Dynamic zk-SNARKs (with applications to sparse zk-SNARKs and IVC)

Weijie Wang[1], Charalampos Papamanthou[1,2], Shravan Srinivasan[2], and Dimitrios Papadopoulos[3,2]

[1] Yale University
[2] Lagrange Labs
[3] Hong Kong University of Science and Technology

Abstract. In this work, we introduce *dynamic zk-SNARKs*. A dynamic zk-SNARK extends a standard zk-SNARK with an additional *update* algorithm. This algorithm takes as input a valid source statement–witness pair $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ together with a verifying proof π, and a valid target statement–witness pair $(\mathbb{x}', \mathbb{w}') \in \mathcal{R}$. It outputs a verifying proof π' for $(\mathbb{x}', \mathbb{w}')$ in *sublinear* time (when $(\mathbb{x}, \mathbb{w})$ and $(\mathbb{x}', \mathbb{w}')$ have small Hamming distance), potentially with the help of a data structure. To the best of our knowledge, no commonly used zk-SNARKs are dynamic: even a single update to $(\mathbb{x}, \mathbb{w})$ currently requires recomputing the proof from scratch, which takes at least linear time. After formally defining dynamic zk-SNARKs, we present two constructions: one with $O(\sqrt{n}\log^2 n)$ update time and $O(1)$ proof size (Dynaverse), and another with $O(\log^3 n)$ update time and $O(\log^3 n)$ proof size (Dynalog). Both Dynaverse and Dynalog rest on Dynamo, a new zk-SNARK for permutation relations that we introduce. Crucially, Dynamo is *sparse*, meaning its prover complexity depends only on the number of non-zero entries in the input vector. Our constructions can also be made universal in the random oracle model. We highlight two central applications of dynamic zk-SNARKs. First, we show that they naturally give rise to sparse zk-SNARKs—SNARKs whose prover complexity can be sublinear when the witness vector contains many zeros. In addition, by slightly modifying Dynaverse (rather than using it as a black box), we construct Aero, which to the best of our knowledge is the first sparse zk-SNARK with $O(k\log^2 k)$ prover complexity, where k is the Hamming weight of the witness. Second, we develop a compiler from any dynamic zk-SNARK to *recursion-free* and *bounded* incremental verifiable computation (BIVC). Interestingly, when instantiated with a dynamic zk-SNARK that uses a sublinear-size data structure (which we build and call Dynavold), this transformation yields the first BIVC scheme with sublinear state. We finally discuss further applications of dynamic zk-SNARKs, including dynamic state proofs and dynamic ML proofs for retraining.

1 Introduction

Zero-Knowledge Proofs (ZKPs) [18] are among the most fundamental primitives in Computer Science and cryptography. Given an NP language $\mathcal{L}$ and a statement $\mathbb{x} \in \mathcal{L}$ with respective witness $\mathbb{w}$, a ZKP system comprises a set of protocols that enable a party to prove that $\mathbb{x} \in \mathcal{L}$ without revealing anything about the witness $\mathbb{w}$. A ZKP can be seen as an alternative way of engaging with computation: In traditional computation, one can verify that $\mathbb{x} \in \mathcal{L}$ by simply checking the witness $\mathbb{w}$ against the NP relation. In contrast, in a ZKP system, one can verify that $\mathbb{x} \in \mathcal{L}$ via a (succinct) proof π, which reveals no information about the witness $\mathbb{w}$. In traditional computation, the concept of a *data structure* is fundamental—it is an object that allows us, for some languages $\mathcal{L}$ [24], to *much more efficiently* decide whether $\mathbb{x} \in \mathcal{L}$ when another (close) instance of the problem $\mathbb{x}'$ has already been processed. Surprisingly, this central concept is yet to be explored in the context of ZKPs. Motivated by applications mainly in the blockchain and ML space, in this paper we focus on *non-interactive* ZKP systems with *succinct* proofs and computational soundness, commonly known as zk-SNARKs [19], and put forth the problem of *dynamic zk-SNARKs*—zk-SNARKs with efficiently-updatable proofs.

To the best of our knowledge, none of the commonly-used zk-SNARKs, such as Groth16 [19], PLONK [16], Bulletproofs [6] and Orion [41], are dynamic: A single update in $(\mathbb{x}, \mathbb{w})$ can be handled only by recomputing the proof, which requires at least linear time. Most of the times, this is due to Fiat-Shamir, that outputs randomness depending on all circuit wires, or the use of polynomial division, sensitive to the changes on the dividend polynomial encoding the witness.

Any application that requires maintaining a computation proof while data evolve is a natural candidate for benefiting from a dynamic zk-SNARK. For example, consider the following "commit-and-prove" map-reduce application appearing in zk-coprocessors (e.g., [21]), where a dynamic zk-SNARK is useful: A prover Merkle-commits to a set of elements $x_1, \ldots, x_n$ outputting a commitment d. Then the prover provides a proof π for the public statement (d, cnt), where cnt is the number of elements x_i (in a Merkle tree whose commitment is d) satisfying a fixed predicate (e.g., signature verification under a public key). Now, whenever any element x_i of the Merkle tree changes (e.g., during a database update), a dynamic zk-SNARK would provide a way to update π to π' efficiently without proof recomputation—just as the Merkle commitment can be updated without recomputation. Of course, appending a Merkle proof for the changed element to the existing SNARK proof and re-computing the predicate on the verified, updated Merkle element would not work, since the SNARK proof after t updates would be proportional to t—here we aim to have *succinct* dynamic proofs. See Section 1.6 for more examples of such applications for dynamic zk-SNARKs.

1.1 Summary of our results

We begin this line of work first by formally defining dynamic zk-SNARKs—see Definition 1. Naturally, a dynamic zk-SNARK for a relation $\mathcal{R}$ is a zk-SNARK

Table 1. Asymptotic comparison Groth16 [19] and PLONK [16] (static zk-SNARKs) with Dynaverse (Section 5) and Dynalog (Section 6) (dynamic zk-SNARKs). In the table below, $\mathcal{G}$ is the key generation algorithm, $\mathcal{P}$ is the prove algorithm, $\mathcal{V}$ is the verification algorithm, $|\pi|$ is the proof size, $|\mathsf{pk}|$ is the prover key size, $|\mathsf{vk}|$ is the verification key size, k is the number of updates between source/target statements and $n = |\mathbb{x}| + |\mathbb{w}|$. Security is in the Algebraic Group Model (AGM) and RO stands for "random oracle".

scheme	$\mathcal{G}$	$\mathcal{P}$	$\mathcal{U}$	$\mathcal{V}$	$\lvert\pi\rvert$	$\lvert\mathsf{pk}\rvert$	$\lvert\mathsf{vk}\rvert$	**security** (circuit-specific)	**security** (universal)
Groth16	n	$n \log n$	✗	1	1	n	1	q-DLOG	✗
PLONK	n	$n \log n$	✗	1	1	n	1	q-DLOG,RO	q-DLOG, RO
Dynaverse	n	$n \log n$	$k\sqrt{n}\log^2 n$	1	1	n	1	q-DLOG	q-DLOG, RO
Dynalog	$n \log n$	$n \log^2 n$	$k \log^3 n$	$\log^3 n$	$\log^3 n$	$n \log^3 n$	$\log^2 n$	q-DLOG, q-ASDBP, RO	

with an additional *update* algorithm $\mathcal{U}$: Algorithm $\mathcal{U}$, run by the prover, takes as input a valid source statement-witness pair $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ along with a verifying proof π and a valid target statement-witness pair $(\mathbb{x}', \mathbb{w}') \in \mathcal{R}$. It outputs a verifying proof π' for $(\mathbb{x}', \mathbb{w}')$ *without* running the prover algorithm $\mathcal{P}$ from scratch, potentially with the help of a data structure aux. In particular, we are only interested in an algorithm $\mathcal{U}$ whose running time for a single change in $(\mathbb{x}, \mathbb{w})$ is *sublinear*. After presenting the definition, we provide our central building block and main result, a *sparse* zk-SNARK for permutation relations.

