performed by $\mathsf{Verify}_{\mathsf{KZH}}$. This function computes the error term in the verifier equations, which should evaluate to 0 when evaluated in a fresh proof. Given a tree $\mathcal{T}$ of depth n and a vector $\vec{x} = (x_1, x_2, \ldots, x_n)$, the error tree $\overline{\mathsf{EqTree}}(\mathcal{T}, \vec{x})$ is another tree of depth n constructed as follows: The root node of the error tree is initialized to 0; for each node in $\mathcal{T}$ at depth i with value t, having left and right children with values ℓ and r, respectively, the corresponding nodes in the error tree have left and right child values $\ell - \mathsf{t} \times (1 - x_i)$ and $r - \mathsf{t} \times x_i$, respectively. Now given the following values,

- $\mathcal{T}_x^{(\text{error})} \leftarrow \overline{\mathsf{EqTree}}(\mathcal{T}^{(x)}, \vec{x}_0)$
- $\mathcal{T}_y^{(\text{error})} \leftarrow \overline{\mathsf{EqTree}}(\mathcal{T}^{(y)}, \vec{y}_0)$
- $e'' \leftarrow \langle f^*, \mathcal{T}^{(y)}.\text{leaves}\rangle - z$
- $\mathsf{E}_{\mathbb{G}} \leftarrow \langle \vec{f}^*, (\mathsf{H}^{(\vec{y})})_{\vec{y}\in B_n}\rangle - \langle \mathcal{T}^{(x)}.\text{leaves}, \vec{\mathsf{D}}\rangle$

Dec is defined as it follows:

$$\mathsf{Dec}(\vec{x}_0, \vec{y}_0, z_0, \vec{f}^*, \mathcal{T}^{(x)}, \mathcal{T}^{(y)}, \vec{\mathsf{D}}) \quad = \langle \mathcal{T}_x^{(\text{error})} || \mathcal{T}_y^{(\text{error})} || e'', \vec{\mathsf{K}} || \mathsf{K}'\rangle + \mathsf{E}_{\mathbb{G}}.$$

The algorithms $\mathsf{Setup}_{\mathsf{acc}}$ and $\mathsf{P}_{\mathsf{acc}}$ are described in Figure 3 whereas $\mathsf{V}_{\mathsf{acc}}$ and $\mathsf{D}_{\mathsf{acc}}$ are in Figure 4. In Figure 3, in fact challenge β is the random challenge coming from the verifier in an interactive accumulation protocol, but we directly apply Fiat-Shamir heuristic to make the protocol non-interactive, deriving the random challenge through a random oracle initialized with a hash function. We present an overview of the efficiency of the accumulation scheme below:

[e] Here, proof refers to a fresh accumulator in the context of an accumulator.
[f] Optimized as $\mathsf{E}'' \leftarrow \mathsf{E} + \beta \times (\mathsf{E} - \mathsf{E}') + (1 - \beta)\beta \times \mathsf{Q}$

$\mathsf{Setup}_{\mathsf{acc}}(1^\lambda, k)$**:**

- $\mathsf{srs}_{\mathsf{KZH}} \leftarrow \mathsf{Setup}_{\mathsf{KZH}}(\lambda, k)$.
- Parse n, m from $\mathsf{srs}_{\mathsf{KZH}}$ and generate $\vec{\mathsf{K}} = (\mathsf{K}_1, \ldots, \mathsf{K}_{2\cdot(n+m-1)}) \in \mathbb{G}_1^{2\cdot(n+m-1)}$ and $\mathsf{K}' \leftarrow \mathbb{G}_1$ (unknown DLOG from all other generators).
- Output $\mathsf{srs} = \left(\mathsf{srs}_{\mathsf{KZH}}, \vec{\mathsf{K}}, \mathsf{K}'\right)$.

$\mathsf{P}_{\mathsf{acc}}(\mathsf{srs}, \mathsf{st}, (\pi.x, \pi.w), (\mathsf{acc}_1.x, \mathsf{acc}_1.w))$**:**

- Build accumulator $(\mathsf{acc}_2.x, \mathsf{acc}_2.w)$ from $(\pi.x, \pi.w)$:
  - Parse $(\mathsf{C}_2, \vec{x}_2, \vec{y}_2, z_2) \leftarrow \pi.x$ and $(\{\mathsf{D}_2^{(\vec{x})}\}_{\vec{x}\in B_n}, f_2^*(\vec{Y})) \leftarrow \pi.w$
  - Let $\mathcal{T}_2^{(x)} \leftarrow \mathsf{EqTree}(\vec{x}_2), \mathcal{T}_2^{(y)} \leftarrow \mathsf{EqTree}(\vec{y}_2)$
  - Parse $\mathcal{T}_2^{(x)} \in \mathbb{F}^{2n-1}$, $\mathcal{T}_2^{(y)} \in \mathbb{F}^{2m-1}$ and compute $T_2 \leftarrow \langle \mathcal{T}_2^{(x)} || \mathcal{T}_2^{(y)}, \vec{\mathsf{K}} \rangle$
  - Output $\mathsf{acc}_2.x = \{\mathsf{C}_2, T_2, \vec{x}_2, \vec{y}_2, z_2, 0_{\mathbb{G}}\}$, $\mathsf{acc}_2.w = (\{\mathsf{D}_2^{(\vec{x})}\}_{\vec{x}\in B_n}, \vec{f}_2^*, \mathcal{T}_2^{(x)}, \mathcal{T}_2^{(y)}\}$
- Compute proof pf:
  - Parse $(\mathsf{C}_1, T_1, \vec{x}_1, \vec{y}_1, z_1, \mathsf{E}_1) \leftarrow \mathsf{acc}_1.x$ and $(\{\mathsf{D}_1^{(\vec{x})}\}_{\vec{x}\in B_n}, \vec{f}_1^*, \mathcal{T}_1^{(x)}, \mathcal{T}_1^{(y)}) \leftarrow \mathsf{acc}.w$
  - Set $\mathsf{pf} = \mathsf{Q}$ for $\mathsf{Q} \in \mathbb{G}_1$ such that,

$$\begin{aligned}&\mathsf{Dec}((1-X)\cdot(\vec{x}_1, \vec{y}_1, z_1, \vec{f}_1^*, \mathcal{T}_1^{(x)}, \mathcal{T}_1^{(y)}, \vec{\mathsf{D}}_1) + X\cdot(\vec{x}_2, \vec{y}_2, z_2, \vec{f}_2^*, \mathcal{T}_2^{(x)}, \mathcal{T}_2^{(y)}, \vec{\mathsf{D}}_2))\\ &= (1-X)\times\mathsf{E}_1 + X\times\mathsf{E}_2 + (1-X)\cdot X\times\mathsf{Q}\end{aligned}$$

- Accumulate $(\mathsf{acc}_1.x, \mathsf{acc}_1.w)$ and $(\mathsf{acc}_2.x, \mathsf{acc}_2.w)$ into $(\mathsf{acc}.x, \mathsf{acc}.w)$:
  - Generate challenge $\beta \leftarrow H(\mathsf{acc}_1.x, \mathsf{acc}_2.x, \mathsf{Q})$ through Fiat-Shamir.
  - Compute new error term

$$\mathsf{E} \leftarrow (1-\beta)\times\mathsf{E}_1 + \beta\times\mathsf{E}_2 + (1-\beta)\beta\times\mathsf{Q}^{\mathrm{f}}$$

  - Compute the following linear combinations:

$$(\mathsf{C}, T, \vec{x}, \vec{y}, z) = (1-\beta)\cdot(\mathsf{C}_1, T_1, \vec{x}_1, \vec{y}_1, z_1) + \beta\cdot(\mathsf{C}_2, T_2, \vec{x}_2, \vec{y}_2, z_2)$$

    and set $\mathsf{acc}.x = (\mathsf{C}, T, \vec{x}, \vec{y}, z, \mathsf{E})$
  - Compute the new accumulator witness

$$\mathsf{acc}.w \leftarrow (1-\beta)\cdot(\{\mathsf{D}_1^{(\vec{x})}\}_{\vec{x}\in B_n}, \vec{f}_1^*, \mathcal{T}_1^{(x)}, \mathcal{T}_1^{(y)}) + \beta\cdot(\{\mathsf{D}_2^{(\vec{x})}\}_{\vec{x}\in B_n}, \vec{f}_2^*, \mathcal{T}_2^{(x)}, \mathcal{T}_2^{(y)})$$

  - Output $(\mathsf{acc}.x, \mathsf{acc}.w, \mathsf{pf})$

Figure 3: Setup and prover algorithms for KZH-fold

$\mathsf{V}_{\mathsf{acc}}(\mathsf{srs}, \mathsf{acc}.x_1, \mathsf{acc}.x_2, \mathsf{pf})$:

- Parse $(\mathsf{C}_1, T_1, \vec{x}_1, \vec{y}_1, z_1, \mathsf{E}_1) \leftarrow \mathsf{acc}_1.x$ and $(\mathsf{C}_2, T_2, \vec{x}_2, \vec{y}_2, z_2, \mathsf{E}_2) \leftarrow \mathsf{acc}_2.x$
- Parse $\mathsf{Q} \leftarrow \mathsf{pf}$
- Regenerate challenge $\beta \leftarrow H(\mathsf{acc}_1.x, \mathsf{acc}_2.x, \mathsf{Q})$
- Compute $\mathsf{E} \leftarrow (1-\beta) \times \mathsf{E}_1 + \beta \times \mathsf{E}_2 + (1-\beta) \cdot \beta \times \mathsf{Q}$ and

$$(\mathsf{C}, T, \vec{x}, \vec{y}, z) = (1-\beta) \cdot (\mathsf{C}_1, T_1, \vec{x}_1, \vec{y}_1, z_1) + \beta \cdot (\mathsf{C}_2, T_2, \vec{x}_2, \vec{y}_2, z_2)$$

- Output $\mathsf{acc}.x = (\mathsf{C}, T, \vec{x}, \vec{y}, z, \mathsf{E})$

$\mathsf{D}_{\mathsf{acc}}(\mathsf{srs}, \mathsf{acc}.x, \mathsf{acc}.w)$:

- Parse $(\mathsf{C},\ T,\ \vec{x}_0, \vec{y}_0, z_0, \mathsf{E}) \leftarrow \mathsf{acc}.x$ and $(\{\mathsf{D}^{(\vec{x})}\}_{\vec{x} \in B_n}, \vec{f}^*,\ \mathcal{T}^{(x)}, \mathcal{T}^{(y)}) \leftarrow \mathsf{acc}.w$
- Output 1 if and only if all following checks pass, otherwise output 0.

  (i) $\sum_{\vec{x} \in B_n} e(\mathsf{D}^{(\vec{x})}, \mathsf{V}^{(\vec{x})}) \stackrel{?}{=} e(\mathsf{C}, \mathsf{V}')$

  (ii) $\langle (\mathcal{T}^{(x)} || \mathcal{T}^{(y)}), \vec{\mathsf{K}} \rangle \stackrel{?}{=} T$

  (iii) $\mathsf{Dec}(\vec{x}_0, \vec{y}_0, z_0, \vec{f}^*, \mathcal{T}^{(x)}, \mathcal{T}^{(y)}, \vec{\mathsf{D}}) \stackrel{?}{=} \mathsf{E}$

Figure 4: Verifier and decider algorithms in KZH-fold

Communication. The size of the accumulation witness is $O(n+m) = O(\ell^{\frac{1}{2}})$. The accumulator instance is constant in size. The communication is significantly lower than Nova, Halo Infinite, and Protostar, where it is $\Theta(\ell)$. Using the generalization to multivariate polynomial commitments in Appendix C, we can reduce the communication to $O(k \cdot \ell^{\frac{1}{k}})$

Decider complexity. The decider is essentially has the same characteristic of KZH verifier, and thus runs in time $O(\ell^{\frac{1}{2}})$ or $O(k \cdot \ell^{\frac{1}{k}})$ in the generalized case.

Prover complexity. The accumulation prover performs $O(n+m) = O(\ell^{\frac{1}{2}})$ operations to combine the two witnesses and compute the cross-term Q. This contrasts with previous approaches - namely Nova, Halo Infinite, and Protostar - which require a linear number of operations.

Verifier complexity. The accumulation verifier performs $3-4$ $\mathbb{G}_1$ exponentiations, along with a constant number of field operations, which is consistent with schemes like Nova (requiring $2-3$ $\mathbb{G}_1$ operations) and Protostar (requiring $3-4$ $\mathbb{G}_1$ operations).

Theorem 4. *The protocol in Figures 3 and 4 is an accumulator scheme satisfying completeness and knowledge soundness as in Definition 2, in the AGM under the* dlog *assumption.*

The proof is deferred to Appendix B.2. Intuitively, correctness follows since $\mathsf{V_{acc}}$ and $\mathsf{D_{acc}}$ together go through the same computations as $\mathsf{P_{acc}}$, and thus the outputs are the same. For soundness, note that if decider's first check passes, since β is computed after the prover outputs acc_1 and acc_2, it implies the first check of the KZH verifier is satisfied for $\mathsf{C}_1, \{\mathsf{D}_1^{(\vec{x})}\}$ and for $\mathsf{C}_2, \{\mathsf{D}_2^{(\vec{x})}\}$. For the other two KZH verifier checks, the decider computes the error terms and verifies their consistency with the error term computed by $\mathsf{P_{acc}}$. Again, since β depends on the outputs of $\mathsf{P_{acc}}$, the error terms of acc_1 and acc_2 are correctly computed and thus the KZH checks are satisfied.

4.2 An IVC scheme from KZH

At the beginning of this Section, we introduced a polynomial commitment scheme and built an accumulation scheme for its verifier, i.e. the polynomial evaluation predicate in Section 4.1. Notably, many modern SNARK constructions have succinct verifiers when given oracle access to a polynomial commitment scheme capable of proving polynomial evaluations. In Appendix D, we introduce a PIOP for R1CS inspired by Spartan with these characteristics. Thus, from Theorem 1 in Section 3.5, an accumulation scheme for KZH implies an accumulation scheme for our SNARK. Next, from Theorem 2 there exists an efficient transformation that takes the SNARK and its accumulation scheme and constructs an IVC scheme $\mathsf{IVC} = (\mathsf{Setup_{IVC}}, \mathsf{P_{IVC}}, \mathsf{V_{IVC}})$. An overview of the IVC step function can be seen in Figure 5. In the next section, we will use the IVC scheme to build an efficient

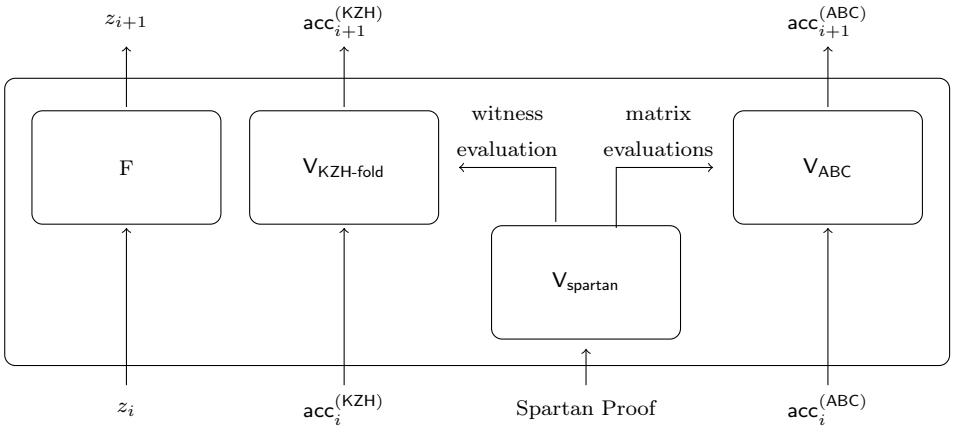

Figure 5: KZH-fold step (augmented) circuit

signature aggregation scheme. Our IVC scheme is not zero knowledge; however, we believe that one can use the techniques presented in HyperNova [KS23b] to make our IVC scheme zero knowledge, we leave this to future work.

4.3 Non-uniform IVC

Non-uniform IVC [KS22b] extends traditional IVC by allowing each step to execute one of several predefined instructions $F_1, F_2, \ldots, F_k$ rather than a single instruction F. Earlier implementations relied on a universal circuit that computes all instructions and selects the output based on the program counter, resulting in inefficiency as the prover computes every instruction, even when only one is needed. SuperNova improves this by maintaining a running accumulator for each instruction and using memory techniques (e.g., Merkle trees) to select and accumulate only the relevant accumulator at each step, reducing computational effort. However, the witness size grows linearly with the combined sizes of all instruction witnesses. Protostar [BC23] offers a similar improvement, leveraging the fact that committing to zeros incurs no additional cost. While it also reduces computational overhead, like SuperNova, it still requires the prover to manage a witness size that scales linearly with the sum of all instruction witnesses.

We propose an alternative approach to achieve non-uniform IVC, in high level by accumulating polynomials corresponding to F_i rather than accumulating circuit F_i (i.e. its R1CS representation) directly. The key insight is that any polynomial of degree $d_i < D$ can be padded to become a polynomial of degree D, allowing it to be accumulated with a running polynomial of degree D. We maintain a running accumulator corresponding to a polynomial of degree D and use it to accumulate PCS opening statements of degree d_i (e.g., opening a witness commitment at a random point) required by the Spartan verifier. This

strategy is compatible with any polynomial accumulation scheme that features a sublinear accumulation verifier. Unlike Protostar and SuperNova, our method achieves sublinear witness size and does not require the PCS to be homomorphic, similar to Protostar. We defer its details to Appendix E. In our approach similar to SuperNova and Protostar, the decider time still depends on the number of instructions. We leave it to future work on how to make the decider time independent of the number of instructions.