Dynamo: A new sparse zk-SNARK for permutation relations. All our dynamic zk-SNARK constructions rest on a sparse zk-SNARK for permutation relations (called Dynamo—see Section 4) that we devise and that can be of independent interest (Recall that permutation agruments are crucial for zk-SNARKs since they enforce consistency among circuit wires.) Given a commitment to a permutation σ of size m, one can prove, using Dynamo, that a vector $\mathbf{z}$ of size m satisfies σ (in the sense that $\mathbf{z}[i] = \mathbf{z}[\sigma(i)]$) in time that is proportional to the number of non-zero elements of $\mathbf{z}$—that is why it is sparse. As a matter of fact, the Dynamo proof is just a linear combination over a set of fixed group elements.

Dynaverse and Dynalog. Based on Dynamo we then provide our two dynamic zk-SNARKs constructions: (i) Dynaverse, with $O(\sqrt{n}\log^2 n)$ update time and $O(1)$ proof size (Section 5); (b) Dynalog, with $O(\log^3 n)$ update time and $O(\log^3 n)$ proof size (Section 6). Both constructions can also be instantiated as universal dynamic zk-SNARKs. A detailed comparison of our constructions with the most efficient static zk-SNARKs [19,16] is shown in Table 1.

Application 1: Sparse zk-SNARKs. As a first application of dynamic zk-SNARKs, we observe there is a trivial compiler that, on input any dynamic zk-SNARK of update time $U(n)$, it outputs a general-purpose "sparse" zk-SNARK of prover time $k \cdot U(n)$, where k is the Hamming weight of the sparse statement $\mathbb{x}||\mathbb{w}$: All you have to do is compute the proof for $\mathbb{x}||\mathbb{w}$ by simply running an update on the dynamic SNARK proof for $\mathbf{0}||\mathbf{0}$, which can be computed at setup time and fixed thereafter. Clearly, such an approach will result in prover time $k \cdot U(n)$, where k is the Hamming weight of $\mathbb{x}'||\mathbb{w}'$ and $U(n)$ is the update time of the underlying dynamic zk-SNARK. While this can technically yield a sparse

zk-SNARK (sublinear proof computation due to zeros), its prover time can still be large, e.g., $k\sqrt{n}$, when a $\sqrt{n}$ dynamic zk-SNARK is used. We observe that by doing a minor modification of Dynaverse (instead of using it as a black box), we can output Aero (Corollary 1), a sparse zk-SNARK with $O(k \log^2 k)$ prover time, where k is the Hamming distance of $\mathbb{x}||\mathbb{w}$. Aero has excellent asymptotics, such as constant proof size and can also be made universal. To the best of our knowledge, Aero *is the first general-purpose sparse zk-SNARK.*

We note here that we are not the first to consider the problem of sparse proof systems. In particular, Lasso [33], as well as Twist and Shout [32], construct sparse lookup arguments for lookup tables that contain very few non-zero entries. Spartan [31] proposes a sparse PCS called Spark, and Lasso [33] proposes Surge, a generalization of Spark, to commit to a sparse matrix. The SpeedySpartan and Spartan++ [32] substantially improve the Spartan [31] prover time. However, as opposed to Aero, the dominant overhead in this line of work remains linear in the size of the statement rather than proportional to the number of non-zeros.

Application 2: Sublinear-state recursion-free BIVC. As a second application of dynamic zk-SNARKs we build *recursion-free bounded incremental verifiable computation* (BIVC), a version of incremental verifiable computation first introduced by Tyagi et al. [39] that builds IVC [40] *without recursion* at the expense of placing a bound on the number of IVC iterations allowed. We show that there is a compiler from dynamic zk-SNARKs to recursion-free BIVC, and in particular, when the compilation is instantiated with a dynamic zk-SNARK that has a sublinear-size data structure, the resulting BIVC scheme has sublinear state, which is crucial when used in proof-carrying data applications where every distributed node must ship its state and proof to the next computation node. Indeed, based off Dynamo, we provide a dynamic zk-NARK (a dynamic non-interactive argument system that is *not* succinct, hence not a zk-SNARK) called Dynavold (see Section H) that has $\sqrt{n}$ proof size and $\sqrt{n}$ data structure size, leading to the first sublinear-state, recursion-free BIVC—see Section 7. For relation between Dynamo, Dynaverse, Dynalog, Aero, and Dynavold please see Fig. 1.

Preliminary experimental evaluation. We performed a preliminary experimental evaluation of Dynaverse and compared our prover times and proof sizes with PLONK [16]. As expected, our update is 2.1× to 43.54× faster than the PLONK recomputation for circuit sizes between 2^{18} to 2^{24}. However, Dynaverse verification time and proof size are more expensive but still reasonable—up to 42.34 milliseconds for verification and 5.86 KiB for proof size even when $n = 2^{24}$.

1.2 Technical overview of Dynamo

Let $\mathcal{R}_{\mathcal{P}}$ be a permutation relation that contains $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = ([m, \sigma], \mathsf{com}(\mathbf{z}), \mathbf{z})$ such that $\mathbf{z}$ is an m-sized vector satisfying $\mathbf{z}[i] = \mathbf{z}[\sigma(i)]$ for all $i \in [m]$ (Think of $\mathsf{com}(\mathbf{z})$ as a KZG commitment [20].) Dynamo is a sparse zk-SNARK for a relaxed version of $\mathcal{R}_{\mathcal{P}}$, $\mathcal{R}^r_{\mathcal{P}}$, containing $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = ([m, \sigma], (\mathsf{com}(\mathbf{z}), \mathsf{com}(\mathbf{h})), (\mathbf{z}, \mathbf{h}))$ such that $\mathbf{z}$ and $\mathbf{h}$ are m-sized vectors satisfying $\mathbf{z}[i] = \mathbf{z}[\sigma(i)] + \mathbf{h}[i]$ for all $i \in [m]$

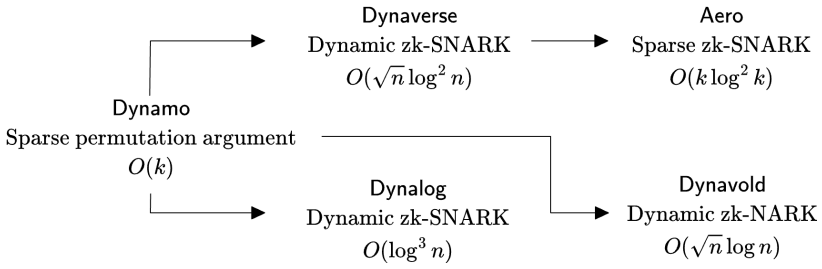

Fig. 1. Relationship between our proof systems. Here, n denotes the number of multiplication and addition gates, and k denotes the number of non-zero wires.