5 PIOP for signature aggregation protocol

In Figure 7 and 8, we provide a protocol enabling bandwidth-efficient recursive aggregation of accountable signatures. A distinctive feature of our protocol is that the circuit size remains independent of the number of signers, made possible by the homomorphic properties of our KZH commitment scheme. Our protocol uses - as building blocks - an aggregate signature scheme $(\mathsf{KGen}_{\mathsf{ss}}, \mathsf{Sign}_{\mathsf{ss}}, \mathsf{Verify}_{\mathsf{ss}})$ (Definition 4), the IVC scheme $(\mathsf{Setup}_{\mathsf{IVC}}, \mathsf{P}_{\mathsf{IVC}}, \mathsf{V}_{\mathsf{IVC}})$ of Section 4.2, and our polynomial commitment scheme $\mathsf{KZH} = (\mathsf{Setup}_{\mathsf{KZH}}, \mathsf{Commit}_{\mathsf{KZH}}, \mathsf{Open}_{\mathsf{KZH}}, \mathsf{Verify}_{\mathsf{KZH}})$. Our scheme also uses the famous sumcheck protocol $(\mathsf{P}_{\mathsf{smck}}, \mathsf{V}_{\mathsf{smck}})$ [Lun+90]. We implement our signature aggregation protocol using BLS as the signature scheme [BLS04], and present its efficiency, along with a comparison to the state of the art, in Section 6. To track the signers, we use a bitvector: a vector $\vec{b}$ with a size equal to the number of validators such that $b_k = 1$ if user k has signed, and 0 otherwise. Here $\langle k \rangle$ is the μ-bit binary representation of k. The circuit size of the scheme is constant, *independent* of the number of validators, enabling the recursive aggregation of signatures for millions of validators with minimal overhead. This is achieved via utilizing the IVC scheme from Section 4.2. The scheme proves the union of the bitvector by relying on the fact that $\vec{b}_1 \vee \vec{b}_2 = \vec{b}_1 + \vec{b}_2 - \vec{b}_1 \circ \vec{b}_2$. In Figure 8, we build a simple sumcheck-based PIOP for this statement. We compile this PIOP into a proof system using KZH, and accumulate the resulting evaluation checks.

The Setup algorithm of the signature aggregation scheme initializes the IVC scheme. Users use KGen to generate their public and signing key pair $(\mathsf{sk}, \mathsf{pk})$. $\mathsf{Initalize}$ initializes the vector of all public keys from the users in the system. To sign, users run $\mathsf{Sign}_{\mathsf{ss}}$. Next, user k runs $\mathsf{SigToAggSig}$ to convert their signature into aggregated form. $\mathsf{Aggregate}$ is similar to an accumulation scheme and is run by a prover and a verifier: it takes two aggregated signatures as input and outputs a new one. The aggregated signature includes an IVC proof π_{IVC} for the function F, which is the KZH verifier. That is, $\pi_{\mathsf{IVC}}.x$ consists of the polynomial evaluation claims (including F and $F.x$) whereas $\pi_{\mathsf{IVC}}.w$ is a proof that they have been aggregated correctly. Finally, Verify (intuitively, the decider of the accumulation scheme) is run to check the validity of aggregated signatures. We provide an overview of the resulting IVC circuit in Figure 6.

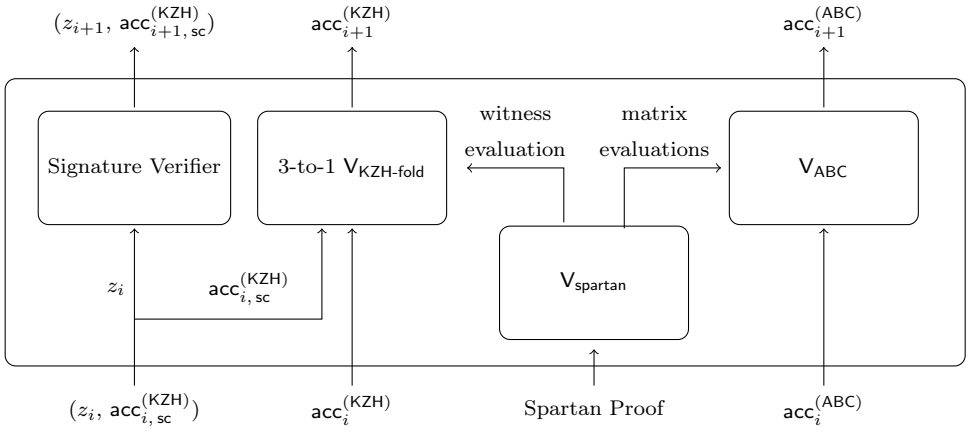

Figure 6: Signature aggregation augmented circuit

Efficiency. The aggregate signature consists of the public key, the signature and a polynomial that interpolates the bitvector of signers, as well as an IVC proof (which contains PCS evaluation claims). Using the IVC protocol, induced by KZH-fold from Section 4.1, the IVC proof is $O(\ell^{\frac{1}{2}})$ in size. Thus the aggregate, accountable signature size is dominated by the bitvector of signers $\vec{c}$, which contains at most ℓ bits. This is the minimal information that can be transmitted for an accountable signature. The aggregator's work consists of computing and committing to $\vec{c}$ which takes at most ℓ group additions (not scalar multiplications as $\vec{c}$ consists of bits), as well as the work of running the sumcheck prover and the accumulation prover. Both of these are linear prover time and do not require additional commitments. The aggregation verifier, which is implemented as a recursive circuit in the IVC protocol, consists of a constant number of group operations and $\log(\ell)$ native field operations and hashes. These are mostly used to verify the accumulation of PCS evaluation claims. The recursive circuit for each leaf also needs to check that the signer polynomial $\vec{b}$ is instantiated correctly and that the correct public key is being aggregated. This consists of a single vector commitment check (can be outsourced using a PC) and an evaluation of two Lagrange polynomials. Using the extension from Section C, the communication cost can be lowered to $O(k \cdot \ell^{\frac{1}{k}}) + \ell$ bits, however, the aggregator's computation overhead is dominated by opening a KZH-k polynomial commitment of size ℓ, which still costs $O(\ell^{\frac{1}{2}})$ group scalar multiplications.

$\mathsf{Setup}(1^\lambda, n)$**:**

- Sample $\{(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)\} \leftarrow\!\!\$\ \mathsf{GGen}(1^\lambda)$
- $\mathsf{G} \leftarrow\!\!\$\ \mathbb{G}_1$
- $\mathsf{srs}_{\mathsf{IVC}} \leftarrow \mathsf{Setup}_{\mathsf{IVC}}(\lambda)$
- Output $\mathsf{srs} = (\mathsf{srs}_{\mathsf{IVC}}, \mathsf{G})$

$\mathsf{KGen}(\lambda)$**:** $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{ss}}(1^\lambda, n)$

$\mathsf{Intialize}(\mathsf{srs}_{\mathsf{PC}}, [\mathsf{pk}_i]_{i=1}^n)$**:**

Compute vector commitment VC such that $\mathsf{VC}[k] = \mathsf{pk}_k$.

$\mathsf{Sign}(\mathsf{pk}, M \in \{0,1\}^*)$ **:** $\sigma \leftarrow \mathsf{Sign}_{\mathsf{ss}}(\mathsf{sk}, M)$

$\mathsf{SigToAggSig}(k \in [n \cdot m], \mathsf{pk}_k, \sigma_k)$**:** user k prepares its signature to aggregate

- Set $\vec{b} = \langle k \rangle \in \{0,1\}^\mu$
- $\mathsf{B}_k \leftarrow \mathsf{Commit}_{\mathsf{KZH}}(\mathsf{srs}_{\mathsf{KZH}}, b(\vec{X}))$
- Set IVC proof $\pi_{\mathsf{IVC}}^{(k)} = \perp$
- Output:

$$A_k = (\mathsf{pk}_k, \mathsf{B}_k, \pi_{\mathsf{IVC},k}.x, b_k(\vec{X}), \sigma_k, \pi_{\mathsf{IVC}}^{(k)}.w)$$

$\mathsf{Verify}(\mathsf{srs}, A_k)$**:** Verify an aggregated signature

- Check $\mathsf{B} = \mathsf{Commit}_{\mathsf{KZH}}(\mathsf{srs}_{\mathsf{KZH}}, b(\vec{X}))$.
- Check $\mathsf{V}_{\mathsf{ss}}(\mathsf{pk}, M, \sigma) = 1$
- Check $\mathsf{Verify}_{\mathsf{IVC}}(\mathsf{srs}_{\mathsf{IVC}}, F, F.x, \pi_{\mathsf{IVC}}) = 1$

Figure 7: Signature aggregation protocol - 1

$\mathsf{Aggregate}(A_{k_1}, A_{k_2})$**:**

Aggregate the signatures:

- $A_{k_1} = (\mathsf{pk}_{k_1}, \mathsf{B}_{k_1}, \pi_{\mathsf{IVC}}^{(k_1)}.x, \sigma_{k_1}, b_{k_1}(\vec{X}), \pi_{\mathsf{IVC}}^{(k_1)}.w)$
- $A_{k_2} = (\mathsf{pk}_{k_2}, \mathsf{B}_{k_2}, \pi_{\mathsf{IVC}}^{(k_2)}.x, \sigma_{k_2}, b_{k_2}(\vec{X}), \pi_{\mathsf{IVC}}^{(k_2)}.w)$
- Set $\mathsf{pk}' \leftarrow \mathsf{pk}_{k_1} + \mathsf{pk}_{k_2}$
- Set $\sigma' \leftarrow \sigma_{k_1} + \sigma_{k_2}$

Proof of well-formedness of new bitvector

- Compute $c(\vec{X})$ such that $c(\vec{x}) = b_{k_1}(\vec{x}) \vee b_{k_2}(\vec{x})\ \forall \vec{x} \in \{0,1\}^\mu$
- Send $\mathsf{C} \leftarrow \mathsf{Commit}_{\mathsf{KZH}}(\mathsf{srs}_{\mathsf{KZH}}, c(\vec{X}))$
- Verifier sends challenge $\vec{r} \leftarrow \mathbb{F}^\mu$
- Define $c(\vec{X}) = b_{k_1}(\vec{X}) + b_{k_2}(\vec{X}) - b_{k_1}(\vec{X}) \cdot b_{k_2}(\vec{X})$ and run the sumcheck to prove that

$$\sum_{\vec{x} \in \{0,1\}^\mu} eq(\vec{x}, \vec{r})(b_{k_1}(\vec{x}) + b_{k_2}(\vec{x}) - b_{k_1}(\vec{x}) \cdot b_{k_2}(\vec{x}) - c(\vec{x})) = 0$$

  Output $b_{k_1}(\vec{\rho})$, $b_{k_2}(\vec{\rho})$, $\vec{c}(\vec{\rho})$ where $\vec{\rho} \in \mathbb{F}^k$ is the vector of randomness sampled by the verifier during the sumcheck.

- Verifier sends random challenges α_1, α_2 from $\mathbb{F}$
- Prover computes the polynomial $p(\vec{X}) = b_{k_1}(\vec{X}) + \alpha_1 b_{k_2}(\vec{X}) + \alpha_2 c(\vec{X})$, runs $(\pi, z_0) \leftarrow \mathsf{Open}_{\mathsf{KZH}}(\mathsf{srs}_{\mathsf{KZH}}, p(\vec{X}), \vec{x}_0, \vec{y}_0, \mathsf{aux})$, for $\vec{x}_0 || \vec{y}_0 = \vec{\rho}$
- Verifier adds evaluation claim $(\mathsf{P} = \mathsf{B}_{k_1} + \alpha_1 \mathsf{B}_{k_2} + \alpha_2 \mathsf{C}, \vec{x}_0, \vec{y}_0, \pi, z_0)$ to $\pi'_{\mathsf{IVC}}.x$

New Aggregated Signature:

- $\pi'_{\mathsf{IVC}} \leftarrow \mathsf{P}_{\mathsf{IVC}}(\mathsf{srs}_{\mathsf{IVC}}, F, F.x, F.w, \pi_{\mathsf{IVC}})$
- Let $A' = (\mathsf{pk}', \sigma', \mathsf{C}, \pi'_{\mathsf{IVC}}.x; \sigma', c(\vec{X}), \pi'_{\mathsf{IVC}}.w)$
- Output A'

Figure 8: Signature aggregation protocol - 2

6 Implementation and efficiency

We implement all our subprotocols in Rust, by leveraging the arkworks library[g]. Our CycleFold module builds on the implementation from the Nexus zkVM project[h], and our Spartan PIOP module builds on the original Spartan codebase[i]. We made our implementation publicly available as an open-source library[j].

The accumulation verifier circuit for PCD (accumulating two accumulators) in KZH2-fold and KZH3-fold schemes, respectively requires four and five scalar multiplications, implemented using CycleFold [KS23a] and Ova [Ova]. The total circuit size for KZH2-fold and KZH3-fold verifiers are approximately 60k and 73k constraints on the primary curve and 12k and 15k on the secondary curve, with about 40% of constraint on the primary curve dedicated to hashing non-native field elements. Our implementation is not highly optimized. Inspired by Nexus, we also implemented a Nova circuit for IVC, accumulating one fresh proof with a running accumulation, resulting in a circuit size of 35k constraints on the primary curve and 6k on the secondary curve[k]. This comparison aligns with expectations, for example, KZH2-fold verifier is naturally larger, requiring three to four group scalar multiplications compared to two to three for Nova.

As part of our implementation, we built an augmented circuit for our signature aggregation protocol as seen in Figure 6, along with a smaller augmented circuit for our KZH-based folding scheme for NP. We used the R1CS PIOP from Section D to prove our circuits. We ran our benchmarks on a laptop with an Intel i7-1370 CPU and 32GB of RAM. We used the half-pairing cycles of BN254 and Grumpkin as our primary and secondary curves. We present our results in the following sections.

6.1 Efficiency of KZH

In Figure 9, we provide benchmarks for our variants of KZH-2, KZH-3 and KZH-4 polynomial commitment schemes and compare them with the celebrated KZG scheme. KZG is efficient and offers constant verification time, while KZH benefits from faster opening times and supports a natural accumulation scheme.

[g] https://arkworks.rs

[h] https://nexus-xyz.github.io/assets/nexus_whitepaper.pdf

[i] https://github.com/microsoft/Spartan

[j] https://github.com/asn-d6/kzh_fold

[k] Unlike our implementation and Nexus, the original Nova implementation by Microsoft does not use Arkworks. We believe a primary reason for our circuit's inefficiency is likely due to inefficiencies in Arkworks' implementation of group operations and non-native field arithmetic in R1CS. For example, a group scalar multiplication by Nova reportedly takes 1k constraints while it takes 3k with Arkworks.

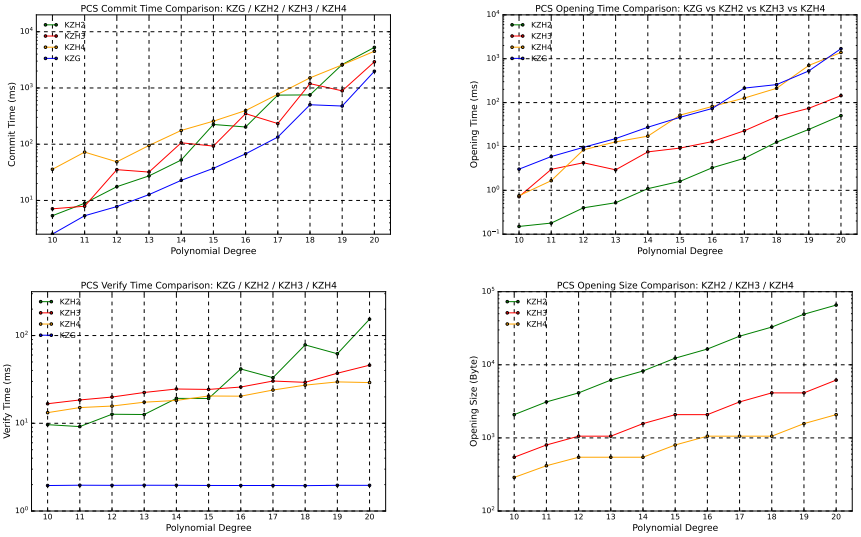

Figure 9: KZH benchmarks and comparison with KZG. Polynomial degree for KZH-k refers to k^m, where m is the number of variables.

6.2 Comparison with Halo Infinite

Halo Infinite [Bon+21] (HI) presents an accumulation scheme for arbitrary homomorphic polynomial opening aggregation schemes. The prover aggregates n polynomial openings taking the polynomials themselves as input. In this section, we compare the efficiency of KZH2-fold, KZH3-fold and HI by implementing all three schemes in our codebase.