(Looking ahead: Using the permutation decomposition lemma (Lemma 2), we will reduce membership in $\mathcal{R}_{\mathcal{P}}$ to membership in $\mathcal{R}^r_{\mathcal{P}}$, which will simplify the task of building a dynamic zk-SNARK by "localizing" the impact of an update.)

Technical overview of Dynamo. Given a permutation σ of size m and a vector $\mathbf{z}$ of size m, we define a permutation polynomial $s(Y)$ over a finite field $\mathbb{F}$ as $s(Y) = \sum_{i\in[m]}(\mathbf{z}[i] - \mathbf{z}[\sigma(i)]) \cdot L_i(Y)$. Note that $s(Y)$, by a simple change of variable, can also be written as

$$s(Y) = \sum_{i\in[m]} \mathbf{z}[i] \cdot \left(L_i(Y) - L_{\sigma^{-1}(i)}(Y)\right) .$$

Consider now a vector $\mathbf{h}$ of size m and its Lagrange polynomial $h(Y)$, i.e., $h(Y) = \sum_{i\in[m]} \mathbf{h}[i]\cdot L_i(Y)$. It is easy to see that $([m,\sigma], (\mathsf{com}(\mathbf{z}), \mathsf{com}(\mathbf{h})), (\mathbf{z}, \mathbf{h})) \in \mathcal{R}^r_{\mathcal{P}}$ if and only if $s(Y) = h(Y)$ for all $Y = \omega^i$ where $i \in [m]$. We build our permutation argument on this idea: In particular, we have a prover commit to vectors $\mathbf{z}$ and $\mathbf{h}$ and provide a proof that, for given σ, $s(Y) - h(Y)$ is the zero polynomial. The high-level reason Dynamo is a sparse argument is due to the fact that terms of zeros $\mathbf{z}[i]$'s vanish in the above sums. See Section 4 for details.

1.3 Technical overview of Dynaverse and Aero

The main idea behind Dynaverse is the following "weak" version of a dynamic SNARK: Instead of supporting updates on an arbitrary valid statement-witness pair (as required by our definition), we will fix an "anchor" statement $\mathbf{z} = \mathbb{x}||\mathbb{w}$ along with its proof and we will support efficient updates only from $\mathbf{z}$. In particular, for every other statement $\mathbf{z}' = \mathbb{x}'||\mathbb{w}'$ that has Hamming distance k from $\mathbf{z}$, Dynaverse can compute the new proof π' in time $O(k\log^2 k)$. The new proof π' contains the anchor proof π along with a proof π^* encoding the difference between π and π'. Let this difference be encoded by the statement $\mathbf{z}^* = \mathbb{x}^*||\mathbb{w}^*$. For example, to prove that the updated statement $\mathbf{z}'$ satisfies the SNARK permutation σ, all the prover has to do is to return a Dynamo proof π_R for $\mathbf{z}$ (precomputed at setup) and a dynamo proof π^*_R for $\mathbf{z}^*$. Then, you are guaranteed that $\mathbf{z}' = \mathbf{z}+\mathbf{z}^*$ will satisfy σ. Crucially, computing the Dynamo proof for $\mathbf{z}^*$ can be done efficiently (in time $O(k)$) due to the sparsity of Dynamo. Achieving

the same goal for the gate constraints is more complex and requires proving only the gate constraints of the k affected gates, allowing us to achieve $O(k \log^2 k)$ time—independent of the size of the circuit n. Deriving the final dynamic zk-SNARK, Dynaverse, requires us to support updates from any (X, W), not just from the anchor $(\mathbb{x}, \mathbb{w})$. This is easy to achieve: Our update algorithm ignores the old statement (X, W) and always uses the anchor, as long as the Hamming distance of $(\mathbb{x}, \mathbb{w})$ and $(\mathbb{x}', \mathbb{w}')$ is less than $\sqrt{n}$. If not, the anchor proof is recomputed from scratch. A careful analysis shows that the amortized comlexity of this approach is $\sqrt{n} \log^2 n$ and standard tricks can be used to deamortize to worst-case $\sqrt{n} \log^2 n$. See Section 5.

Aero sparse zk-SNARK. It is straighforward to build Aero from Dynaverse: Use as anchor statement $(\mathbb{x}, \mathbb{w}) = (\mathbf{0}, \mathbf{0})$ and then, every $(\mathbb{x}', \mathbb{w}')$ with k non-zero elements will have Hamming distance k from $(\mathbb{x}, \mathbb{w})$, leading to $O(k \log^2 k)$ prover time. See Corollary 1. In general, as we mentioned before, we can build a sparse zk-SNARK from any dynamic zk-SNARK. However, the other direction is not true. A sparse zk-SNARK does not necessarily provide a dynamic zk-SNARK.

1.4 Technical overview of Dynalog

In Section 6, we present our polylogarithmic dynamic zk-SNARK construction, Dynalog. Dynalog can be viewed as yet another application of the permutation decomposition lemma (Lemma 2). In particular, Dynalog decomposes the witness vector $\mathbf{z} \in \mathbb{F}^m$ into $\ell = \log n + 1$ vectors $\mathbf{z}_0, \ldots, \mathbf{z}_\ell$ (all in $\mathbb{F}^m$) such that $\mathbf{z} = \sum \mathbf{z}_i$. Dynalog's permutation proof (computed with Dynamo) consists of $\ell = O(\log n)$ relaxed-permutation proofs, one per $\mathbf{z}_i$—as opposed to two such proofs in Dynaverse. In Dynalog, we crucially maintain the invariant that vector $\mathbf{z}_i$ has always at most 2^i non-zero elements by using a standard "waterfall" data structure, with levels that are exponentially-increasing in size. Therefore whenever there is an update only the relaxed permutation proof for the merged level k must be recomputed, which can be done in $O(2^k)$ time, due to sparsity of Dynamo. In amortized terms, this is just $O(\log n)$ time, which can also be deamortized with standard techniques.

Dynalog dynamic gate proof. While dynamically updating the permutation proof is a relatively straightforward application of the decomposition lemma, defining and updating the Dynalog gate proof is more challenging. In particular, one way to check the gate relations hold in Dynalog is to compute a single gate proof for the vector $\mathbf{z} = \sum \mathbf{z}_i$ since the witness $\mathbf{z}$ is distributed across the levels of the data structure. Clearly, this technique does not produce an updatable proof. Instead we produce $\ell + 1$ gate proofs $\pi_0, \ldots, \pi_\ell$ such that each proof π_j proves that the gate relationships hold for the sum $\sum_{i=j}^{\ell} \mathbf{z}_i$, but projected only on *all* the non-zero indices of $\mathbf{z}_j$. Now, it turns out that if all π_j's verify, then all gate constraints are satisfied for the committed $\mathbf{z}$, which is enough for proving security—see Lemma 3. All details of the construction can be found in Section 6.

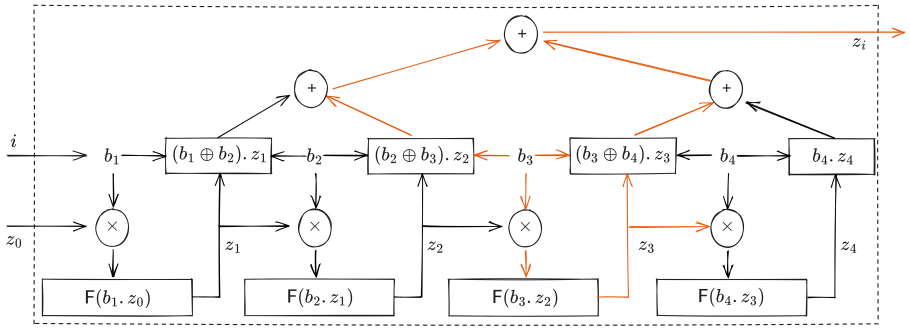

Fig. 2. Circuit F_N for recursion-free BIVC using dynamic zk-SNARKs. We do not show non-deterministic input for simplicity. When proceeding from step 2 (1100) to step 3 (1110), only $\tilde{O}(1)$ wires change, indicated with orange above.

1.5 From dynamic zk-SNARKs to recursion-free BIVC

As our second application, we propose a compiler from dynamic zk-SNARKs to *Bounded Incremental Verifiable Computation* (BIVC) *without recursion*. In BIVC, there is an initial input z_0, a function F and a bound N for the number of steps to be performed, and the goal is to provide a proof for the statement (i, z_0, z_i) (where $i \leq N$), meaning that z_i is the output of F on z_0 a number of i times, i.e., there exist $z_{i-1}, \ldots, z_1$ and $w_{i-1}, \ldots, w_0$ with $z_i = \mathsf{F}(z_{i-1}, w_{i-1}), z_{i-1} = \mathsf{F}(z_{i-2}, w_{i-2}), \ldots, z_1 = \mathsf{F}(z_0, w_0)$, where the w_i's denote non-deterministic inputs. The seminal work by Tyagi et al. [39] showed that it is possible to build BIVC *without recursion* by maintaining a tree of SNARK proofs that grows to the right as new iterations are executed. Their execution step takes $\log N + |\mathsf{F}|$ time, producing a proof of $\log^2 N$ size by using a logarithmic number of IPA protocol [5] instantiations to verify several SNARK proofs in a succinct manner. Crucial for the Tyagi et al. [39] construction is the requirement for a third party to maintain a linear-size data structure that assists the party running the computation to produce the proof for the latest execution step. Our compiler presents a tradeoff: It yields a construction with an increased computation time per step ($\sqrt{N|F|}$ v. $|F| + \log N$) but with a sublinear-size state ($\sqrt{N|F|}$ v. N), making it more appropriate for a distributed setting, e.g., not requiring a third party to store a data structure.

We build our compiler as follows: Given a dynamic zk-SNARK we build BIVC by treating every BIVC execution step as a SNARK update on a special circuit F_N of size N that has the following properties: (a) On input a counter i, initial value z_0 and final value z_i it checks whether z_i is the i-th application of F on z_0; (b) Every *neighboring* public statements (i, z_0, z_i) (with corresponding witness $\mathbb{w}_i$) and $(i+1, z_0, z_{i+1})$ (with corresponding witness $\mathbb{w}_{i+1}$) differ only in $\tilde{O}(1)$ wires. Note that this requires special care. For example, we had to pass the counter i in unary (we show how this does not affect the verifier efficiency in Section 7) and we had to sum N elements on a binary tree, instead on a line—see Figs. 2 and 16. Given the circuit F_N, it is now natural to build recursion-free BIVC using a dynamic zk-SNARK in the following way: To compute π_{i+1} for

$(i+1, z_0, z_{i+1})$, run the dynamic zk-SNARK update algorithm on π_i and aux_i. The size of the state of the resulting BIVC scheme depends on the dynamic zk-SNARK data structure size. In particular, we build Dynavold, a dynamic zk-NARK with sublinear-size data structure, yielding a recursion-free BIVC with sublinear state. Due to space limitations the Dynavold dynamic zk-SNARK is presented in Section H.

1.6 Other applications of dynamic zk-SNARKs

Any application that requires a computation proof while data evolve could benefit from a dynamic zk-SNARK. We provide two examples here.

Updating block proofs. Blockchain systems nowadays deploy smart contracts that must run complex computations (e.g., SQL) on transaction data that are ever changing, as new blocks are generated—yet smart contracts are computationally limited. For example, in Ethereum, consider a smart contract that needs to maintain a running count (per block) of accounts (corresponding to a specific ERC-20 token) with balances greater than a fixed threshold. Due to increased gas costs, the specific count for a specific block i is provided, by zk-coprocessors, directly to the contract along with a ZKP proof p_i. When block $i+1$ is generated, the count changes and therefore a new proof p_{i+1} is required. Dynamic zk-SNARKs can be used to efficiently compute proof p_{i+1} in time proportional to the number of changing balances, which is small (bounded by the size of the block) compared to the total number of ETH accounts (in the millions). Therefore the smart contract would be able to access the proof and the computation result fast, without having to wait for the costly linear-time proof computation. This is the motivation behind the Merkle example presented in the introduction.

Updating ML inference proofs. ML inference is a computationally intensive process, and can be performed at remote, potentially untrusted machines—therefore a zk-SNARK proof p can be used to increase the trust in this process. Once such a proof p of correct inference is derived, the model (e.g., weights) might change due to retraining and therefore a new proof p' might need to be computed. In particular, consider the most fundamental ML operation—multiplying a weight matrix $\mathbf{W}$ with a vector $\mathbf{t}$, outputting $\mathbf{y} = \mathbf{W} \cdot \mathbf{t}$ which requires an n^2 prover circuit. If a single element (or even a whole row) of the weight matrix $\mathbf{W}$ changes, one could use a dynamic zk-SNARK to update the proof in linear time, as opposed to quadratic time. This can speed up inference computation in such a ZK-enabled environment when some of the weights change.

1.7 Related work on dynamic zk-SNARKs

Some proof systems that are dynamic have been proposed before, but they use recursive zk-SNARKs as a black box. For example, Reckle trees [28] support dynamic batch proofs. In general, any type of proof-carrying data system [14] could be potentially turned into a dynamic zk-SNARK (e.g., see Mangrove [25]). However, as we mentioned before, black-box use of recursive zk-SNARKs accepts only

heuristic security arguments due to potential extractor size blowup issues [4], e.g., when recursion is applied for more than a constant number of iterations. In addition, when efficient recursive zk-SNARKs are used [36], one needs to encode the random oracle in the SNARK relation, leading to an additional heuristic security argument. In conclusion, dynamic zk-SNARKs implemented with recursion face severe security limitations. While recent works focus on resolving some of these issues [22,13], they operate on novel idealized models that are not well-studied for the time being. *Our work does not use recursive zk-SNARKs (or any static zk-SNARKs as black box)* and proposes fully-secure constructions for dynamic zk-SNARKs. It is however, worth noting that dynamic proof systems have appeared before in the literature that do not use recursion—but they have limited expressiveness. For example, authenticated data structures [35] and updatable vector commitments [9,34] are dynamic proof systems for simple data structure queries, such as membership, range search and vector queries. Other examples include certain constructions for batch-membership proofs, e.g., [7], as well as functional vector commitments supporting linear functions, e.g., [8]. Another related line of work is that of *malleable proofs* [11]. The goal of a malleable proof system is to compute a proof p' for a statement x', on input a proof p for a related statement x, without knowing the witness w' for x. Due to this, the extractability property is weaker, i.e., not guaranteed to extract a witness for derived proofs. Regarding *homomorphic proofs* [1], given only the proofs $p_1, \ldots, p_n$ for the outputs of circuits $y_1 = C_1(x_1), \ldots, y_n = C_n(x_n)$, the goal is to compute a proof p' for a circuit C' over the y_i values, without requiring x_i's. Unfortunately, it is not possible to capture arbitrary updates on the original data in this model. Finally, we note that in *neither malleable nor homomorphic proofs* do the definitions explicitly capture the performance requirement that updates are faster than proof recomputation. In fact, all proposed schemes have linear-size proofs, i.e., they are not succinct.