In Figure 10, we observe that KZH-fold's prover is significantly faster than HI's. For large witness sizes, HI spends most of its time committing to the polynomial $q(x)$. On the other hand, we also see that HI's accumulation verifier is significantly faster compared to KZH-fold. Finally, we see how KZH-fold's communication overhead scales far more efficiently as the polynomial's size increases, since in HI's private aggregation scheme the prover must transmit the entire polynomial to the aggregator.

6.3 Comparison with Nova

We implemented our IVC scheme based on KZH2-fold and KZH3-fold, and then compared it to our implementation of Nova[1] using different-sized F circuits in Table 3. For the purposes of IVC, we use a circuit F that iteratively computes Poseidon hashes, and the first column contains the number of Poseidon invocations.

At the end of the R1CS PIOP protocol, the decider must evaluate the R1CS matrices A,

[1]Primarily borrowed from Nexus zkVM project

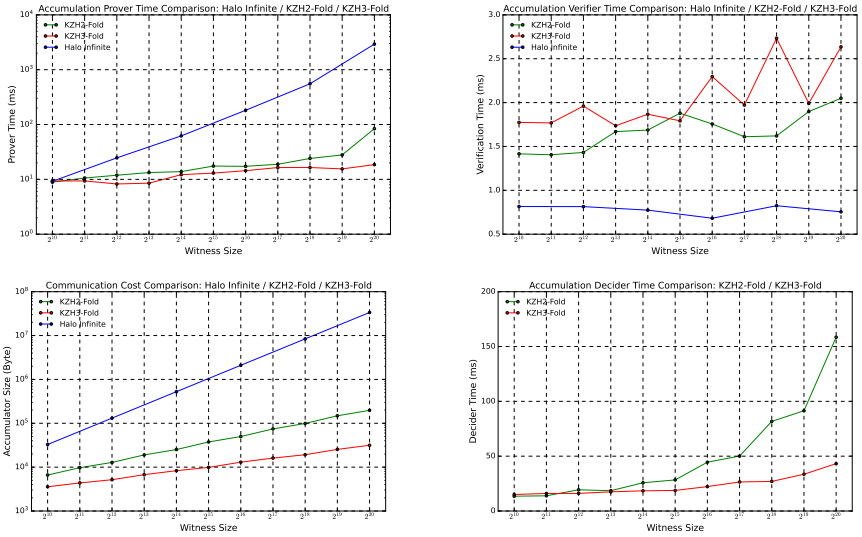

Figure 10: KZH-fold performance and communication cost, along with a comparison with Halo Infinite private aggregation

B and C at a random point. We outsource this computation by having the prover provide an opening proof using KZH. This outsourcing is independent of the rest of the protocol, allowing any polynomial commitment scheme to be used. The commitment to the matrices, which the costly part, can be performed during the setup phase.

As expected, our prover is slower than Nova, with a factor of almost 3 for moderate computations. However, our accumulator size is more compact and our verifier times are faster. Our prover is slower for two reasons. First, Nova's prover cost is essentially two MSMs, whereas our prover uses KZH to commit to the witness. Furthermore, our augmented circuit must partially verify the Spartan proof's sumcheck and the accumulation of the matrix evaluations (Appendix D). We believe that a large part of the slowdown is due to unoptimized code which can be improved.

6.4 Comparison with BLS aggregation

We implemented our KZH3-based accountable signature aggregation scheme from Figure 8 and compared it to the accountable BLS signature aggregation scheme in Table 4. In the BLS scheme, communication cost scales with the size of the multiplicity vector, which incurs a redundancy overhead of $r \cdot \log d$, where r is the number of recursive aggregation layers and d is the number of aggregators per layer. For an aggregation scheme with a single layer ($r = 1$) with 1 million validators, the multiplicity vector requires 128 kB multiplied by $\log d$. With additional recursive layers, this cost increases linearly with r, further exacer-

# of H	Scheme	Prover	Verifier	|Acc|	# Constraints
0	KZH2-fold	1.1 s	103 ms	74 KB	$2^{17} \approx 131k$
	KZH3-fold	0.87 s	62 ms	16 KB	$2^{17} \approx 131k$
	Nova	157 ms	250 ms	3.8 MB	75k
150	KZH2-fold	4.0 s	133 ms	148 KB	$2^{19} \approx 524k$
	KZH3-fold	3.1 s	63 ms	25 KB	$2^{19} \approx 524k$
	Nova	447 ms	704 ms	9.6 MB	165k
1000	KZH2-fold	7.2 s	193 ms	197 KB	$2^{20} \approx 1048k$
	KZH3-fold	6.68 s	93 ms	31 KB	$2^{20} \approx 1048k$
	Nova	2.0 s	3.8 s	42 MB	675k
2000	KZH2-fold	23.1 s	256 ms	295 KB	$2^{21} \approx 2097k$
	KZH3-fold	16.5 s	135 ms	37 KB	$2^{21} \approx 2097k$
	Nova	4.8 s	5.6 s	80.8 MB	1185k

Table 3: Comparison of IVC schemes based on KZH2-fold, KZH3-fold and Nova

bating bandwidth requirements for larger validator sets. Verification involves a multiscalar multiplication (MSM) to compute the aggregated public key, using the multiplicity vector and validators' public keys.

In contrast, the KZH3-based scheme eliminates the need for multiplicity vectors, making its communication cost independent of the number of recursive layers and the number of aggregators. The communication cost of the KZH3-based scheme is dominated by the participation bitfield, whereas the recursive proof itself is less than 40% of the overall size. Our signature aggregation augmented circuit is described in approximately 2^{19} constraints. We find that the witness for this circuit is low-weight (with about 33% of the entries being zero or one). As a result, committing to this low-weight witness is significantly more efficient compared to committing to a random vector of the same size. To exploit this property, we pair KZH3-fold with the Spartan PIOP, which only requires computing a single commitment to the witness vector.

Table 4 highlights the communication and verification costs of both schemes across different validator counts and recursive layers. The KZH3-based approach maintains consistent communication costs regardless of r, while BLS incurs significant growth due to the $r \cdot \log d$ multiplicative factor.

# of validators	Scheme	Communication	Verifier
2^{20}	BLS ($r = 1, d = 16$)	512 kB	338 ms
	BLS ($r = 4, d = 16$)	2 MB	340 ms
	BLS ($r = 4, d = 32$)	4 MB	342 ms
	Ours	**205 kB**	**226 ms**
2^{21}	BLS ($r = 1, d = 16$)	1 MB	669 ms
	BLS ($r = 4, d = 16$)	4 MB	670 ms
	BLS ($r = 4, d = 32$)	8 MB	673 ms
	Ours	**333 kB**	**276 ms**
2^{22}	BLS ($r = 1, d = 16$)	2 MB	1.29 s
	BLS ($r = 4, d = 16$)	8 MB	1.3 s
	BLS ($r = 4, d = 32$)	16 MB	1.3 s
	Ours	**589 kB**	**322 ms**

Table 4: Comparison of BLS and KZH3-based accountable signature aggregation schemes

References

[CT10] Alessandro Chiesa and Eran Tromer. "Proof-Carrying Data and Hearsay Arguments from Signature Cards". In: 2010, pp. 310–331.

[CTV15] Alessandro Chiesa, Eran Tromer, and Madars Virza. "Cluster Computing in Zero Knowledge". In: 2015, pp. 371–403. DOI: 10.1007/978-3-662-46803-6_13.

[Her24] Polygon Hermez. *Polygon zkevm: Recursion, aggregation and composition of proofs*. https://github.com/0xPolygonHermez/zkevm-techdocs/blob/main/docs/proof-recursion.pdf. Accessed: 2024-11-15. 2024. URL: https://github.com/0xPolygonHermez/zkevm-techdocs/blob/main/docs/proof-recursion.pdf.

[KB23] Assimakis Kattis and Joseph Bonneau. "Proof of Necessary Work: Succinct State Verification with Fairness Guarantees". In: 2023, pp. 18–35. DOI: 10.1007/978-3-031-47751-5_2.

[Ben+14] Eli Ben-Sasson et al. "Scalable Zero Knowledge via Cycles of Elliptic Curves". In: 2014, pp. 276–294. DOI: 10.1007/978-3-662-44381-1_16.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. "Nova: Recursive Zero-Knowledge Arguments from Folding Schemes". In: 2022, pp. 359–388. DOI: 10.1007/978-3-031-15985-5_13.

[Bün+21] Benedikt Bünz et al. "Proof-Carrying Data Without Succinct Arguments". In: 2021, pp. 681–710. DOI: 10.1007/978-3-030-84242-0_24.

[KS24] Abhiram Kothapalli and Srinath T. V. Setty. "HyperNova: Recursive Arguments for Customizable Constraint Systems". In: 2024, pp. 345–379. DOI: 10.1007/978-3-031-68403-6_11.

[BC23] Benedikt Bünz and Binyi Chen. "Protostar: Generic Efficient Accumulation/Folding for Special-Sound Protocols". In: 2023, pp. 77–110. DOI: 10.1007/978-981-99-8724-5_3.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: 2010, pp. 177–194. DOI: 10.1007/978-3-642-17373-8_11.

[Bün+20] Benedikt Bünz et al. "Recursive Proof Composition from Accumulation Schemes". In: 2020, pp. 1–18. DOI: 10.1007/978-3-030-64378-2_1.

[Set20] Srinath Setty. "Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup". In: 2020, pp. 704–737. DOI: 10.1007/978-3-030-56877-1_25.

[PST13] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. "Signatures of Correct Computation". In: 2013, pp. 222–242. DOI: 10.1007/978-3-642-36594-2_13.

[Lee21] Jonathan Lee. "Dory: Efficient, Transparent Arguments for Generalised Inner Products and Polynomial Commitments". In: 2021, pp. 1–34. DOI: 10.1007/978-3-030-90453-1_1.

[KS23a] Abhiram Kothapalli and Srinath Setty. *CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves*. Cryptology ePrint Archive, Report 2023/1192. 2023. URL: https://eprint.iacr.org/2023/1192.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. "Transparent SNARKs from DARK Compilers". In: 2020, pp. 677–706. DOI: 10.1007/978-3-030-45721-1_24.

[KS22a] Abhiram Kothapalli and Srinath Setty. *SuperNova: Proving universal machine executions without universal circuits*. Cryptology ePrint Archive, Report 2022/1758. 2022. URL: https://eprint.iacr.org/2022/1758.

[ZSC24] Jiaxing Zhao, Srinath Setty, and Weidong Cui. *MicroNova: Folding-based arguments with efficient (on-chain) verification*. Cryptology ePrint Archive, Paper 2024/2099. 2024. URL: https://eprint.iacr.org/2024/2099.

[Che+20] Weikeng Chen et al. *Reducing Participation Costs via Incremental Verification for Ledger Systems*. Cryptology ePrint Archive, Paper 2020/1522. 2020. URL: https://eprint.iacr.org/2020/1522.

[Bon+20] Joseph Bonneau et al. *Coda: Decentralized Cryptocurrency at Scale*. Cryptology ePrint Archive, Paper 2020/352. 2020. URL: https://eprint.iacr.org/2020/352.

[Gro+18] Jens Groth et al. "Updatable and Universal Common Reference Strings with Applications to zk-SNARKs". In: 2018, pp. 698–728. DOI: 10.1007/978-3-319-96878-0_24.

[Resb] Ethereum Research. *Sticking to 8192 Signatures per Slot Post-SSF: How and Why*. https://ethresear.ch/t/sticking-to-8192-signatures-per-slot-post-ssf-how-and-why/. Accessed: 2024-11-15.

[Sin] *Single Slot Finality*. https://ethereum.org/en/roadmap/single-slot-finality/. Accessed: 2024-11-15.

[But] Vitalik Buterin. *Possible futures of the Ethereum protocol, part 1: The Merge*. https://vitalik.eth.limo/general/2024/10/14/futures1.html. Accessed: 2024-11-15.

[D'A+24b] Francesco D'Amato et al. *TOB-SVD: Total-Order Broadcast with Single-Vote Decisions in the Sleepy Model*. 2024. arXiv: 2310.11331 [cs.DC]. URL: https://arxiv.org/abs/2310.11331.

[DZ23] Francesco D'Amato and Luca Zanolini. *A Simple Single Slot Finality Protocol For Ethereum*. Cryptology ePrint Archive, Report 2023/280. 2023. URL: https://eprint.iacr.org/2023/280.

[D'A+24a] Francesco D'Amato et al. *3-Slot-Finality Protocol for Ethereum*. 2024. arXiv: 2411.00558 [cs.DC]. URL: https://arxiv.org/abs/2411.00558.

[D'A+22] Francesco D'Amato et al. *No More Attacks on Proof-of-Stake Ethereum?* Cryptology ePrint Archive, Report 2022/1171. 2022. URL: https://eprint.iacr.org/2022/1171.

[Gil+17] Yossi Gilad et al. *Algorand: Scaling Byzantine Agreements for Cryptocurrencies*. Cryptology ePrint Archive, Report 2017/454. 2017. URL: https://eprint.iacr.org/2017/454.

[Yin+19] Maofan Yin et al. "HotStuff: BFT Consensus with Linearity and Responsiveness". In: 2019, pp. 347–356. DOI: 10.1145/3293611.3331591.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: 17.4 (Sept. 2004), pp. 297–319. DOI: 10.1007/s00145-004-0314-9.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. "Compact Multi-signatures for Smaller Blockchains". In: 2018, pp. 435–464. DOI: 10.1007/978-3-030-03329-3_15.

[Resa] Ethereum Research. *Signature Merging for Large-Scale Consensus*. https://ethresear.ch/t/signature-merging-for-large-scale-consensus/17386. Accessed: 2024-11-15.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. *Halo: Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Report 2019/1021. 2019. URL: https://eprint.iacr.org/2019/1021.

[Wu+18] Howard Wu et al. *DIZK: A Distributed Zero Knowledge Proof System*. Cryptology ePrint Archive, Paper 2018/691. 2018. URL: https://eprint.iacr.org/2018/691.

[Liu+23] Tianyi Liu et al. *Pianist: Scalable zkRollups via Fully Distributed Zero-Knowledge Proofs*. Cryptology ePrint Archive, Paper 2023/1271. 2023. URL: https://eprint.iacr.org/2023/1271.

[Ros+24] Michael Rosenberg et al. *Hekaton: Horizontally-Scalable zkSNARKs via Proof Aggregation*. Cryptology ePrint Archive, Paper 2024/1208. 2024. URL: https://eprint.iacr.org/2024/1208.

[Wan+24] Wenhao Wang et al. Cirrus*: Performant and Accountable Distributed SNARK*. Cryptology ePrint Archive, Paper 2024/1873. 2024. URL: https://eprint.iacr.org/2024/1873.

[Bé+19] Olivier Bégassat et al. *Handel: Practical Multi-Signature Aggregation for Large Byzantine Committees*. 2019. arXiv: 1906.05132 [cs.DC]. URL: https://arxiv.org/abs/1906.05132.

[Kha+21] Irakliy Khaburzaniya et al. *Aggregating hash-based signatures using STARKs*. Cryptology ePrint Archive, Report 2021/1048. 2021. URL: https://eprint.iacr.org/2021/1048.

[Aar+24] Marius A. Aardal et al. "Aggregating Falcon Signatures with LaBRADOR". In: 2024, pp. 71–106. DOI: 10.1007/978-3-031-68376-3_3.

[Wah+18] Riad S. Wahby et al. "Doubly-Efficient zkSNARKs Without Trusted Setup". In: 2018, pp. 926–943. DOI: 10.1109/SP.2018.00060.

[Ped92] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: 1992, pp. 129–140. DOI: 10.1007/3-540-46766-1_9.

[Abe+16] Masayuki Abe et al. "Structure-Preserving Signatures and Commitments to Group Elements". In: 29.2 (Apr. 2016), pp. 363–421. DOI: 10.1007/s00145-014-9196-7.

[Che+23] Binyi Chen et al. "HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates". In: 2023, pp. 499–530. DOI: 10.1007/978-3-031-30617-4_17.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. "The Algebraic Group Model and its Applications". In: 2018, pp. 33–62. DOI: 10.1007/978-3-319-96881-0_2.

[Bü+19] Benedikt Bünz et al. *Proofs for Inner Pairing Products and Applications*. Cryptology ePrint Archive, Paper 2019/1177. 2019. URL: https://eprint.iacr.org/2019/1177.

[KS23b] Abhiram Kothapalli and Srinath Setty. *HyperNova: Recursive arguments for customizable constraint systems*. Cryptology ePrint Archive, Paper 2023/573. 2023. URL: https://eprint.iacr.org/2023/573.

[KS22b] Abhiram Kothapalli and Srinath Setty. *SuperNova: Proving universal machine executions without universal circuits*. Cryptology ePrint Archive, Paper 2022/1758. 2022. URL: https://eprint.iacr.org/2022/1758.

[Lun+90] Carsten Lund et al. "Algebraic Methods for Interactive Proof Systems". In: 1990, pp. 2–10. DOI: 10.1109/FSCS.1990.89518.

[Ova] *Ova: A slightly better Nova*. https://hackmd.io/V4838nnlRKal9ZiTHiGYzw. Accessed: 2024-11-15.

[Bon+21] Dan Boneh et al. "Halo Infinite: Proof-Carrying Data from Additive Polynomial Commitments". In: 2021, pp. 649–680. DOI: 10.1007/978-3-030-84242-0_23.

[Blu+91] M. Blum et al. "Checking the correctness of memories". In: *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*. 1991, pp. 90–99. DOI: 10.1109/SFCS.1991.185352.

A Deferred definitions

A.1 Signature schemes

Definition 4. *A signature scheme is parametrized by a message space $\mathcal{M}$ and consists on a tuple of PPT algorithms* $(\mathsf{KGen}_{\mathsf{ss}}, \mathsf{Sign}_{\mathsf{ss}}, \mathsf{Verify}_{\mathsf{ss}})$ *such that:*

- $\mathsf{KGen}_{\mathsf{ss}}(n) \to (\mathsf{sk}, \mathsf{pk})$*: On input security parameter n, outputs public parameters* pp*, signing key* pk *and verification key* pk*.*
- $\mathsf{Sign}_{\mathsf{ss}}(\mathsf{sk}, m) \to \sigma_m$*: On input* sk *and $m \in \mathcal{M}$, outputs a signature σ_M on M.*
- $\mathsf{Verify}_{\mathsf{ss}}(\mathsf{pk}, m, \sigma_m) \to 0/1$*: takes as input* pk*, m and signature σ_m and produces a bit expressing acceptance* (1)*, or rejection* (0)*;*

That must satisfy correctness and unforgeability:

Correctness. *For all $m \in \mathcal{M}$ the following holds:*

$$\Pr\left[\mathsf{Verify}_{\mathsf{ss}}(\mathsf{pk}, m, \sigma_m) = 1 \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}_{\mathsf{ss}}(n) \\ \sigma_m \leftarrow \mathsf{Sign}_{\mathsf{ss}}(\mathsf{sk}, m) \end{array} \right] = 1$$

Unforgeability. *For all PPT adversaries $\mathcal{A}$, the following probability is negligible:*

$$\Pr\left[(m, \sigma_m) \notin \mathcal{Q} \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{ss}}(n) \\ (m, \sigma_m) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk}, \mathsf{sk}) \\ \mathsf{Verify}_{\mathsf{ss}}(\mathsf{pk}, m, \sigma_m) = 1 \end{array} \right]$$

where $\mathcal{Q}$ is a list of all the queries $\mathcal{A}$ makes to the signing oracle $\mathcal{O}$.

B Deferred proofs

B.1 Proof of theorem 3

Proof.

Completeness. The commitment C to $f(\vec{X})$ is given by

$$\mathsf{C} = \sum_{\vec{x} \in B_n} \sum_{\vec{y} \in B_m} f(\vec{x}, \vec{y}) \times \mathsf{H}^{(\vec{x},\vec{y})}.$$

Since $nm = k$, this can be computed from the structured reference string (srs). The opening proof π for $(\vec{x}_0, \vec{y}_0)$ is defined as: $\pi = \big(f^*(\vec{Y}), \mathsf{aux} = \{\mathsf{D}^{(\vec{x})}\}_{\vec{x} \in B_n}\big)$, where $f^*(\vec{Y}) = f(\vec{x}_0, \vec{Y})$, and $\mathsf{D}^{(\vec{x})} = \sum_{\vec{y} \in B_m} f(\vec{x}, \vec{y}) \times \mathsf{H}^{(\vec{y})}$.

We see that $\mathsf{Verify}_{\mathsf{KZH}}(\mathsf{C}, (\vec{x}_0, \vec{y}_0), \pi) = 1$:

(I) Check that $e(\mathsf{C}, \mathsf{V}') = \sum_{\vec{x} \in B_n} e(\mathsf{D}^{(\vec{x})}, \mathsf{V}^{(\vec{x})})$:

$$\begin{aligned}
e(\mathsf{C}, \mathsf{V}') &\overset{(1)}{=} e\Big(\sum_{\vec{x} \in B_n} \sum_{\vec{y} \in B_m} f(\vec{x}, \vec{y}) \times \mathsf{H}^{(\vec{x},\vec{y})}, \alpha \times \mathsf{V}\Big) \\
&\overset{(2)}{=} \sum_{\vec{x} \in B_n} e\Big(\sum_{\vec{y} \in B_m} f(\vec{x}, \vec{y}) \cdot \tau^{(\vec{x})} \times \mathsf{G}^{(\vec{y})}, \alpha \times \mathsf{V}\Big) \\
&\overset{(3)}{=} \sum_{\vec{x} \in B_n} e\Big(\sum_{\vec{y} \in B_m} f(\vec{x}, \vec{y}) \cdot \alpha \times \mathsf{G}^{(\vec{y})}, \tau^{(\vec{x})} \times \mathsf{V}\Big) \\
&\overset{(4)}{=} \sum_{\vec{x} \in B_n} e\Big(\sum_{\vec{y} \in B_m} f(\vec{x}, \vec{y}) \times \mathsf{H}^{(\vec{y})}, \mathsf{V}^{(\vec{x})}\Big) \\
&\overset{(5)}{=} \sum_{\vec{x} \in B_n} e(\mathsf{D}^{(\vec{x})}, \mathsf{V}^{(\vec{x})})
\end{aligned}$$

(1) and (2) is exploding terms $\mathsf{C}, \mathsf{H}^{(\vec{x},\vec{y})}$, V' and using the bilinearity property, (3) is using property $e(g^a, g^b) = e(g^b, g^a)$ by interchanging the exponents α and $\tau^{(\vec{x})}$. (4) is replaying $\alpha \times \mathsf{G}^{(\vec{y})}$ with the equivalent value $\mathsf{H}^{(\vec{y})}$ and finally (5) follows from the definition of $\mathsf{D}^{(\vec{x})}$.

(II) Check that $\sum_{\vec{y}\in B_m} f^*(\vec{y}) \times \mathsf{H}^{(\vec{y})} = \sum_{\vec{x}\in B_n} \mathsf{eq}(\vec{x}, \vec{x}_0) \times \mathsf{D}^{(\vec{x})}$.

$$\begin{aligned}
\sum_{\vec{x}\in B_n} \mathsf{eq}(\vec{x}, \vec{x}) \times \mathsf{D}^{(\vec{x})} &\overset{(1)}{=} \sum_{\vec{x}\in B_n} \sum_{\vec{y}\in B_m} \mathsf{eq}(\vec{x}, \vec{x}_0) \cdot f(\vec{x}, \vec{y}) \times \mathsf{H}^{(\vec{y})} \\
&\overset{(2)}{=} \sum_{\vec{y}\in B_m} \left(\sum_{\vec{x}\in B_n} \mathsf{eq}(\vec{x}, \vec{x}_0) \cdot f(\vec{x}, \vec{y}) \right) \times \mathsf{H}^{(\vec{y})} \\
&\overset{(3)}{=} \sum_{\vec{y}\in B_m} f^*(\vec{y}) \times \mathsf{H}^{(\vec{y})}
\end{aligned}$$

Finally, (III) checking that $f^*(\vec{y_0}) = z_0$, follows from the definition of z_0.

Knowledge soundness. Let $\mathcal{A}$ be a PPT adversary that, on input srs outputs a commitment C. We define an extractor $\mathcal{E}$ that outputs $p(\vec{X}, \vec{Y})$ such that, for any $((\vec{x}_0, \vec{y}_0), z_0, \pi)$, if the verifier accepts then $p(\vec{x}_0, \vec{y}_0) = z_0$ with overwhelming probability. Given C and $\{c_{\vec{x},\vec{y}}, \hat{c}_{\vec{y}}\}_{\vec{x}\in B_n, \vec{y}\in B_m}$ such that

$$\mathsf{C} = \sum_{\vec{x}\in B_n} \sum_{\vec{y}\in B_m} c_{\vec{x},\vec{y}} \times \mathsf{H}^{(\vec{x},\vec{y})} + \sum_{\vec{y}\in B_m} \hat{c}_{\vec{y}} \times \mathsf{H}^{(\vec{y})},$$

$\mathcal{E}$ outputs $f(\vec{X}, \vec{Y}) = \sum_{\vec{x}\in B_n} \sum_{\vec{y}\in B_m} c_{\vec{x},\vec{y}} \vec{X}^{\vec{x}} \vec{Y}^{\vec{y}}$. Under the AGM, $\mathcal{E}$ does not abort. Similarly, under the AGM we have that all the $\mathsf{D}^{(\vec{x})}$ are represented as a linear combination of the srs elements, i.e.,

$$\mathsf{D}^{(\vec{x})} = \sum_{\vec{y}\in B_m} d_{\vec{y}}^{(\vec{x})} \times \mathsf{H}^{(\vec{y})} + \sum_{\vec{x}\in B_n} \sum_{\vec{y}\in B_m} \hat{d}_{\vec{x},\vec{y}}^{(\vec{x})} \times \mathsf{H}^{(\vec{x},\vec{y})}$$

The following conditions are satisfied:

(I) $e(\mathsf{C}, \mathsf{V}') = \sum_{\vec{x}\in B_n} e(\mathsf{D}^{(\vec{x})}, \mathsf{V}^{(\vec{x})})$, we first consider each side separately:

$$\begin{aligned}
e(\mathsf{C}, \mathsf{V}') &= e(\sum_{\vec{x}\in B_n} \sum_{\vec{y}\in B_m} c_{\vec{x},\vec{y}} \times \mathsf{H}^{(\vec{x},\vec{y})} + \sum_{\vec{y}\in B_m} \hat{c}_{\vec{y}} \times \mathsf{H}^{(\vec{y})}, \alpha \times \mathsf{V}) \\
&= e(\sum_{\vec{x}\in B_n} \sum_{\vec{y}\in B_m} c_{\vec{x},\vec{y}} \times \mathsf{H}^{(\vec{x},\vec{y})}, \alpha \times \mathsf{V}) + e(\sum_{\vec{y}\in B_m} \hat{c}_{\vec{y}}(\vec{y}) \times \mathsf{H}^{(\vec{y})}, \alpha \times \mathsf{V}) \\
&= \alpha \sum_{\vec{x}\in B_n} \sum_{\vec{y}\in B_m} c_{\vec{x},\vec{y}} \cdot \tau^{(\vec{x})} \times e(\mathsf{G}^{(\vec{y})}, \mathsf{V}) + \alpha^2 \times e(\sum_{\vec{y}\in B_m} \hat{c}_{\vec{y}} \times \mathsf{G}^{(\vec{y})}, \mathsf{V})
\end{aligned}$$

$$\begin{aligned}\sum_{\vec{x}\in B_n} e(\mathsf{D}^{(\vec{x})}, \mathsf{V}^{(\vec{x})}) &= \sum_{\vec{x}\in B_n} e(\sum_{\vec{y}\in B_m} d_{\vec{y}}^{(\vec{x})} \times \mathsf{H}^{(\vec{y})} + \sum_{\vec{x}\in B_n}\sum_{\vec{y}\in B_m} \hat{d}_{\vec{x},\vec{y}}^{(\vec{x})} \times \mathsf{H}^{(\vec{x},\vec{y})}, \tau^{(\vec{x})} \times \mathsf{V}) \\ &= \sum_{\vec{x}\in B_n} e(\sum_{\vec{y}\in B_m} d_{\vec{y}}^{(\vec{x})} \cdot \alpha \times \mathsf{G}^{(\vec{y})} + \sum_{\vec{x}\in B_n}\sum_{\vec{y}\in B_m} \hat{d}_{\vec{x},\vec{y}}^{(\vec{x})} \cdot \tau^{(\vec{x})} \times \mathsf{G}^{(\vec{y})}, \tau^{(\vec{x})} \times \mathsf{V}) \\ &= \sum_{\vec{x}\in B_n} \tau^{(\vec{x})} \cdot \Big(\alpha \cdot \sum_{\vec{y}\in B_m} d_{\vec{y}}^{(\vec{x})} + \sum_{\vec{x}\in B_n}\sum_{\vec{y}\in B_m} \hat{d}_{\vec{x},\vec{y}}^{(\vec{x})} \cdot \tau^{(\vec{x})}\Big) \times e(G^{(\vec{y})}, \mathsf{V})\end{aligned}$$

Now the difference $e(\mathsf{C}, \mathsf{V}') - \sum_{\vec{x}\in B_n} e(\mathsf{D}^{(\vec{x})}, \mathsf{V}^{(\vec{x})})$ can be seen as degree 2 polynomial of α, then the coefficients of 1, α, α^2 on the two terms are equal or we can break quadratic CDH, which implies:

- Coefficient 1.

  $$\sum_{\vec{x}\in B_n}\sum_{\vec{y}\in B_m} \hat{d}_{\vec{x},\vec{y}}^{(\vec{x})} \cdot \tau^{(\vec{x})} \times e(G^{(\vec{y})}, \mathsf{V}) = 0 \implies \sum_{\vec{x},\vec{y}} \hat{d}_{\vec{x},\vec{y}}^{(\vec{x})} \times \mathsf{H}^{(\vec{x},\vec{y})} = 0$$

  This implies $\hat{d}_{\vec{x},\vec{y}}^{(\vec{x})} = 0$ for all $\vec{x}, \vec{y}$, or we found a polynomial that vanishes at $\mathsf{H}^{(\vec{x},\vec{y})}$ and thus can calculate τ and the discrete log relation between the generators $\mathsf{G}^{(\vec{y})}$ by finding the roots of it, which breaks the dlog assumption.

- Coefficient α.

  $$\sum_{\vec{x}\in B_n}\sum_{\vec{y}\in B_m} \tau^{(\vec{x})} \cdot c_{\vec{x},\vec{y}} \times e(G^{(\vec{y})}, \mathsf{V}) = \sum_{\vec{x}\in B_n} \tau^{(\vec{x})} \cdot \sum_{\vec{y}\in B_m} d_{\vec{y}}^{(\vec{x})} \times e(G^{(\vec{y})}, \mathsf{V})$$

  Then we have that $c_{\vec{x},\vec{y}} = d_{\vec{y}}^{(\vec{x})}$ for all $\vec{x}, \vec{y}$ or we can break the dlog assumption as above.

- Coefficient α^2.

  $$e(\sum_{\vec{y}\in B_m} \hat{c}_{\vec{y}} \times \mathsf{G}^{(\vec{y})}, \mathsf{V}) = 0 \implies \sum_{\vec{y}\in B_m} \hat{c}_{\vec{y}} \times \mathsf{G}^{(\vec{y})} = 0$$

We then have:

$$\mathsf{C} = \sum_{\vec{x}\in B_n}\sum_{\vec{y}\in B_m} c_{\vec{x},\vec{y}} \times \mathsf{H}^{(\vec{x},\vec{y})}, \quad \mathsf{D}^{(\vec{x})} = \sum_{\vec{y}\in B_m} d_{\vec{y}}^{(\vec{x})} \times \mathsf{H}^{(\vec{y})}$$

(II) $\sum_{\vec{y}\in B_m} f^*(\vec{y}) \times \mathsf{H}^{(\vec{y})} = \sum_{\vec{x}\in B_n} \mathsf{eq}(\vec{x}, \vec{x}_0) \times \mathsf{D}^{(\vec{x})}$, we have:

$$\begin{aligned}\sum_{\vec{y}\in B_m} f^*(\vec{y}) \times \mathsf{H}^{(\vec{y})} &= \sum_{\vec{x}\in B_n} \mathsf{eq}(\vec{x}, \vec{x_0}) \times \mathsf{D}^{(\vec{x})} \\ &= \sum_{\vec{x}\in B_n} \mathsf{eq}(\vec{x}, \vec{x_0}) \times \sum_{\vec{y}\in B_m} c_{\vec{x},\vec{y}} \times \mathsf{H}^{(\vec{y})} \\ &= \sum_{\vec{y}\in B_m} c_{\vec{x_0},\vec{y}} \times \mathsf{H}^{(\vec{y})}\end{aligned}$$

$$\implies \sum_{\vec{y}\in B_m} f^*(\vec{y}) \times \mathsf{H}^{(\vec{y})} = \sum_{\vec{y}\in B_m} c_{\vec{x_0},\vec{y}} \times \mathsf{H}^{(\vec{y})}$$

Now, $f^*(\vec{y}) = \sum_{\vec{y}\in B_m} c_{\vec{x_0},\vec{y}} \mathsf{H}^{(\vec{x}_0,\vec{y})} = f(\vec{x}_0, \vec{Y})$ or we can break the dlog assumption by finding the roots of $P(Y) = \sum_{\vec{y}\in B_m} (f^*(\vec{y}) - c_{\vec{x_0},\vec{y}}) \times (Y \times \mathsf{G}^{(\vec{y})})$ and give trapdoor α as output. Finally, the extractor outputs polynomial $f(\vec{x}, \vec{y})$. Since $f^*(\vec{y}_0) = z_0$, and C commits to $f(\vec{x}, \vec{y})$, we have the construction is knowledge sound.

□

B.2 Proof of theorem 4

Proof.