2 Preliminaries

Roots of unity, Lagrange polynomials and vectors. For m power of two, we denote with ω the m-th root of unity in a field $\mathbb{F}$, i.e., $\omega^m = 1$. We also use Ω to denote the set of m-th roots of unity, i.e., $\Omega = \{\omega, \ldots, \omega^m\}$. The Lagrange polynomial is $L_i(X) = \omega^i(X^m - 1)/m(X - \omega^i)$ such that $L_i(\omega^i) = 1$ and $L_i(\omega^j) = 0$ $(i \neq j)$. $[n]$ is the set $\{1, \ldots, n\}$ and $[n_1, n_2]$ is the set $\{n_1, n_1+1, \ldots, n_2-1, n_2\}$. For vector $\mathbf{z} = [a_1, \ldots, a_n]$, $\mathbf{z}[i:j]$ is the sub-vector $[a_i, \ldots, a_j]$.

Bilinear groups. Let $\mathbb{G}$ be group of prime order $\mathbb{p}$ with generator g and pairing function $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, such that $\forall u, w \in \mathbb{G}$ and $a, b \in \mathbb{Z}_{\mathbb{p}}$, it is $e(u^a, w^b) = e(u, w)^{ab}$. Let $\mathsf{pp}_{\mathsf{bl}} := (\mathbb{p}, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$ denote the pairing parameters. In particular, an implementation would use asymmetric pairings for efficiency, but we use symmetric pairings in our presentation for notational convenience.

KZG commitments. For any polynomial f, we use the notation $[f]$ for its KZG commitment [20], e.g., if f is bivariate with variables X, Y, then $[f] =$

$g^{f(\tau_X,\tau_Y)}$ where τ_X, τ_Y are secrets for X, Y respectively. See Section A for more preliminaries about KZG commitments.

Pairing-check. A pairing-check ensures that an adversary cannot output two different polynomial commitments $[f(X)]$ and $[g(X)]$ such that $f(X) \cdot g(X) = 0$ but $e([f(X)], [g(X)]) \neq 1_{\mathbb{G}_T}$. See Lemma 8 in Appendix C.

Degree-check. Degree-check is a useful tool to show the maximum degree of some variable(s) in a polynomial. In particular, when a prover commits to polynomial $f(X)$, we want to ensure that variable X has degree at most d and variable Y is not present. To do that, we ask the prover to provide a KZG commitment to $f(X) \cdot X^{q-d}Y^q$ as well, and we use the pairing to check whether $e([f(X)], [X^{q-d}Y^q]) = e([f(X) \cdot X^{q-d}Y^q], g)$. Clearly, if X has higher degree than d or Y was present in $f(X)$, the prover cannot compute $[f(X) \cdot X^{q-d}Y^q]$ since $[X^{q+1}]$ is not output as part of the setup. See Lemma 9 in Appendix C.

Indexed relations and permutation relation. Recall that a *relation* is a set of pairs $(\mathbb{x}, \mathbb{w})$. An *indexed relation* is a set of triples $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ where $\mathbb{i}$ is the index fixed at setup time. For instance, the indexed permutation relation $\mathcal{R}_{\mathcal{P}}$ is the set of tuples $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = ([m, \sigma], \mathsf{com}(\mathbf{z}), \mathbf{z})$, where σ is a permutation of size m, $\mathsf{com}(\mathbf{z})$ is a commitment to $\mathbf{z}$ which satisfies $\mathbf{z}[i] = \mathbf{z}[\sigma(i)]$ for all $i \in [m]$.

Plonkish arithmetization. Per Plonkish arithmetization [12,16], an indexed relation $\mathcal{R}_{\mathcal{C}}$ is the set of tuples $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ where $\mathbb{i} = [n, n_0, \sigma]$ is an index for a fan-in 2 arithmetic circuit $\mathcal{C}$ over $\mathbb{F}$ with n_0 input gates ($n_0 \leq n$), n addition gates and n multiplication gates (padding can handle the general case), where:

- Gate 1 to n are addition gates, gate $n+1$ to $2n$ are multiplication gates, and gates $2n+1$ to $2n+n_0$ are input gates (holding the public statement).
- $\sigma : [6n + n_0] \to [6n + n_0]$ is a permutation (bijection) describing the wire connections. For every addition gate i ($1 \leq i \leq n$), its left input, right input and output are labeled by $i,\ n+i,\ 2n+i$ respectively. Similarly, for every multiplication gate i ($n+1 \leq i \leq 2n$) its left input, right input and output are labeled by $2n+i,\ 3n+i,\ 4n+i$ respectively. Input wires are labeled from $6n+1$ to $6n+n_0$. E.g., if addition gate i's right input is connected to input wire j, then we have $\sigma(6n+j) = n+i$.

For any fixed index $\mathbb{i} = [n, n_0, \sigma]$ describing a circuit $\mathcal{C}$, an instance of public inputs $\mathbb{x} \in \mathbb{F}^{n_0}$, and a witness $\mathbb{w} \in \mathbb{F}^{6n}$, we have $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathcal{C}}$ if and only if the following hold, where $\mathbf{z} = [\mathbb{w}; \mathbb{x}] \in \mathbb{F}^{6n+n_0}$: (a) $([6n + n_0, \sigma], \mathsf{com}(\mathbf{z}), \mathbf{z}) \in \mathcal{R}_{\mathcal{P}}$; (b) $\forall i \in [n]$, $\mathbf{z}[i] + \mathbf{z}[n+i] = \mathbf{z}[2n+i]$ and $\mathbf{z}[3n+i] \cdot \mathbf{z}[4n+i] = \mathbf{z}[5n+i]$.

3 Dynamic and sparse zk-SNARKs definitions

In this section we present the formal definition of dynamic zk-SNARKs as well as sparse zk-SNARKs. Our new definition (Definition 1) is an extension of the original zk-SNARKs definition (Definition 3) in two ways, as we explain below: First we require an *updatability property*, stating that there should be an update algorithm $\mathcal{U}$, such that, on input a valid instance $(\mathbb{x}, \mathbb{w})$ along with its proof

π, a "data structure" aux and another valid instance $(\mathbb{x}', \mathbb{w}')$ that has "small" Hamming distance k from $(\mathbb{x}, \mathbb{w})$, it should be able to output the updated proof π' (along with the updated data structure aux') in time strictly less than $T(\mathcal{P})$, where $\mathcal{P}$ is the prove algorithm of the SNARK. Note the requirement for "small" Hamming distance is necessary: If, say, a linear number of positions change from $(\mathbb{x}, \mathbb{w})$ to $(\mathbb{x}', \mathbb{w}')$, it will be impossible to update the proof in sublinear time: If such an algorithm existed, it would have to ignore some of the updates. Second, we must slightly modify the definition for zero-knowledge. Now the simulator is asked to simulate not a single proof, but a series of honestly-generated proofs that are produced by running the update algorithm.