Completeness. Consider the following two satisfying accumulators:

- $\mathsf{acc}.x_1 = \{\mathsf{C}_1, T_1, \mathsf{E}_1, \vec{x}_1, \vec{y}_1, z_1\}$
- $\mathsf{acc}.w_1 = \{[\mathsf{D}_1^{(\vec{i})}]_{\vec{i}\in B_n}, \vec{f_1^*}, \mathcal{T}_x{}^{(1)}, \mathcal{T}_y{}^{(1)}\}$
- $\mathsf{acc}.x_2 = \{\mathsf{C}_2, T_2, \mathsf{E}_2, \vec{x}_2, \vec{y}_2, z_2\}$
- $\mathsf{acc}.w_2 = \{[\mathsf{D}_2^{(\vec{i})}]_{\vec{i}\in B_n}, \vec{f_2^*}, \mathcal{T}_x{}^{(2)}, \mathcal{T}_y{}^{(2)}\}$

As they are valid accumulators, the following hold:

- $\sum_{i=0}^{n} e(\mathsf{D}_1^{(i)}, \mathsf{V}_i) = e(\mathsf{C}_1, \mathsf{V})$
  - $\langle (\mathcal{T}_x^{(1)} || \mathcal{T}_y^{(1)}), \vec{\mathsf{K}} \rangle = T_1$
  - $E_1 = \mathsf{Dec}(\vec{x_1}, \vec{y_1}, z_1, \vec{f_1^*}, \mathcal{T}_x^{(1)}, \mathcal{T}_y^{(1)}, \vec{\mathsf{D}_1})$
- $\sum_{i=0}^{n} e(\mathsf{D}_2^{(i)}, \mathsf{V}_i) = e(\mathsf{C}_2, \mathsf{V})$
  - $\langle (\mathcal{T}_x^{(2)} || \mathcal{T}_y^{(2)}), \vec{\mathsf{K}} \rangle = T_2$
  - $E_2 = \mathsf{Dec}(\vec{x_2}, \vec{y_2}, z_2, \vec{f_2^*}, \mathcal{T}_x^{(2)}, \mathcal{T}_y^{(2)}, \vec{\mathsf{D}_2})$

Now let $\mathsf{acc}.x, \mathsf{acc}.w$ be the accumulated instance/witness computed as follows.

$$\mathsf{acc}.x \leftarrow ((1-\beta) \cdot [\mathsf{C}_1, T_1, \vec{x_1}, \vec{y_1}, z_1] + \beta \cdot [\mathsf{C}_2, T_2, , \vec{x}_2, \vec{y}_2, z_2]),\ \mathsf{E}$$
$$\mathsf{acc}.w \leftarrow (1-\beta) \cdot [\vec{\mathsf{D}_1}, \vec{f_1^*}, \mathcal{T}_x^{(1)}, \mathcal{T}_y^{(1)}] + \beta \cdot [\vec{\mathsf{D}_2}, \vec{f_2^*}, \mathcal{T}_x^{(2)}, \mathcal{T}_y^{(2)}],\ \mathsf{E}$$

where $\mathsf{E} \leftarrow (1-\beta) \times \mathsf{E}_1 + \beta \times \mathsf{E}_2 + (1-\beta)\beta \times \mathsf{Q}$. Below, we show acc is also a satisfying accumulator:

- First condition

$$\begin{aligned}\sum_{i=0}^{n} e(\mathsf{D}^{(i)}, \mathsf{V}_i) &\overset{(1)}{=} \sum_{i=0}^{n} e\Big((1-\beta)\times \mathsf{D}_1^{(i)} + \beta \times \mathsf{D}_2^{(i)}, \mathsf{V}_i\Big)\\ &\overset{(2)}{=} (1-\beta)\times \sum_{i=0}^{n} e(\mathsf{D}_1^{(i)}, \mathsf{V}_i) + \beta \times \sum_{i=0}^{n} e(\mathsf{D}_2^{(i)}, \mathsf{V}_i)\\ &\overset{(3)}{=} (1-\beta)\times e(\mathsf{C}_1, \mathsf{V}) + \beta \times e(\mathsf{C}_2, \mathsf{V})\\ &\overset{(4)}{=} e\big((1-\beta)\times \mathsf{C}_1 + \beta \times \mathsf{C}_2, \mathsf{V}\big)\\ &\overset{(5)}{=} e(\mathsf{C}, \mathsf{V})\end{aligned}$$

  Where (1) holds by expanding $\mathsf{D}^{(i)}$ terms, (2), (4) because of bilinearity property, (3) holds because the first condition was satisfied for the underlying instances, and finally (5) holds because of the definition of C.

- Second condition.

$$\begin{aligned}\langle(\mathcal{T}_x||\mathcal{T}_y), \vec{\mathsf{K}}\rangle &\overset{(1)}{=} \Big\langle(1-\beta)\cdot(\mathcal{T}_x^{(1)}||\mathcal{T}_y^{(1)}) + \beta\cdot(\mathcal{T}_x^{(2)}||\mathcal{T}_y^{(2)}), \vec{\mathsf{K}}\Big\rangle\\ &\overset{(2)}{=} (1-\beta)\cdot\Big\langle\mathcal{T}_x^{(1)}||\mathcal{T}_y^{(1)}, \vec{\mathsf{K}}\Big\rangle + \beta\cdot\Big\langle\mathcal{T}_x^{(2)}||\mathcal{T}_y^{(2)}, \vec{\mathsf{K}}\Big\rangle\\ &\overset{(3)}{=} (1-\beta)\times T_2 + \beta \times T_2\\ &\overset{(4)}{=} T\end{aligned}$$

  (1) holds by definition, whereas (2) is implied by bilinearity of the inner product. (3) and (4) follow by definition.

- Third condition.

$$\begin{aligned}\mathsf{Dec}(\vec{x}, \vec{y}, z, \vec{f}^*, \mathcal{T}_x, \mathcal{T}_y, \vec{\mathsf{D}}) &\overset{(1)}{=} \mathsf{Dec}((1-\beta)\cdot[\vec{x_1}, \vec{y_1}, z_1, \vec{f_1^*}, \mathcal{T}_x^{(1)}, \mathcal{T}_y^{(1)}, \vec{\mathsf{D}_1}]\\ &\qquad + \beta\cdot[\vec{x_2}, \vec{y_2}, z_2, \vec{f_2^*}, \mathcal{T}_x^{(2)}, \mathcal{T}_y^{(2)}, \vec{\mathsf{D}_2}])\\ &\overset{(2)}{=} (1-\beta)\times \mathsf{E}_1 + \beta \times \mathsf{E}_2 + (1-\beta)\cdot\beta\times \mathsf{Q}\\ &\overset{(3)}{=} \mathsf{E}\end{aligned}$$

by definition of E.

Knowledge soundness. Consider an adversary $\mathcal{A}$ that outputs $\hat{\pi} = (\pi.x, \pi.w)$, $\hat{\mathsf{acc}} = (\mathsf{acc}.x, \mathsf{acc}.w), \hat{\mathsf{pf}} \in \mathbb{G}_1$ and $\mathsf{acc}_1.x, \mathsf{acc}_2.x$. We build an extractor $\mathsf{Ext}_{\mathsf{acc}}$ that if $\mathsf{V}_{\mathsf{acc}}$ and $\mathsf{D}_{\mathsf{acc}}$ accept, extracts valid witnesses $\mathsf{acc}_1.w, \mathsf{acc}_2.w$ for $\mathsf{acc}_1.x, \mathsf{acc}_2.x$. Since $\mathsf{acc}.w$ is an equation of degree one, given two accepting transcripts for different challenges β_1, β_2, $\mathsf{Ext}_{\mathsf{acc}}$ can use the Vandermonde matrix to extract

$$\mathsf{acc}_1.w = (\{\mathsf{D}_1^{(\vec{x})}\}_{\vec{x}\in B_n}, \vec{f}_1^*, \mathcal{T}_1^{(x)}, \mathcal{T}_1^{(y)}),\ \mathsf{acc}_2.w = (\{\mathsf{D}_2^{(\vec{x})}\}_{\vec{x}\in B_n}, \vec{f}_2^*, \mathcal{T}_2^{(x)}, \mathcal{T}_2^{(y)}).$$

We now prove that the extracted witnesses are valid. First, note that since the verifier accepts, $\mathsf{E} = (1-\beta) \times \mathsf{E}_1 + \beta \times \mathsf{E}_2 + (1-\beta)\cdot\beta \times \mathsf{Q}$ and

$$(\mathsf{C}, T, \vec{x}, \vec{y}, z) = (1-\beta)\cdot(\mathsf{C}_1, T_1, \vec{x}_1, \vec{y}_1, z_1) + \beta\cdot(\mathsf{C}_2, T_2, \vec{x}_2, \vec{y}_2, z_2)$$

The left side of Eq.(i) of the decider is:

$$\begin{aligned}\sum_{\vec{x}\in B_n} e(\mathsf{D}^{(\vec{x})}, \mathsf{V}^{(\vec{x})}) &= \sum_{\vec{x}\in B_n} e\Big((1-\beta)\times \mathsf{D}_1^{(\vec{x})} + \beta \times \mathsf{D}_2^{(\vec{x})}, \mathsf{V}^{(\vec{x})}\Big) \\ &= (1-\beta)\times \sum_{\vec{x}\in B_n} e(\mathsf{D}_1^{(\vec{x})}, \mathsf{V}^{(\vec{x})}) + \beta \times \sum_{\vec{x}\in B_n} e(\mathsf{D}_2^{(\vec{x})}, \mathsf{V}^{(\vec{x})})\end{aligned}$$

Whereas the right side equals

$$e(\mathsf{C}, \mathsf{V}) = e\big((1-\beta)\times \mathsf{C}_1 + \beta \times \mathsf{C}_2, \mathsf{V}\big) = (1-\beta)\times e(\mathsf{C}_1, \mathsf{V}) + \beta \times e(\mathsf{C}_2, \mathsf{V})$$

Since β is computed as a hash of $\mathsf{acc}_1.x$ and $\mathsf{acc}_2.x$, except with negligible probability $\sum_{\vec{x}\in B_n} e(\mathsf{D}_1^{(\vec{x})}, \mathsf{V}^{(\vec{x})}) = e(\mathsf{C}_1, \mathsf{V})$ and $\sum_{\vec{x}\in B_n} e(\mathsf{D}_2^{(\vec{x})}, \mathsf{V}^{(\vec{x})}) = e(\mathsf{C}_2, \mathsf{V})$. Similarly, Eq.$(ii)$ holds if and only if

$$\begin{aligned}\big\langle\big((1-\beta)\mathcal{T}_1^{(x)} + \beta\mathcal{T}_2^{(x)} \,\big\|\, (1-\beta)\mathcal{T}_1^{(y)} + \beta\mathcal{T}_2^{(y)}\big), \vec{\mathsf{K}}\big\rangle& \\ = (1-\beta)\big\langle\big(\mathcal{T}_1^{(x)} \,\big\|\, \mathcal{T}_1^{(y)}\big), \vec{\mathsf{K}}\big\rangle + \beta\big\langle\big(\mathcal{T}_2^{(x)} \,\big\|\, \mathcal{T}_2^{(y)}\big), \vec{\mathsf{K}}\big\rangle& \\ = (1-\beta)T_1 + \beta T_2& \\ = T.&\end{aligned}$$

Thus the equation holds for $\mathsf{acc}_1.w$ and $\mathsf{acc}_2.w$. Finally, replacing the accumulation witness by the extracted ones, we have that Eq.(iii) is

$$\begin{aligned}\mathsf{Dec}((1-\beta)\vec{x}_1 \quad &+\beta\vec{x}_2, (1-\beta)\vec{y}_1 + \beta\vec{y}_2, (1-\beta)z_1 + \beta z_2, (1-\beta)\vec{f}_1^* + \beta\vec{f}_2^*, \\ &(1-\beta)\mathcal{T}_{x,1} + \beta\mathcal{T}_{x,2}, (1-\beta)\mathcal{T}_{y,1} + \beta\mathcal{T}_{y,2}, (1-\beta)\vec{D}_1 + \beta\vec{D}_2) \\ &= (1-\beta)\times \mathsf{E}_1 + \beta\times\mathsf{E}_2 + (1-\beta)\beta\times\mathsf{Q}\end{aligned}$$

The left side of the equation equals

$$\begin{aligned}&\langle(1-\beta)\mathcal{T}_{x_1}^{(\text{error})} + \beta\mathcal{T}_{x_2}^{(\text{error})} || (1-\beta)\mathcal{T}_{y_1}^{(\text{error})} + \beta\mathcal{T}_{y_2}^{(\text{error})} || \\ &\qquad \langle(1-\beta)\vec{f}_1^* + \beta\vec{f}_2^*,\ (1-\beta)\mathcal{T}^{(y_1)}.\text{leaves} + \beta\mathcal{T}^{(y_2)}.\text{leaves}\rangle \\ &\quad -((1-\beta)z_1 + \beta z_2), \vec{\mathsf{K}} || \mathsf{K}'\rangle \\ &+\langle(1-\beta)\vec{f}_1^* + \beta\vec{f}_2^*, (\mathsf{H}^{(\vec{y})})_{\vec{y}\in B_n}\rangle - \langle(1-\beta)\mathcal{T}^{(x_1)}.\text{leaves} + \beta\mathcal{T}^{(x_2)}.\text{leaves}, (1-\beta)\vec{\mathsf{D}}_1 + \beta\vec{\mathsf{D}}_2\rangle\end{aligned}$$

Term β contains equation (iii) of the Decider for acc_1, whereas term $\beta(1-\beta)$ contains the one for acc_2. Cross terms are in the coefficient of $\beta(1-\beta)$. Then, except with negligible probability, $\mathsf{D}_{\mathsf{acc}}(\mathsf{acc}_1) = \mathsf{D}_{\mathsf{acc}}(\mathsf{acc}_2) = 1$ and the extractor succeeds. We now prove that if $\mathsf{E}_1 = 0$, we can extract a valid opening to $\vec{C}_1$. That is, $(\mathsf{acc}_1.x, \mathsf{acc}_1.w)$ is a valid pair for the predicate Φ. From term $(1-\beta)$ in the equation above we have that, except with negligible probability, $\mathsf{Dec}(\vec{x}_1, \vec{y}_1, z_1, \vec{f}_1^*, \mathcal{T}^{(x_1)i.}, \mathcal{T}^{(x_2)}, \vec{\mathsf{D}}) = 0$. That is,

$$0 = \langle \mathcal{T}_x^{(\text{error})} || \mathcal{T}_y^{(\text{error})} || \langle \vec{f}_1^*, \mathcal{T}^{(y)}.\text{leaves} \rangle - z_1, \vec{\mathsf{K}} || \mathsf{K}' \rangle + \langle \vec{f}_1^*, (\mathsf{H}^{(\vec{y})})_{\vec{y} \in B_n} \rangle - \langle \mathcal{T}^{(x)}.\text{leaves}, \vec{\mathsf{D}}_1 \rangle$$

Claim: $\vec{\mathsf{D}}_1$ is base $(\mathsf{H}^{(\vec{y})})_{\vec{y} \in B_n}$ We first show that in the algebraic group model, we can write $\vec{\mathsf{D}}_1$ base $\mathsf{H}^{(\vec{y})}$. Consider the following check:

$$\sum_{\vec{x} \in B_n} e(\mathsf{D}_1^{(\vec{x})}, \mathsf{V}^{(\vec{x})}) = e(\mathsf{C}_1, \mathsf{V}')$$

Note that, $\mathsf{V}^{(\vec{x})}$ contains a factor of $\tau^{(\vec{x})}$. The only elements that contain this factor in $\mathbb{G}_1$ are the $\mathsf{H}^{((\vec{i}),(\vec{j}))}$ generators. Thus, C_1 can only be written base $\mathsf{H}^{(\vec{i},\vec{j})}$. Otherwise, we can break the discrete logarithm assumption. Conversely, this implies that each $\mathsf{D}_1^{(\vec{x})}$ must be written base $\mathsf{H}^{(\vec{i})}$, i.e. $\mathsf{D}_1^{(\vec{x})} = \langle \vec{f}^{(x)}, (\mathsf{H}^{(\vec{i})})_{\vec{i} \in B_n} \rangle$. This proves the claim. Given this, we can write

$$\begin{aligned} \langle \mathcal{T}^{(x)}.\text{leaves}, \vec{\mathsf{D}}_1 \rangle &= \textstyle\sum_{\vec{x} \in B_n} \mathcal{T}^{(x)}.\text{leaves} \cdot \langle \vec{f}^{(x)}, (\mathsf{H}^{(\vec{i})})_{\vec{i} \in B_n} \rangle \\ &= \langle \textstyle\sum_{\vec{x} \in B_n} \mathcal{T}^{(x)}.\text{leaves} \cdot \vec{f}^{(x)}, (\mathsf{H}^{(\vec{i})})_{\vec{i} \in B_n} \rangle \end{aligned}$$

$$0 = \langle \mathcal{T}_x^{(\text{error})} || \mathcal{T}_y^{(\text{error})} || \langle \vec{f}_1^*, \mathcal{T}^{(y)}.\text{leaves} \rangle - z_1, \vec{\mathsf{K}} || \mathsf{K}' \rangle + \langle \vec{f}_1^* - \textstyle\sum_{\vec{x} \in B_n} \mathcal{T}^{(x)}.\text{leaves} \cdot \vec{f}^{(x)}, (\mathsf{H}^{(\vec{i})})_{\vec{i} \in B_n} \rangle$$

We can split this equation into $\langle \mathcal{T}_x^{(\text{error})} || \mathcal{T}_y^{(\text{error})} || \langle \vec{f}_1^*, \mathcal{T}^{(y)}.\text{leaves} \rangle - z_1, \vec{\mathsf{K}} || \mathsf{K}' \rangle = 0$ and $\langle \vec{f}_1^* - \sum_{\vec{x} \in B_n} \mathcal{T}^{(x)}.\text{leaves} \cdot \vec{f}^{(x)}, (\mathsf{H}^{(\vec{i})})_{\vec{i} \in B_n} \rangle = 0$ or we can break the dlog assumption as the left side of the product is a polynomial that vanishes at the logarithms of the group elements in $\vec{K}$. Then, we have that that Eq.(3) in Fig. 2 is satisfied since $f^*(\vec{y}_1) = \langle \vec{f}_1^*, \mathcal{T}^{(y_1)}.\text{leaves} \rangle = z_1$. Also, $0 = \langle \vec{f}_1^*, (\mathsf{H}^{(\vec{y})})_{\vec{y} \in B_n} \rangle = \langle \mathcal{T}^{(x)}.\text{leaves}, \vec{\mathsf{D}}_1 \rangle$, representing Eq. (2) in Fig. 2. Finally, from above we have that Eq.(1) is also satisfied. Then, it is enough for $\mathsf{Ext}_{\mathsf{acc}}$ to run $\mathsf{Ext}_{\mathsf{KZH}}$. □

C Higher dimension PCS for smaller deciders

We extend KZH-2 to a higher dimensional polynomial commitment which has a more efficient verifier and smaller proof size. This results in a smaller accumulator and more efficient decider. In particular, we generalize the 2-dimensional KZH-2, which can be viewed as a vector-matrix-vector product to higher dimensional tensor products. This reduces the PC verifier from $\sqrt{n}$ to $n^{1/k}$. Note that in order to build efficient accumulation for a multilinear PCS we need to expand a *short* evaluation point $\vec{x}$ into *long* vectors, which correspond to $\mathsf{eq}(\vec{x}, \vec{b})$ for $\vec{b} \in H \subset \mathbb{F}$. This transformation is equivalent to the one in KZH-2, and we omit it here for ease of presentation.

C.1 KZH-k

Notation. We denote the space of n dimensional tensors with dimensions $(d_1, d_2, \ldots, d_n)$ on field $\mathbb{F}$ with $\mathbb{F}^{d_1 \times d_2 \times \cdots \times d_n}$. For $\mathbf{T} \in \mathbb{F}^{d_1 \times d_2 \times \cdots \times d_k}$ and $\vec{x}_1 \in \mathbb{F}^{d_1}$, we denote tensor inner product as

$$\langle \mathbf{T}, \vec{x}_1 \rangle = \langle \mathbf{T}_1, \vec{x}_1 \rangle \otimes \mathbf{T}_2 \otimes \ldots \otimes \mathbf{T}_k.$$

Note that the result is a tensor in $\mathbb{F}^{d_2 \times \cdots \times d_k}$. Similarly for $\vec{x}_1 \in \mathbb{F}^{d_1}$ and $\vec{x}_2 \in \mathbb{F}^{d_2}$, $\langle \mathbf{T}, \vec{x}_1 \otimes \vec{x}_2 \rangle = \langle \langle \mathbf{T}, \vec{x}_1 \rangle, \vec{x}_2 \rangle \in \mathbb{F}^{d_3 \times \cdots \times d_k}$.

Protocol description. We construct a commitment scheme for such tensors, where $\mathbf{T}$ represents the polynomial that can be opened at evaluation point $(\vec{x}_1, \ldots, \vec{x}_k)$ for $\vec{x}_j \in \mathbb{F}^{d_j}$ as:

$$\langle \langle \langle \langle \mathbf{T}, \vec{x}_1 \rangle, \vec{x}_2 \rangle, \ldots, \vec{x}_{k-1} \rangle, \vec{x}_k \rangle = y \in \mathbb{F}.$$

The scheme admits an efficient accumulation scheme, and is general enough to support multilinear and univariate polynomial commitments. Compared to two-dimensional KZH, the tradeoff is that the accumulator instance and verifier are of size k, but the accumulation witness is of size $n^{\frac{1}{k}}$. The core insight is that we commit to $\mathsf{C} = \langle \mathbf{T}, \vec{\mu}_1 \otimes \vec{\mu}_2 \cdots \otimes \vec{\mu}_k \rangle \times \mathsf{G}$, for secret elements $\vec{\mu}_1, \ldots, \vec{\mu}_k$ and group generator G, and open $[\mathsf{C}_i]_{i=1}^k$, commitments to all $k-1$ dimensional slices of $\mathbf{T}$. The verifier can check that these slices are correct using a pairing with $\vec{\mu}_1$ and compute $\mathsf{C}' = \langle (\mathsf{C}_1, \ldots, \mathsf{C}_k), \vec{x}_1, \rangle$. C' is now a commitment to the $k-1$ dimensional tensor $\mathbf{T}_1 = \langle \mathbf{T}, \vec{x}_1 \rangle$ and we can recursively apply the scheme. We present the full protocol for degree $[d]^k := (d, d, \ldots, d)$ tensor in Figure 11.

Efficiency. The commitment size is a single $\mathbb{G}_1$ element. The commitment time is dominated by an MSM of size n for tensors of size n. Part of the opening can be preprocessed, as with KZH-2, reducing the opening time to the tensor product plus $O(n^{1/2})$ group operations. Concretely, the prover will compute commitments to $f(\vec{X}, \vec{b})$ for all $\vec{b} \in \{0,1\}^{\log(n)/2}$. Using these the first $\log(n)/2$ steps of the opening proof can be computed in time $O(\sqrt{n})$. The second half of the opening proof can also be computed efficiently using $f(\vec{\alpha}, \vec{X})$ for the

partial evaluation point $\vec{\alpha} \in \mathbb{F}^{\log(n)/2}$. The proof size is $(k-1) \cdot n^{1/k}$ $\mathbb{G}_1$ elements, as well as $n^{1/k}$ field elements. The verification time is $O(k \cdot n^{1/k})$ and dominated by $k-1$ pairing products of size $n^{1/k}$.

$\mathsf{KZH}^{(k)}.\mathsf{Setup}(\lambda, d, k)$**:**

- Sample $\mathsf{G} \leftarrow\!\!\$\ \mathbb{G}_1, \mathsf{V} \leftarrow\!\!\$\ \mathbb{G}_2$ and $[[\mu_{i,j}]_{i=1}^{d}]_{j=1}^{k} \leftarrow\!\!\$\ \mathbb{F}$
- $\mathbf{H}_1 = \{\mathsf{H}_{i_1,\ldots,i_k} \leftarrow (\prod_{j=1}^{k} \mu_{i_s,j}) \times \mathsf{G} : \forall i_1, \ldots, i_k \in [d]\}$
- $\mathbf{H}_2 = \{\mathsf{H}_{i_2,\ldots,i_k} \leftarrow (\prod_{j=2}^{k} \mu_{i_j,j}) \times \mathsf{G} : \forall i_2, \ldots, i_k \in [d]\}$
- $\ldots$
- $\mathbf{H}_k = \{\mathsf{H}_{i_k} \leftarrow \mu_{i_k,k} \times \mathsf{G} : \forall i_k \in [d]\}$
- $\mathbf{V} = \{\mathsf{V}_{i,j} \leftarrow \mu_{i,j} \times \mathsf{V} : \forall i \in [d],\ j \in [k]\}$
- Output $(\mathsf{G}, \mathsf{V}, \mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_k, \mathbf{V})$

$\mathsf{KZH}^{(k)}.\mathsf{Commit}(\mathsf{srs}, \mathbf{T})$**:** For $\mathbf{T} \in \mathbb{F}^{d \times d \times \cdots \times d}$, compute the commitment as it follows:

- Output $\mathsf{C} \leftarrow \langle \mathbf{T}, \mathbf{H}_1 \rangle = \sum_{(i_1,i_2,\ldots,i_k) \in [d]^k} \mathbf{T}_{i_1,i_2,\ldots,i_k} \times \mathsf{H}_{i_1,i_2,\ldots,i_k}$

$\mathsf{KZH}^{(k)}.\mathsf{Open}(\mathsf{srs}, \mathsf{C}, \mathbf{T}, \vec{x}_1, \ldots, \vec{x}_k)$**:** Given commitment $\mathsf{C} \in \mathbb{G}_1$, tensor $\mathbf{T} \in \mathbb{F}^{d \times d \times \cdots \times d}$ and inputs $\vec{x}_j \in \mathbb{F}^d$, compute opening as it follows:

- Let $\mathbf{T}_1 = \mathbf{T}$
- For $j = 1, \ldots, k-1$:
  - Set $\mathbf{T}_{j+1} \leftarrow \langle \mathbf{T}_j, \vec{x_j} \rangle$
  - Compute vector $\vec{\mathsf{D}}_j$ e.g. $\mathsf{D}_j[i] \leftarrow \langle \langle i \rangle \otimes \mathbf{T}_j, \mathbf{H}_{j+1} \rangle$ for all $i \in [d]$, where $\langle i \rangle$ represents decomposition of i into d bits.
- Output $\pi \leftarrow \{[\vec{\mathsf{D}}_j]_{j \in [k-1]}, \mathbf{T}_k\}$

$\mathsf{KZH}^{(k)}.\mathsf{Verify}(\mathsf{srs}, \mathsf{C}, [\vec{x}_j]_{j \in [k]}, y, \pi)$**:** Given commitment $\mathsf{C} \in \mathbb{G}_1$, inputs $\vec{x}_j \in \mathbb{F}^d$, output $y \in \mathbb{F}$ and opening proof π, does as it follows:

- Parse $\{[\vec{\mathsf{D}}_j]_{j \in [k-1]}, \mathbf{T}_k\} \leftarrow \pi$
- Set $\mathsf{C}_0 = \mathsf{C}$
- For $j \in 1, \ldots, k-1$
  1. Check that $e(\mathsf{C}_{j-1}, \mathsf{V}) = \sum_{i=0}^{d} e(\mathsf{D}_j[i], \mathsf{V}_{i,j})$
  2. Compute $\mathsf{C}_j \leftarrow \langle \vec{x}_j, \vec{\mathsf{D}}_j \rangle$
- Check that $\mathsf{C}_{k-1} = \langle \mathbf{T}_k, \mathbf{H}_k \rangle$
- Check that $\langle \mathbf{T}_k, \vec{x}_k \rangle = y$

Figure 11: KZH-k description

Theorem 5. *The protocol in Figure 11 is a complete and knowledge-sound polynomial commitment scheme in the AGM under the* dlog *assumption.*

Proof.

Completeness. Consider honestly generated $\mathsf{C}, [\mathsf{C}_j]_{j=1}^k$. For the first check, note that $e(\mathsf{C}_0, \mathsf{V}) = e(\langle \mathbf{T}, \mathbf{H}_1 \rangle, \mathsf{V})$ by construction of C. On the other hand,

$$\begin{aligned}\sum_{i=0}^{d} e(\mathsf{D}_1[i], \mathsf{V}_{i,1}) &= \sum_{i=0}^{d} e(\langle \langle i \rangle \otimes \mathbf{T}_1, \mathbf{H}_2 \rangle, \mu_{i,1} \times \mathsf{V}) = \sum_{i=0}^{d} e(\langle \langle i \rangle \otimes \mathbf{T}_1, \mu_{i,1} \times \mathbf{H}_2 \rangle, \mathsf{V}) \\ &= e(\langle \mathbf{T}_1, \mathbf{H}_1 \rangle, \mathsf{V})\end{aligned}$$

and since $\mathbf{T}_1 = \mathbf{T}$, the verifier accepts. For the general case,

$$\begin{aligned}e(\mathsf{C}_{j-1}, \mathsf{V}) &= e(\langle \vec{\mathsf{D}}_{j-1}, \vec{x}_{j-1} \rangle, \mathsf{V}) = e(\sum_{i=1}^{d} \vec{\mathsf{D}}_{j-1}[i] \vec{x}_{j-1}[i], \mathsf{V}) \\ &= e(\sum_{i=1}^{d} \langle \langle i \rangle \otimes \mathbf{T}_{j-1},\, \mathbf{H}_j \rangle \vec{x}_{j-1}[i], \mathsf{V}) = e(\sum_{i=1}^{d} \langle \langle \vec{x}_{j-1}, \mathbf{T}_{j-1} \rangle,\, \mathbf{H}_j \rangle, \mathsf{V}) \\ &= e(\langle \mathbf{T}_j,\, \mathbf{H}_j \rangle, \mathsf{V}) = \sum_{i=1}^{d} e(\langle \mathbf{T}_j,\, \mathbf{H}_{j+1} \rangle, \mathsf{V}_{i,j}) \\ &= \sum_{i=1}^{d} e(\mathsf{D}_j[i], \mathsf{V}_{i,j}).\end{aligned}$$

In the second verification equation we have:

$$\begin{aligned}\mathsf{C}_{k-1} &= \langle \vec{x}_{k-1}, \vec{\mathsf{D}}_{k-1} \rangle = \sum_{i=1}^{d} \vec{\mathsf{D}}_{k-1}[i] \vec{x}_{k-1}[i] \\ &= \sum_{i=1}^{d} \langle \langle i \rangle \otimes \mathbf{T}_{k-1},\, \mathbf{H}_k \rangle \vec{x}_{k-1}[i] = \sum_{i=1}^{d} \langle \langle \mathbf{T}_{k-1}, \vec{x}_{k-1} \rangle,\, \mathbf{H}_k \rangle \\ &= \langle \mathbf{T}_k, \mathbf{H}_k \rangle.\end{aligned}$$

And since the third check follows directly, we have that the verifier accepts.

Knowledge soundness. Let $\mathcal{A}$ be a PPT adversary that on input srs outputs a commitment C. We define an extractor $\mathcal{E}$ that outputs $p(\vec{X}_1, \ldots, \vec{X}_k)$ such that, for any tuple $((\vec{x}_1, \ldots, \vec{x}_k), y, \pi)$, accepted by the verifier, $p(\vec{x}_1, \ldots, \vec{x}_k) = y$ with overwhelming probability. Under the AGM, we assume $\mathcal{A}$ is algebraic and thus $\mathcal{A}$ outputs C along with $\{\vec{c}^r\}_{r=1}^k$ such that:

$$\mathsf{C} = \sum_{r=1}^{k} \langle \vec{c}^r, \mathbf{H}_r \rangle$$

$\mathcal{E}$ outputs $p(\vec{X}_1, \ldots, \vec{X}_k) = (\vec{c}_1, \ldots, \vec{c}_k) \otimes (\vec{X}_1, \ldots, \vec{X}_k)$. Under the AGM, $\mathcal{E}$ does not abort. Similarly, under the AGM we have that there exist $[\vec{d}^r_{i,j}]_{r=1}^k$ such that for all i, j:

$$\mathsf{D}_j[i] = \sum_{r=1}^{k} \langle \vec{d}^r_{i,j}, \mathbf{H}_r \rangle$$

Because the verifier accepts, we have that all their checks are satisfied. In particular, for C and all $[\mathsf{D}_1[i]]_{i=0}^d$:

$$e(\mathsf{C}, \mathsf{V}) = \sum_{i=1}^{d} e(\mathsf{D}_1[i], \mathsf{V}_{i,1})$$

Replacing by the extracted $\mathsf{C}, \mathsf{D}_1[i]$ and the form of $\mathsf{V}_{i,1}$, we have

$$e\left(\sum_{r=1}^{k} \langle \vec{c}^r, \mathbf{H}_r \rangle, \mathsf{V}\right) = \sum_{i=1}^{d} e\left(\sum_{r=1}^{k} \langle \vec{d}^r_{i,1}, \mathbf{H}_r \rangle, \mu_{i,1} \times \mathsf{V}\right)$$

Then,

(1) It must be the case that $\vec{c}^r = 0$ for all $r \neq 1$ or we can calculate the discrete logarithm relation between $\mathbf{H}_2, \ldots, \mathbf{H}_k$ and $[\mathsf{V}_{i,1}]_{i=1}^d$, breaking the dlog assumption. Similarly, we have $\vec{d}^r_{i,1} = 0$ for all $r > 2$ or we can find the discrete log relation between $\mathbf{H}_2$ and $\mathbf{H}_3, \ldots, \mathbf{H}_k$.

(2) Also, $\vec{d}^1_{i,1} = 0$ or we can extract $(\mu_{i,1}^2 \prod_{j \neq 1} \mu_{i,j}) \times \mathsf{G}$, breaking CDH.

(3) Finally, we have that each $\mathsf{D}_1[i]$ is base $[\mathsf{H}_{ii_3 \ldots i_k}]_{i_3, \ldots, i_k \in [d]}$, i.e., $\mathsf{D}_1[i] = \langle \vec{d}^2_{i,1}, \mu_{i,2} \times \mathbf{H}_3 \rangle$, or we can find the discrete log relation between $\mu_{i,1} \times \mu_{s,2} \times \mathbf{H}_3$ and $\mathbf{H}_1$ for $s \neq i$. This implies $(\vec{d}^2_{i,1})_s = 0$ for all $s \neq i$.