Definition 1 (Dynamic zk-SNARKs). *A dynamic zero-knowledge succinct non-interactive argument of knowledge (dynamic zk-SNARK) for indexed relation $\mathcal{R}$ is a tuple of the following PPT algorithms $\mathsf{DS} = (\mathcal{G}, \mathcal{P}, \mathcal{U}, \mathcal{V})$:*

- $\mathcal{G}(1^\lambda, \mathbb{i}) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$: *Given 1^λ and indexed relation $\mathbb{i}$, outputs prover key pk, an update key upk and a verifier key vk.*
- $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{aux})$: *Given prover key pk, instance $\mathbb{x}$, and witness $\mathbb{w}$, outputs a proof π and auxiliary information aux.*
- $\mathcal{U}(\mathsf{upk}, \mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}, \pi, \mathsf{aux}) \to (\pi', \mathsf{aux}')$: *Given update key upk, new instance $\mathbb{x}'$, new witness $\mathbb{w}'$, the previous proof π for instance $\mathbb{x}$ and witness $\mathbb{w}$ and auxiliary information aux, outputs a new proof π' for $\mathbb{x}'$ and $\mathbb{w}'$, and new auxiliary information aux'.*
- $\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$: *Given verifier key vk, instance $\mathbb{x}$, and a proof π, outputs accept or reject.*

A dynamic zk-SNARK DS should have polylog-sized proofs and satisfy the following properties.

- Updatability*: We say that DS satisfies* updatability *if there is a function $f(|\mathbb{x}| + |\mathbb{w}|) = o(|\mathbb{x}| + |\mathbb{w}|)$ such that algorithm $\mathcal{U}(\mathsf{upk}, \mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}, \pi, \mathsf{aux})$ runs in time $O(k \cdot f(|\mathbb{x}| + |\mathbb{w}|))$, where k is the Hamming distance of vectors $\mathbb{x}||\mathbb{w}$ and $\mathbb{x}'||\mathbb{w}'$.*[4]
- Completeness*: Let $(\mathsf{pk}, \mathsf{upk}, \mathsf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathbb{i})$. We say that DS satisfies* completeness *if for any $\mathbb{i}$, for any $(\mathbb{i}, \mathbb{x}_0, \mathbb{w}_0) \in \mathcal{R}, \ldots, (\mathbb{i}, \mathbb{x}_\ell, \mathbb{w}_\ell) \in \mathcal{R}$, if*

$$\mathcal{P}(\mathsf{pk}, \mathbb{x}_0, \mathbb{w}_0) \to (\pi_0, \mathsf{aux}_0), \text{ and}$$

$$\mathcal{U}(\mathsf{upk}, \mathbb{x}_{i+1}, \mathbb{w}_{i+1}, \mathbb{x}_i, \mathbb{w}_i, \pi_i, \mathsf{aux}_i) \to (\pi_{i+1}, \mathsf{aux}_{i+1}),$$

for $i = 0, \ldots, \ell - 1$, then $\mathcal{V}(\mathsf{vk}, \mathbb{x}_\ell, \pi_\ell) \to 1$.

[4] Note that for $k = o(T(\mathcal{P})/f(|\mathbb{x}| + |\mathbb{w}|))$, where $T(\mathcal{P})$ is the prover's runtime, this is $o(T(\mathcal{P}))$, as desired. Besides, for simplicity of notation, we write $\mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}$ as explicit inputs of $\mathcal{U}$ but, in practice, it suffices to receive the old and new instance/witness elements at the k modified positions.

– Knowledge Soundness*: We say that* DS *satisfies* knowledge soundness *if for any PPT adversary $\mathcal{A}$ and for any $\mathtt{i}$ there exists a PPT extractor $\mathcal{E}_{\mathcal{A}}$ such that*

$$\Pr\left[\begin{array}{c}\mathcal{G}(1^\lambda,\mathtt{i})\to(\mathsf{pk},\mathsf{upk},\mathsf{vk});((\mathbb{x},\pi);\mathbb{w})\leftarrow(\mathcal{A}||\mathcal{E}_{\mathcal{A}})(\mathsf{pk},\mathsf{upk},\mathsf{vk})\\ :\\ \mathcal{V}(\mathsf{vk},\mathbb{x},\pi)\to 1\wedge(\mathtt{i},\mathbb{x},\mathbb{w})\notin\mathcal{R}\end{array}\right]$$

is negligible.

– Zero Knowledge*: Fix any $\mathtt{i}$ and $(\mathtt{i},\mathbb{x}_0,\mathbb{w}_0)\in\mathcal{R},\ldots,(\mathtt{i},\mathbb{x}_\ell,\mathbb{w}_\ell)\in\mathcal{R}$ for some polynomially bounded ℓ. Let $\mathcal{D}$ be the distribution of $(\pi_0,\ldots,\pi_\ell)$ as output by the experiment below.*
  1. $\mathcal{G}(1^\lambda,\mathtt{i})\to(\mathsf{pk},\mathsf{upk},\mathsf{vk})$.
  2. $\mathcal{P}(\mathsf{pk},\mathbb{x}_0,\mathbb{w}_0)\to(\pi_0,\mathsf{aux}_0)$.
  3. $\mathcal{U}(\mathsf{upk},\mathbb{x}_{i+1},\mathbb{w}_{i+1},\mathbb{x}_i,\mathbb{w}_i,\pi_i,\mathsf{aux}_i)\to(\pi_{i+1},\mathsf{aux}_{i+1})$, *for* $i=0,\ldots,\ell-1$.

  We say that DS *satisfies* zero-knowledge *if there exists a PPT simulator $\mathcal{S}$ such that the distribution $\tilde{\mathcal{D}}$ of $(\tilde\pi_0,\ldots,\tilde\pi_\ell)$ output by the following experiment is computationally-indistinguishable from $\mathcal{D}$.*
  1. $(t,\mathsf{pk},\mathsf{upk},\mathsf{vk})\leftarrow\mathcal{S}(1^\lambda,\mathtt{i})$.
  2. $(\tilde\pi_0,\ldots,\tilde\pi_\ell)\leftarrow\mathcal{S}(t,\mathsf{pk},\mathsf{upk},\mathsf{vk},\mathbb{x}_0,\ldots,\mathbb{x}_\ell)$.

  This definition also extends to statistical/perfect zero-knowledge.

Note here that one can envision a stronger definition for zero-knowledge which requires that the output of $\mathcal{U}$ is indistinguishable from the output of $\mathcal{P}$. Our definition above, however, is still meaningful (and we keep it simple on purpose) since in most applications (e.g., blockchains) it is public knowledge that updates take place. In any case, our constructions also satisfy this stronger indistinguishability definition.