The equation then is

$$e\left(\langle \vec{c}^1, \mathbf{H}_1 \rangle, \mathsf{V}\right) = \sum_{i=1}^{d} e\left(\langle \vec{d}^2_{i,1}, \mathbf{H}_{2,i} \rangle, \mu_{i,1} \times \mathsf{V}\right)$$

which implies $\vec{c}_1 = \sum_{i=1}^{d} \vec{d}^2_{i,1}$. Indeed, if there exists $s \in [k]$ such that $c_s \neq d^2_{s,1}$, $\mu_{s,1}$ is a root of the polynomial $c_s X - d^2_{s,1} X$ and we can find it, breaking dlog. In the general case, for every $j = 2, \ldots, k$ we have:

$$e\left(\mathsf{C}_j, \mathsf{V}\right) = \sum_{i=0}^{d} e\left(\mathsf{D}_{j+1}[i], \mu_{i,j+1} \times \mathsf{V}\right)$$

for $\mathsf{C}_j = \langle \vec{x}_j, \vec{\mathsf{D}}^j \rangle$. Then, if $\mathsf{D}^j = \langle \vec{d^j}, \mathbf{H}_j \rangle$, it must be the case that $\mathsf{D}^{j+1} = \langle \vec{d^j}^{+1}, \mathbf{H}_{j+1} \rangle$, with $\vec{d^j}^{+1}_s = 0$ fro all $s \neq j+1$ or we break dlog as in item (1) and (3), and CDH as in (2) above. By induction, we have $\mathsf{D}^j = \langle \vec{d^j}, \mathbf{H}_j \rangle$ for all $j = 1, \ldots, k.$, Finally, we have

$$e\left(\langle \vec{x}_j, \vec{\mathsf{D}}^j \rangle, \mathsf{V}\right) = \sum_{i=0}^{d} e\left(\langle \vec{d}_i^{j+1}, \mathbf{H}_{j+1} \rangle, \mu_{i,j+1} \times \mathsf{V}\right)$$

And thus

$$\langle \vec{x}_j, \langle \vec{d^j}, \mathbf{H}_j \rangle \rangle = \langle \sum_{i=0}^{d} \vec{d}_i^{j+1}, \mathbf{H}_{j+1} \rangle$$

So $\sum_{i=0}^{d} \vec{d}_i^{j+1} = \vec{x}_j \otimes \vec{d^j}$, This implies $\mathbf{T}_k = \vec{x}_{k-1} \otimes \ldots \otimes \vec{x}_1 \otimes \vec{c}$ and thus $y = \vec{x}_k \otimes \mathbf{T}_k$ implies $y = p(\vec{x}_1, \ldots, \vec{x}_k)$ for the polynomial $p(\vec{X}_1, \ldots, \vec{X}_k)$ encoded in C and we conclude the proof.

□

C.2 KZH-k accumulation

We now construct an accumulation scheme for KZH-k. In particular, we focus on the case where the tensor $\mathbf{T}$ is a multilinear polynomial. At a high level, the KZH-k verifier is still low degree and algebraic, and therefore, we can apply the same accumulation strategy as in the two-dimensional case. Importantly, the accumulator size and the decider time are reduced to $O(k \cdot n^{\frac{1}{k}})$. The accumulation verifier performs $O(k)$ $\mathbb{G}_1$ operations. For a $k \cdot d$-linear polynomial, the accumulator instance and witness can be described as it follows: (red terms only appear in accumulators not proofs)

$$\mathsf{acc}.x = \{\mathsf{C}, \mathsf{C}_1, \ldots, \mathsf{C}_{k-1} \in \mathbb{G}, \vec{x}_1, \ldots, \vec{x}_k \in \mathbb{F}^d, y \in \mathbb{F}, E_{\mathbb{G}} \in \mathbb{G}, e_{\mathbb{F}} \in \mathbb{F}\}$$

$$\mathsf{acc}.w = \{[\mathsf{D}_{i,j}]_{j \in [k-1], i \in [0,d]}, \mathbf{T}_k \in \mathbb{F}^d\}$$

The accumulation prover, verifier and decider of KZH-k are respectively defined in Figures 12 and 13.

Efficiency. The accumulator consists of the PCS proof and is of size $O(k \cdot n^{1/k})$. The accumulation decider runs the PCS verifier and is dominated by $k-1$ $n^{1/k}$ pairing products and the accumulation verifier is dominated by $k+1$ $\mathbb{G}_1$ operations.

$\mathsf{P}_{\mathsf{acc}}(\mathsf{srs}, (\mathsf{acc}.x, \mathsf{acc}.w), (\mathsf{acc}'.x, \mathsf{acc}'.w))$**:**

- Parse $\mathsf{acc}.x, \mathsf{acc}.w$ and $\mathsf{acc}'.x$, $\mathsf{acc}'.w$ as below:
  - $\{\mathsf{C}, \mathsf{C}_1, \ldots, \mathsf{C}_{k-1}, (\vec{x}_1, \ldots, \vec{x}_k, y), (E_{\mathbb{G}}, e_{\mathbb{F}})\} \leftarrow \mathsf{acc}.x$
  - $\{\{[\mathsf{D}_{i,j}]_{j\in[k-1],\, i\in[0,d]}, \mathbf{T}_k\} \leftarrow \mathsf{acc}.w$
  - $\{\mathsf{C}', \mathsf{C}'_1, \ldots, \mathsf{C}'_{k-1}, (\vec{x}'_1, \ldots, \vec{x}'_k, y'), (E'_{\mathbb{G}}, e'_{\mathbb{F}})\} \leftarrow \mathsf{acc}'.x$
  - $\{\{[\mathsf{D}'_{i,j}]_{j\in[k-1],\, i\in[0,d]}, \mathbf{T}'_k\} \leftarrow \mathsf{acc}'.w$
- Let $\mathsf{Dec}_{\mathbb{G}}$ and $\mathsf{Dec}_{\mathbb{F}}$ be the verification checks as defined in Figure **??**. Compute $Q \in \mathbb{G}$ such that:

$$\begin{aligned}&\mathsf{Dec}_{\mathbb{G}}\big((1-X)\cdot(\vec{x}_j, \mathsf{C}_j, [\mathsf{D}_{i,j}]_{i=0}^{d}) + X\cdot(\vec{x}'_j, \mathsf{C}'_j, [\mathsf{D}'_{i,j}]_{i=0}^{d_j})\big)\\ &\qquad = (1-X)\times E_j + X\times E'_j + (1-X)\cdot X\times Q.\end{aligned}$$

  and $q \in \mathbb{F}$ such that

$$\begin{aligned}&\mathsf{Dec}_{\mathbb{F}}\big((1-X)\cdot(\mathbf{T}_k, \vec{x}_k, y) + X\cdot(\mathbf{T}'_k, \vec{x}'_k, y')\big)\\ &\qquad = (1-X)\cdot e_{\mathbb{F}} + X\cdot e'_{\mathbb{F}} + (1-X)\cdot X\cdot q.\end{aligned}$$

- Derive challenge $\alpha \leftarrow H(\mathsf{acc}.x, \mathsf{acc}'.x, Q, q)$ through Fiat-Shamir.
- Compute
  - $\mathsf{C}'' \leftarrow (1-\alpha)\times \mathsf{C} + \alpha \times \mathsf{C}'$
  - $\vec{x}_i{}'' \leftarrow (1-\alpha)\cdot \vec{x}_i + \alpha \cdot \vec{x}_i{}'$ for $i \in [k]$
  - $y'' \leftarrow (1-\alpha)\cdot y + \alpha \cdot y'$
  - $\mathsf{C}''_j \leftarrow (1-\alpha)\times \mathsf{C}_j + \alpha \times \mathsf{C}'_j$ for $j \in [1, k-1]$
  - $\mathsf{D}''_{i,j} \leftarrow (1-\alpha)\times \mathsf{D}_{i,j} + \alpha \times \mathsf{D}'_{i,j}$ for each j, i for $i \in [k]$ and $j \in [k-1]$
  - $E''_{\mathbb{G}} \leftarrow (1-\alpha)\times E_{\mathbb{G}} + \alpha \times E'_{\mathbb{G}} + (1-\alpha)\cdot\alpha\times Q$
  - $e''_{\mathbb{F}} \leftarrow (1-\alpha)\cdot e_{\mathbb{F}} + \alpha\cdot e'_{\mathbb{F}} + (1-\alpha)\cdot\alpha\cdot q$
  - $\mathbf{T}''_k \leftarrow (1-\alpha)\times \mathbf{T}_k + \alpha \times \mathbf{T}'_k$
- Output the new accumulator $\mathsf{acc}'' = (\mathsf{acc}''.x, \mathsf{acc}''.w)$ and $\mathsf{pf} = \{Q, q\}$ as the accumulation proof.

$$\mathsf{acc}''.x = \{\mathsf{C}'', [\mathsf{C}''_j]_{j\in[k-1]}, [\vec{x}''_i]_{i\in[k]}, y'', [E''_{\mathbb{G}}, e_{\mathbb{F}}\}$$

$$\mathsf{acc}''.w = \{\{[\mathsf{D}''_{i,j}]_{j\in[k-1],\, i\in[0,d_j]}, \mathbf{T}''_k\}$$

Figure 12: KZH-k accumulation prover

$\mathsf{V}_{\mathsf{acc}}(\mathsf{srs}, \mathsf{acc}.x, \mathsf{acc}'.x, \mathsf{pf} = \{Q \in \mathbb{G}, q \in \mathbb{F}\})$**:**

- Input $\mathsf{acc}.x, \mathsf{acc}'.x$ and $\mathsf{pf} = \{Q, q\}$.
- Regenerate challenge $\alpha \leftarrow H(A.X, A'.X, \mathsf{pf})$
- Compute
  - $\mathsf{C}'' \leftarrow (1-\alpha) \times \mathsf{C} + \alpha \times \mathsf{C}'$
  - $\vec{x_i}'' \leftarrow (1-\alpha) \cdot \vec{x_i} + \alpha \cdot \vec{x_i}'$ for $i \in [k]$
  - $y'' \leftarrow (1-\alpha) \cdot y + \alpha \cdot y'$
  - $\mathsf{C}''_j \leftarrow (1-\alpha) \times \mathsf{C}_j + \alpha \times \mathsf{C}'_j$ for $j \in [1, k-1]$
  - $E''_{\mathbb{G}} \leftarrow (1-\alpha) \times E_{\mathbb{G}} + \alpha \times E'_{\mathbb{G}} + (1-\alpha) \cdot \alpha \times Q$
  - $e''_{\mathbb{F}} \leftarrow (1-\alpha) \cdot e_{\mathbb{F}} + \alpha \cdot e'_{\mathbb{F}} + (1-\alpha) \cdot \alpha \times q$
- Output the new accumulator instance $\mathsf{acc}''.x$.

$$\mathsf{acc}''.x = \{\mathsf{C}'', [\mathsf{C}''_j]_{j \in [k-1]}, [\vec{x}''_i]_{i \in [k]}, y'', E''_{\mathbb{G}}, e''_{\mathbb{F}}\}$$

$\mathsf{D}_{\mathsf{acc}}(\mathsf{srs}, \mathsf{acc}.x, \mathsf{acc}.w)$**:**

- Parse instance and witness as it follows:
  - $\{\mathsf{C}, (\mathsf{C}_1, \ldots, \mathsf{C}_{k-1}), (\vec{x}_1, \ldots, \vec{x}_k, y), E_{\mathbb{G}}, e_{\mathbb{F}}\} \leftarrow \mathsf{acc}.x$
  - $\{[\vec{\mathsf{D}_j} = [\mathsf{D}_{i,j}]_{i \in [0,d]}]_{j=1}^{k-1}, \mathbf{T}_k\} \leftarrow \mathsf{acc}.w$
- Set $\mathsf{C}_0 \leftarrow \mathsf{C}$
- For each $j \in [1, k-1]$ check that

$$e(\mathsf{C}_{j-1}, \mathsf{V}) - \sum_{i=0}^{d} e(\mathsf{D}_{i,j}, \mathsf{V}_{i,j}) = 0$$

- Check that $\mathsf{Dec}_{\mathbb{G}}([\vec{x}_i]_{i=1}^{k} \mathsf{C}, [\mathsf{C}_i]_{i=1}^{k-1}, [\vec{\mathsf{D}_j} \in \mathbb{G}^d]_{j=1}^{k-1}) = \sum_{j=1}^{k-1}(\mathsf{C}_j - \langle \vec{x}_j, \vec{\mathsf{D}_j} \rangle) = E_{\mathbb{G}}$
- Check that $\langle \mathbf{T}_k, \mathbf{H}_k \rangle = \mathsf{C}_{k-1}$
- Check that $\mathsf{Dec}_{\mathbb{F}}(\vec{x}_k, \mathbf{T}_k, y) = \langle \mathbf{T}_k, \vec{x}_k \rangle - y = e_{\mathbb{F}}$

Figure 13: KZH-k accumulation verifier and decider

Theorem 6. KZH-*k-fold is a secure accumulation scheme, also under the* dlog-*assumption in the algebraic group model.*

Proof.

Completeness. Consider the following two satisfying accumulator instances:

- $\mathsf{acc}.x = \{\mathsf{C}, \mathsf{C}_1, \ldots, \mathsf{C}_{k-1}, (\vec{x}_1, \ldots, \vec{x}_k, y), (E_{\mathbb{G}}, e_{\mathbb{F}})\}$

- $\mathsf{acc}.w = \{\{[\mathsf{D}_{i,j}]_{j\in[k-1],\, i\in[0,d]},\, \mathbf{T}_k\}$
- $\mathsf{acc}'.x = \{\mathsf{C}', \mathsf{C}'_1, \ldots, \mathsf{C}'_{k-1},\, (\vec{x}'_1, \ldots, \vec{x}'_k, y'),\, (E'_{\mathbb{G}}, e'_{\mathbb{F}})\}$
- $\mathsf{acc}'.w = \{\{[\mathsf{D}'_{i,j}]_{j\in[k-1],\, i\in[0,d]}, \mathbf{T}'_k\}$

It is straightforward to see that honest $\mathsf{P}_{\mathsf{acc}}$ and $\mathsf{V}_{\mathsf{acc}}$ output the same $\mathsf{acc}''.x$ as they follow the same instructions. Then it is left to see that on input $(\mathsf{acc}''.x, \mathsf{acc}''.w)$ computed by an honest prover, $\mathsf{D}_{\mathsf{acc}}$ always accepts. For each $j \in [1, k-1]$:

$$\begin{aligned} e(\mathsf{C}''_{j-1}, \mathsf{V}) &= e\left((1-\alpha)\times \mathsf{C}_{j-1} + \alpha \times \mathsf{C}'_{j-1}, \mathsf{V}\right) = (1-\alpha)\times e\left(\mathsf{C}_{j-1}, \mathsf{V}\right) + \alpha \times e\left(\mathsf{C}'_{j-1}, \mathsf{V}\right) \\ &= (1-\alpha)\times \textstyle\sum_{i=0}^{d} e\left(\mathsf{D}_{i,j}, \mathsf{V}_{i,j}\right) + \alpha \times \sum_{i=0}^{d} e\left(\mathsf{D}'_{i,j}, \mathsf{V}_{i,j}\right) \\ &= \sum_{i=0}^{d} e\left((1-\alpha)\times \mathsf{D}_{i,j} + \alpha \times \mathsf{D}'_{i,j}, \mathsf{V}_{i,j}\right) \\ &= \sum_{i=0}^{d} e\left(\mathsf{D}''_{i,j}, \mathsf{V}_{i,j}\right) \end{aligned}$$

so the first equation holds. For the second check, we have:

$$\begin{aligned} &\mathsf{Dec}_{\mathbb{G}}([\vec{x}_i]_{i=1}^{k}, \mathsf{C}'', [\mathsf{C}''_i]_{i=1}^{k-1}, [\vec{\mathsf{D}''_j}]_{j=1}^{k-1}) \\ &\quad = \mathsf{Dec}_{\mathbb{G}}([\vec{x}_i]_{i=1}^{k}, (1-\alpha)\times \mathsf{C} + \alpha \times \mathsf{C}', [(1-\alpha)\times \mathsf{C}_i + \alpha \times \mathsf{C}'_i]_{i=1}^{k-1}, [(1-\alpha)\times \vec{\mathsf{D}}_j + \alpha \times \vec{\mathsf{D}}'_j]_{j=1}^{k-1}) \\ &\quad = (1-\alpha)\times \mathsf{E}_{\mathbb{G}} + \alpha \times \mathsf{E}'_{\mathbb{G}} + (1-\alpha)\alpha \times \mathsf{Q} = \mathsf{E}''_{\mathbb{G}} \end{aligned}$$

by construction of Q. The third equation verifies as

$$\begin{aligned} \langle \mathbf{T}''_k, \mathbf{H}_k\rangle &= \langle (1-\alpha)\times \mathbf{T}_k + \alpha \times \mathbf{T}'_k, \mathbf{H}_k\rangle = (1-\alpha)\times \langle \mathbf{T}_k, \mathbf{H}_k\rangle + \alpha \times \langle \mathbf{T}'_k, \mathbf{H}_k\rangle \\ &= (1-\alpha)\times \mathsf{C}_{k-1} + \alpha \times \mathsf{C}'_{k-1} = \mathsf{C}''_{k-1}. \end{aligned}$$

Finally, by definition of $e''_{\mathbb{F}}$

$$\begin{aligned} \mathsf{Dec}_{\mathbb{F}}(\vec{x}''_k, \mathbf{T}''_k, y'') &= \langle \mathbf{T}''_k, \vec{x}''_k\rangle - y'' \\ &= \langle (1-\alpha)e_{\mathbb{F}} + \alpha e'_{\mathbb{F}} + (1-\alpha) + (1-\alpha)\alpha q \\ &= e''_{\mathbb{F}}, \end{aligned}$$

and $\mathsf{D}_{\mathsf{acc}}$ outputs 1, concluding the proof of completeness.

Knowledge soundness. Consider an adversary $\mathcal{A}$ that outputs $\hat{\pi} = (\pi.x, \pi.w)$, $\hat{\mathsf{acc}}'' = (\mathsf{acc}''.x, \mathsf{acc}''.w), \hat{\mathsf{pf}} \in \mathbb{G}_1 \times \mathbb{F}$ and $\mathsf{acc}.x, \mathsf{acc}'.x$. We build an extractor $\mathsf{Ext}_{\mathsf{acc}}$ such that if $\mathsf{V}_{\mathsf{acc}}$ and $\mathsf{D}_{\mathsf{acc}}$ accept, extracts valid witnesses $\mathsf{acc}.w, \mathsf{acc}'.w$ for $\mathsf{acc}.x, \mathsf{acc}'.x$. Since $\mathsf{acc}''.w$ is an equation of degree one, given two accepting transcripts for different challenges α_1, α_2, $\mathsf{Ext}_{\mathsf{acc}}$ can use the Vandermonde matrix to extract

$$\mathsf{acc}.w = \{\{[\mathsf{D}_{i,j}]_{j\in[k-1],\, i\in[0,d]},\, \mathbf{T}_k\}, \quad \mathsf{acc}'.w = \{\{[\mathsf{D}'_{i,j}]_{j\in[k-1],\, i\in[0,d]}, \mathbf{T}'_k\}.$$

Since $\mathsf{D}_{\mathsf{acc}}(\mathsf{acc}''.x, \mathsf{acc}''.w)$ accepts, we have the first check passes, i.e., $e(\mathsf{C}''_{j-1}, \mathsf{V}) = \sum_{i=0}^{d} e\left(\mathsf{D}''_{i,j}, \mathsf{V}_{i,j}\right)$. We analyze each side of the equation independently:

$$\begin{aligned} e(\mathsf{C}''_{j-1}, \mathsf{V}) &= e\left((1-\alpha) \times \mathsf{C}_{j-1} + \alpha \times \mathsf{C}'_{j-1}, \mathsf{V}\right) = (1-\alpha) \times e\left(\mathsf{C}_{j-1}, \mathsf{V}\right) + \alpha \times e\left(\mathsf{C}'_{j-1}, \mathsf{V}\right) \\ \sum_{i=0}^{d} e\left(\mathsf{D}''_{i,j}, \mathsf{V}_{i,j}\right) &= \sum_{i=0}^{d} e\left((1-\alpha) \times \mathsf{D}_{i,j} + \alpha \times \mathsf{D}'_{i,j}, \mathsf{V}_{i,j}\right) \\ &= (1-\alpha) \times \sum_{i=0}^{d} e\left(\mathsf{D}_{i,j}, \mathsf{V}_{i,j}\right) + \alpha \times \sum_{i=0}^{d} e\left(\mathsf{D}'_{i,j}, \mathsf{V}'_{i,j}\right) \end{aligned}$$

Since α is computed as a hash of $\mathsf{acc}.x$ and $\mathsf{acc}'.x$, except with negligible probability $e\left(\mathsf{C}_{j-1}, \mathsf{V}\right) - \sum_{i=0}^{d} e\left(\mathsf{D}_{i,j}, \mathsf{V}_{i,j}\right) = 0$ and $e\left(\mathsf{C}'_{j-1}, \mathsf{V}\right) - \sum_{i=0}^{d} e\left(\mathsf{D}'_{i,j}, \mathsf{V}'_{i,j}\right) = 0$. Similarly, replacing the accumulation witness by the extracted ones, we have that Eq.(ii)'s left side is

$$\sum_{j=1}^{k-1}(\mathsf{C}''_j - \langle \vec{x}''_j, \vec{\mathsf{D}}''_j \rangle) = \sum_{j=1}^{k-1}((1-\alpha)\mathsf{C}_j + \alpha\mathsf{C}'_k - \langle (1-\alpha)\vec{x}_j + \alpha\vec{x}'_j, (1-\alpha)\vec{\mathsf{D}}_j + \alpha\vec{\mathsf{D}}'_j \rangle)$$

whereas from the verifier's output we have that the right side equals $(1-\alpha) \times \mathsf{E}_{\mathbb{G}} + \alpha \times \mathsf{E}'_{\mathbb{G}} + (1-\alpha) \cdot \alpha \times \mathsf{Q}$. Because α is computed as a hash of $\mathsf{acc}.x$ and $\mathsf{acc}'.x$ we have that except with negligible probability the equation hold for any X and, in particular, when $X = 1$ we got $\sum_{j=1}^{k-1} \mathsf{C}_j - \langle \vec{x}_j, \vec{\mathsf{D}}_j \rangle = \mathsf{E}_{\mathbb{G}}$ and for $X = 0$, $\sum_{j=1}^{k-1} \mathsf{C}'_j - \langle \vec{x}'_j, \vec{\mathsf{D}}'_j \rangle = \mathsf{E}'_{\mathbb{G}}$. With identical reasoning, we have that $\langle \mathbf{T}_k, \vec{x}_k \rangle - y = e_{\mathbb{F}}$ and $\langle \mathbf{T}'_k, \vec{x}'_k \rangle - y' = e'_{\mathbb{F}}$. Finally,

$$\langle \mathbf{T}''_k, \mathbf{H}_k \rangle = \langle (1-\alpha)\mathbf{T}_k + \alpha\mathbf{T}_k, \mathbf{H}_k \rangle = (1-\alpha)\langle \mathbf{T}_k, \mathbf{H}_k \rangle + \alpha\langle \mathbf{T}'_k, \mathbf{H}_k \rangle$$
$$\mathsf{C}''_{k-1} = (1-\alpha)\mathsf{C}_{k-1} + \alpha\mathsf{C}'_{k-1}$$

which, as above, implies $\mathsf{C}_{k-1} = \langle \mathbf{T}_k, \mathbf{H}_k \rangle$, and $\mathsf{C}'_{k-1} = \langle \mathbf{T}'_k, \mathbf{H}'_k \rangle$ except with negligible probability. We conclude the extracted $\mathsf{acc}.w$ and $\mathsf{acc}'.w$ are valid witnesses. We now prove that if $\mathsf{E}_{\mathbb{G}} = e_{\mathbb{F}} = 0$, we can extract a valid opening to $\vec{c}$. That is, $(\mathsf{acc}.x, \mathsf{acc}.w)$ is a valid pair for the predicate Φ. Note that the first check by the decider is the same as $\mathsf{KZH}'s$ verifier first check. Following above, from the second check we can extract that $\sum_{j=1}^{k-1} \mathsf{C}_j - \langle \vec{x}_j, \mathsf{D}_j \rangle = \mathsf{E}_{\mathbb{G}} = 0$. Since D_j is base $\mathbf{H}_j$, we have that $\mathsf{C}_j - \langle \vec{x}_j, \mathsf{D}_j \rangle = 0$ for all $j \in [k]$. Finally, the last check is $\langle \mathbf{T}_k, \vec{x}_k \rangle = y$, and we have extracted satisfying KZH verifier checks, and thus can extract $p(\vec{X}_1, \ldots, \vec{X}_k)$ such that $p(\vec{x}_1, \ldots, \vec{x}_k) = y$.

□

D Simple accumulatable R1CS PIOP

We use the Spartan NIZK to reduce R1CS to an accumulation scheme. Let $A, B, C \in \mathbb{F}^{n \times m}$ be the R1CS matrices and subsequently define $\mu_n = \log_2(n)$ and $\mu_m = \log_2(m)$. Further, let $\tilde{A}(X, Y) \in \mathbb{F}[X_1, \ldots, X_{\mu_n}, Y_1, \ldots, Y_{\mu_m}]$ be the multi-linear extension of A and similarly define $\tilde{B}, \tilde{C}$ for B, C respectively. Then, given a random challenge $\vec{r} \in \mathbb{F}^{\mu_n}$, the prover and verifier run a sumcheck protocol for the following equation:

$$\sum_{b_x \in \{0,1\}^{\mu_n}} \mathsf{eq}(\vec{r}, b_x) \cdot f(b_x) = 0$$

for $f(b_x) := \tilde{A}z(b_x) \cdot \tilde{B}z(b_x) - \tilde{C}z(b_x)$, in which $\tilde{A}z(b_x) = \sum_{b_y \in \{0,1\}^{\mu_m}} \tilde{A}(b_x, b_y) \cdot \tilde{z}(b_y)$. Spartan [Set20] gives an efficient protocol for this sumcheck. To accumulate it, we need to accumulate the evaluations of $\tilde{A}, \tilde{B}, \tilde{C}, z$. For the latter, which includes the witness, we can use the KZH-fold protocol. For $\tilde{A}$, note that we need to accumulate the evaluations of the same multilinear polynomial at multiple evaluation points, i.e. $\tilde{A}(r_x^{(1)}, r_y^{(1)}) = z^{(1)}$ and $\tilde{A}(r_x^{(2)}, r_y^{(2)}) = z^{(2)}$. This can be done easily using a simple random linear combination. We present an accumulation scheme in Figure 14 for relation $\mathcal{R}_{\tilde{A}}$, i.e. $\tilde{A} \in \mathsf{MLP}(\mathbb{F}, \mu_n + \mu_m)$ as defined below:

$$\mathcal{R}_{\tilde{A}} = \{(r_x \in \mathbb{F}^{\mu_n}, r_y \in \mathbb{F}^{\mu_m}, z \in \mathbb{F}) \ : \ \tilde{A}(r_x, r_y) = z\}$$

$\mathsf{P}_{\mathbf{ABC}}(\tilde{A}, (r_x^{(1)}, r_y^{(1)}, z^{(1)}), (r_x^{(2)}, r_y^{(2)}, z^{(2)}))$**:**

- Given two satisfying instances $(r_x^{(1)}, r_y^{(2)}, z^{(1)})$, $(r_x^{(2)}, r_y^{(2)}, z^{(2)}) \in \mathcal{R}_{\tilde{A}}$
- Prover computes $q(X)$ in the following polynomial identity where $q(x)$ is a polynomial of degree $\mu + \nu - 2$.

$$\begin{aligned}\tilde{A}((1-X)\cdot r_x + X \cdot r'_x, (1-X)\cdot r_y + X \cdot r'_y) \\ = (1-X)\cdot z + X \cdot z' + (1-X)\cdot X \cdot q(X)\end{aligned}$$

  Note $q(X)$ can be computed directly through polynomial interpolation by evaluating the identity above with different X values.

- Derive challenge

$$\alpha \leftarrow H((r_x^{(1)}, r_y^{(2)}, z^{(1)}), (r_x^{(2)}, r_y^{(2)}, z^{(2)}), q(X))$$

- Compute the accumulated instances as follows:

$$\begin{aligned}r_x \leftarrow (1-\alpha)\cdot r_x^{(1)} + \alpha \cdot r_x^{(2)} \quad r_y \leftarrow (1-\alpha)\cdot r_y^{(1)} + \alpha \cdot r_y^{(2)} \\ z \leftarrow (1-\alpha)\cdot z^{(1)} + \alpha \cdot z^{(2)} + \alpha \cdot (1-\alpha)\cdot q(\alpha)\end{aligned}$$

- Output accumulated instance (r_x, r_y, z) along with accumulation proof $q(x)$

$\mathsf{V}_{\mathbf{ABC}}((r_x^{(1)}, r_y^{(1)}, z^{(1)}), (r_x^{(2)}, r_y^{(2)}, z^{(2)}), q(X))$**:**

- Derive challenge $\alpha \leftarrow H((r_x^{(1)}, r_y^{(2)}, z^{(1)}), (r_x^{(2)}, r_y^{(2)}, z^{(2)}), q(X))$
- Compute the accumulated instances as it follows:

$$\begin{aligned}r_x \leftarrow (1-\alpha)\cdot r_x^{(1)} + \alpha \cdot r_x^{(2)} \quad r_y \leftarrow (1-\alpha)\cdot r_y^{(1)} + \alpha \cdot r_y^{(2)} \\ z \leftarrow (1-\alpha)\cdot z^{(1)} + \alpha \cdot z^{(2)} + \alpha \cdot (1-\alpha)\cdot q(\alpha)\end{aligned}$$

- Output accumulated instance (r_x, r_y, z)

$\mathsf{D}_{\mathbf{ABC}}(\tilde{A}, (r_x, r_y, z))$**:** Compute $z' \leftarrow \tilde{A}(r_x, r_y)$ and assert $z \stackrel{?}{=} z'$

Figure 14: Matrix evaluation accumulation description

E Non-Uniform IVC

Background. Non-uniform IVC [KS22b] extends the definition of IVC by allowing each step to execute one of several predefined instructions $F_1, F_2, \ldots, F_k$ instead of a single instruction F. Previously, non-uniform IVC was implemented using a universal circuit that contains subcircuits for all instructions F_i. This circuit evaluates all subcircuits and selects the correct output based on the program counter. However, this approach is inefficient because the prover must perform computations for every instruction, even though only one instruction is needed at each step and the witness size of the universal circuit scales linearly with the sum of the witness size of all instructions, making it non-ideal both computation-wise and memory-wise

Previous work. SuperNova [KS22b] introduced a more efficient method where the step circuit maintains a running accumulator U_i (a relaxed committed R1CS instance) for each instruction F_i. When receiving a new fresh accumulator instance u_i, the prover uses memory techniques (e.g. a Merkle tree or offline memory techniques [Blu+91]) to select the appropriate running accumulator U_i. Next, the prover accumulates U_i with u_i. This approach ensures the computational effort corresponds only to the selected instruction, but the witness size grows linearly with the sum of the witness sizes for all instructions F_i.

Protostar [BC23] offers a similar improvement, leveraging the fact that committing to zeros with Pederson commitment incurs no additional cost. While it also reduces computational overhead, like SuperNova, it still requires the prover to manage a witness size that scales linearly with the sum of all instruction witnesses. Another drawback, Protostar's approach unlike SuperNova is dependent on Pederson being homomorphic and may not be compatible with hash-based polynomial commitment schemes.

Our approach. We observe polynomial accumulation offers more flexibility than circuit-specific accumulation. For example, each polynomial of degree $d < D$, can be seen as a polynomial of degree D by simply assuming the coefficients of $x^{d+1}, x^{d+2}, \ldots, x^D$ are zero. As a result, given an accumulation scheme for a polynomial of degree D, different polynomials of degree $d_i < D$ can be accumulated by considering them as degree D. Supernova directly translates each circuit as an R1CS instance and since two different R1CS instances cannot be accumulated, the prover reqquires to keep one running accumulator for each different instruction F_i. However, similar to our IVC approach in Section 4.2, we leverage Spartan PIOP to translate each instruction F_i as polynomials. Recalling Appendix D, to accumulate circuit F_i, we need to accumulate polynomial $\omega_i(\cdot)$ corresponding to the R1CS witness and matrix evaluations of A_i, B_i and C_i, corresponding to the R1CS construction of F_i. Similar to Section 4.2, we take two different strategies to handle the accumulation of witness polynomial $\omega_i(\cdot)$ and matrices evaluations. For each F_i, assume that its witness polynomial is of degree d_i. We consider a running polynomial of degree $d_i < D$, and in each

step accumulate $\omega_i(\cdot)$ with this running PCS accumulator. However, the same strategy cannot be applied to matrix evaluations. Accumulating different A_i, B_i, and C_i evaluations via PCS accumulation is impractical because the resulting accumulated matrices may not be sparse. This would lead to matrices with $O(kn)$ non-zero elements instead of $O(n)$. To address this, we adopt an approach similar to SuperNova's to handle matrix evaluations efficiently.

Let us revisit the matrix evaluation accumulator ($\mathsf{V}_{\mathsf{ABC}}$), displayed in Figure 5. In uniform IVC, the circuit keeps three running matrix evaluation accumulator instances $\{(r_x^{(M)}, r_y^{(M)}), z^{(M)}\}$ for $M \in \{A, B, C\}$, i.e. $\tilde{M}(r_x^{(M)}, r_y^{(M)}) = z^{(M)}$. And accumulates them with the next incoming instances, as described in Figure 14. The matrix evaluation accumulator circuit is relatively efficient: given an input polynomial $q(x)$ (logarithmic in size relative to the original R1CS), the circuit hashes $q(x)$ to derive randomness r, evaluates $q(r)$, and performs a series of random combinations. Consequently, the witness size of the matrix evaluation accumulator circuit is $O(|r_x^{(M)}| + |r_y^{(M)}|)$, which means it is logarithmic in the size of the original instruction F. To accumulate matrix evaluations corresponding to different instructions F_i, we take a similar strategy to SuperNova. The prover keeps k accumulators $\{(r_x^{(M,i)}, r_y^{(M,i)}), z^{(M,i)}\}$ for $1 \leq i \leq k$, each corresponding to instruction F_i. Using memory techniques, at each step, the circuit selects the correct accumulator instances and performs the accumulation as described in Figure 14. Intuitively, we have reduced the prover's job, from storing a running accumulator corresponding to F_i, to only storing matrix inputs and evaluations, which is exponentially smaller in size. As a result, we reduce the witness size of the circuit to $\max_i |F_i| + \sum_i \log |F_i|$, compared to $\sum_i |F_i|$ in previous approaches.