Given that we will be building a sparse zk-SNARK we provide the following definition. In particular, a sparse zk-SNARK is a SNARK whose prover time can become sublinear when the Hamming weight of $\mathbb{x}||\mathbb{w}$ is small.

Definition 2 (Sparse zk-SNARKs). *A zk-SNARK $(\mathcal{G},\mathcal{P},\mathcal{V})$ is* sparse *if there exists a function $f(|\mathbb{x}|+|\mathbb{w}|)=o(|\mathbb{x}|+|\mathbb{w}|)$ such that for all $0\le k\le|\mathbb{x}|+|\mathbb{w}|$, the running time of $\mathcal{P}(\mathsf{pk},\mathbb{x},\mathbb{w})$ is $O(k\cdot f)$, where k is the Hamming weight of $\mathbb{x}||\mathbb{w}$.*

4 Dynamo: A sparse zk-SNARK for permutation relations

We now introduce our central building block to be used in our dynamic zk-SNARKs: A *sparse* zk-SNARK for a *relaxed* permutation relation. Formally, we want to build a zk-SNARK for the indexed relation $\mathcal{R}^r_{\mathcal{P}}$ that contains those tuples $(\mathtt{i},\mathbb{x},\mathbb{w})=([m,\sigma],(\mathsf{com}(\mathbf{z}),\mathsf{com}(\mathbf{h})),(\mathbf{z},\mathbf{h}))$ such that

$$\mathbf{z}[i]=\mathbf{z}[\sigma(i)]+\mathbf{h}[i],\text{ for all } i=1,\ldots,m\,.$$

We call this zk-SNARK Dynamo.

Our starting point: Permutation polynomial. Given a permutation σ of size m and a vector $\mathbf{z}$ of size m, one can define a permutation polynomial $s(Y)$ over a finite field $\mathbb{F}$ as

$$s(Y) = \sum_{i\in[m]} (\mathbf{z}[i] - \mathbf{z}[\sigma(i)]) \cdot L_i(Y).$$

Note that $s(Y)$, by a simple change of variable, can also be written as

$$s(Y) = \sum_{i\in[m]} \mathbf{z}[i] \cdot \big(L_i(Y) - L_{\sigma^{-1}(i)}(Y)\big) = \sum_{i\in[m]} \mathbf{z}[i] \cdot \mathbf{y}_i(Y)\,,$$

where, for ease of notation, we write $\mathbf{y}_i(Y) = L_i(Y) - L_{\sigma^{-1}(i)}(Y)$. Consider now a vector $\mathbf{h}$ of size m and its Lagrange polynomial $h(Y)$, i.e.,

$$h(Y) = \sum_{i\in[m]} \mathbf{h}[i] \cdot L_i(Y)\,. \tag{1}$$

It is easy to see that $([m,\sigma], (\mathsf{com}(\mathbf{z}), \mathsf{com}(\mathbf{h})), (\mathbf{z},\mathbf{h})) \in \mathcal{R}^r_{\mathcal{P}}$ if and only if $s(Y) = h(Y)$ for all $Y = \omega^i$, $i \in [m]$. We build our permutation argument on this idea: In particular, we have a prover commit to vectors $\mathbf{z}$ and $\mathbf{h}$ and provide a proof that, for given σ, $s(Y) - h(Y)$ is the zero polynomial. We pick $\mathsf{com}(\mathbf{z}), \mathsf{com}(\mathbf{h})$ as the KZG commitments to the Lagrange interpolations of $\mathbf{z}$ and $\mathbf{h}$. More specifically, $\mathsf{com}(\mathbf{z}) = [z(X)]$ and $\mathsf{com}(\mathbf{h}) = [h(Y)]$ where $z(X)$ encodes the $\mathbf{z}$ elements ($z(\omega^i) = \mathbf{z}[i]$ for all $i = 1,\ldots,m$), i.e.,

$$z(X) = \sum_{i\in[m]} L_i(X) \cdot \mathbf{z}[i]\,, \tag{2}$$

and $h(Y)$ encodes the $\mathbf{h}$ elements as in Eq. (1).

Computing the proof. To compute the proof, the prover will commit to a bivariate polynomial $v(X,Y)$, using Lagrange interpolation and KZG commitments. In particular $v(X,Y)$ encodes $\mathbf{z}[i] \cdot \mathbf{y}_i(Y)$ ($v(\omega^i, Y) = \mathbf{z}[i] \cdot \mathbf{y}_i(Y)$ for all $i = 1,\ldots,m$), i.e.,

$$v(X,Y) = \sum_{i\in[m]} L_i(X) \cdot \mathbf{z}[i] \cdot \mathbf{y}_i(Y)\,. \tag{3}$$

The input of the verifier is an honestly-computed commitment to a bivariate polynomial $u(X,Y)$ that encodes the permutation σ in a natural manner, i.e.,

$$u(X,Y) = \sum_{i\in[m]} L_i(X) \cdot \mathbf{y}_i(Y)\,. \tag{4}$$

Now to prove that the commitments to the polynomials $z(X)$, $h(Y)$, $v(X,Y)$ and $u(X,Y)$ satisfy $s(Y) = h(Y)$ the prover must provide additional proofs (in addition to the commitment of z, h and v) as we detail in the following.

Zero-check. First the prover must prove that for all $i = 1, \ldots, m$ it is $v(\omega^i, Y) = u(\omega^i, Y) \cdot z(\omega^i)$. This is a standard zero-check (see Section A) for the polynomial $u(X, Y) \cdot z(X) - v(X, Y)$ on the set (Ω, Y), where $\Omega = \{\omega, \ldots, \omega^m\}$. The proof for that is a KZG commitment to the quotient polynomial

$$\alpha(X, Y) = \frac{u(X, Y) \cdot z(X) - v(X, Y)}{X^m - 1}. \tag{5}$$

Sum-check. Finally, the prover will have to provide a proof that

$$\sum_{i \in [m]} v(\omega^i, Y) = h(Y). \tag{6}$$

Inspired by the univariate sum-check from Aurora [3], as long as $v(X, Y)$ has degree at most $m-1$ in X, we have that $v(X, Y)$ can be written as $\sum_{i \in [m]} L_i(X) \cdot v(\omega^i, Y)$ and therefore

$$v(0, Y) = \sum_{i \in [m]} L_i(0) \cdot v(\omega^i, Y) = \frac{1}{m} \cdot \sum_{i \in [m]} v(\omega^i, Y).$$

Therefore showing Eq. (6) is true is equivalent to showing that (i) $v(X, Y)$ has degree at most $m-1$ over X, which we can do via a "degree check" as we described in Section 2; (ii) $v(0, Y) = \frac{1}{m} h(Y)$, which we can do through a standard KZG commitment to the quotient polynomial

$$\beta(X, Y) = \frac{v(X, Y) - \frac{1}{m} h(Y)}{X}. \tag{7}$$

Wrapping up. By combining Eqs. (5) and (7), we can remove $v(X, Y)$ and check the following equation

$$u(X, Y) \cdot z(X) = \alpha(X, Y)(X^m - 1) + \beta(X, Y) \cdot X + \frac{1}{m} h(Y) \tag{8}$$

as well as the degree for $z(X)$ (degree 0 in Y), the degree of $h(Y)$ (0 in X) and the degree of $\beta(X, Y)$ (at most $m - 2$ in X). The final Dynamo proof for index $\mathbb{i} = [m, \sigma]$ consists of five KZG commitments to the polynomials $\alpha(X, Y)$ and $\beta(X, Y)$ as defined above as well as an additional three KZG commitments to polynomials $Z(X, Y)$, $H(X, Y)$ and $B(X, Y)$ used for degree checks via $z(X) \cdot Y^m = Z(X, Y)$, $h(Y) \cdot X^m = H(X, Y)$ and $\beta(X, Y) \cdot X^2 = B(X, Y)$.

Sparsity of Dynamo. One notable feature of our Dynamo SNARK is that it is particularly structured, allowing us to express each commitment contained in the proof as a linear combination of vector $\mathbf{z}$. This feature is crucial for using Dynamo in our dynamic SNARK later since *it allows us to compute the proof in time proportional to the number of non-zero elements of* $\mathbf{z}$, as opposed to the size m of $\mathbf{z}$. Interestingly, this feature allows us to also update a Dynamo proof in constant time. We now have the following theorem for proof computation, whose proof can be found in Section I.

– $\mathcal{G}(1^\lambda, [m, \sigma]) \to (\mathsf{pk}, \mathsf{vk})$:
  - Let $\mathcal{F} = \{\alpha, \beta, Z, H, B\}$.
  - Pick random τ_X, τ_Y from $\mathbb{F}$ for variables X and Y respectively.
  - Set pk to contain the following KZG commitments, defined in Theorem 1, and computed using τ_X and τ_Y directly $\{[f_1], \ldots, [f_m]\}_{f \in \mathcal{F}}$.
  - Set $\mathsf{vk} = \{[u(X, Y)], [X^2], [X^m], [Y^m]\}$ (u is from Eq. (4)).

– $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to \pi$:
  - Parse $\mathbb{x}$ as $([z], [h])$ and $\mathbb{w}$ as $\mathbf{z}[1], \ldots, \mathbf{z}[m]$ and $\mathbf{h}[1], \ldots, \mathbf{h}[m]$.
  - Following Theorem 1, output six commitments as π, i.e., $\forall f \in \mathcal{F}$ output

$$[f] = \prod_{i \in [m]} [f_i]^{\mathbf{z}[i]} .$$

– $\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$:
  - Parse vk as $\{[u], [X^2], [X^m], [Y^m]\}$, $\mathbb{x}$ as $[z], [h]$, π as $\{[\alpha], [\beta], [H], [Z], [B]\}$.
  - Output 1 if and only if all the following relations hold:

$$e([u], [z]) = e([\alpha], [X^m - 1]) \cdot e([\beta], [X]) \cdot e([h]^{1/m}], g).$$
$$e([z], [Y^m]]) = e([Z], g).$$
$$e([h], [X^m]]) = e([H], g).$$
$$e([\beta], [X^2]]) = e([B], g).$$

Fig. 3. The Dynamo SNARK.

Theorem 1 (Dynamo sparsity). *Let $\mathcal{F} = \{\alpha, \beta, Z, H, B\}$ be the set of five polynomials contained in the* Dynamo *proof. Every $f \in \mathcal{F}$ can be expressed as*

$$f = \sum_{i \in [m]} f_i \cdot \mathbf{z}[i] \,,$$

where $\{f_1, \ldots, f_m\}_{f \in \mathcal{F}}$ are five fixed sets of m polynomials each defined for every $f \in \mathcal{F}$.

For example, for $Z(X, Y) \in \mathcal{F}$ we have $Z(X, Y) = Y^m \sum_{i \in [m]} L_i(X) \cdot \mathbf{z}[i]$ and therefore the set of polynomials $\{Z_1, \ldots, Z_m\}$ for Z is $\{Y^m L_1(X), \ldots, Y^m L_m(X)\}$. We derive closed formulas for the remaining polynomials in the proof.

Final protocol. Our complete circuit-specific Dynamo protocol is shown in Fig. 3. We summarize our protocol in the following theorem (see the proof in Section I).

Theorem 2 (Dynamo). *The protocol of Fig. 3 is a sparse SNARK (per Definitions 2 and 3) for $\mathcal{R}^r_{\mathcal{P}}$ based on q-DLOG (see Assumption 1) in the AGM. Its complexities are as follows.*

1. *$\mathcal{G}$ runs in $O(m)$ time, outputs pk of $O(m)$ size and vk of $O(1)$ size;*
2. *$\mathcal{P}$ runs in $O(k)$ time and outputs a proof π of $O(1)$ size, where k is the number of non-zero entries in $\mathbf{z}$;*
3. *$\mathcal{V}$ runs in $O(1)$ time.*

We now note that by using the random oracle we can provide a universal version of Dynamo. Due to space limitations, we provide this in Section G.

Theorem 3 (Universal Dynamo). *Assuming a random oracle* $\mathsf{H} : \{0,1\}^* \to \mathbb{F}$*, the protocol of Fig. 12 is a universal SNARK (per Definition 3) for* $\mathcal{R}_{\mathcal{P}}^r$ *based on q-DLOG (see Assumption 1) in the AGM. Its complexities are as follows.*

1. $\mathcal{G}$ *runs in* $O(m)$ *time and outputs* pp *of* $O(m)$ *size;*
2. $\mathcal{I}$ *runs in* $O(m \log m)$ *time, outputs* pk *of* $O(m)$ *size and* vk *of* $O(1)$ *size;*
3. $\mathcal{P}$ *runs in* $O(k)$ *time and outputs a proof* π *of* $O(1)$ *size, where* k *is the number of non-zero entries in* $\mathbf{z}$*;*
4. $\mathcal{V}$ *runs in* $O(1)$ *time.*

We finally note, that by following ideas from [16,30], we can use random masks for the polynomials and add zero-knowledge to Dynamo. The proof of the lemma below can be found in Section J.

Lemma 1 (Zero-knowledge (universal) Dynamo). *There are zero-knowledge versions of* Dynamo *and universal* Dynamo *with the same complexities.*

4.1 Decomposition of permutation relations using Dynamo

Consider a vector $\mathbf{z}$ of size m for which we want to show that it satisfies the permutation relation σ. In our constructions, vector $\mathbf{z}$ is typically decomposed into a set of vectors $\mathbf{z}_1, \ldots, \mathbf{z}_k$ (each being of size m) such that $\sum_{i=1}^k \mathbf{z}_i = \mathbf{z}$. If we can show that for all $j = 1, \ldots, k$ and for all $i = 1, \ldots, m$, it is $\mathbf{z}_j[i] = \mathbf{z}_j[\sigma(i)] + \mathbf{h}_j[i]$ (in other words $([m, \sigma], (\mathsf{com}(\mathbf{z}_j), \mathsf{com}(\mathbf{h}_j)), (\mathbf{z}_j, \mathbf{h}_j)) \in \mathcal{R}_{\mathcal{P}}^r$ for all $j = 1, \ldots, k$) while also $\sum_{j=1}^k \mathbf{h}_j = \mathbf{0}$ then it is $([m, \sigma], \mathsf{com}(\sum_{j=1}^k \mathbf{z}_j), \sum_{j=1}^k \mathbf{z}_j) \in \mathcal{R}_{\mathcal{P}}$. In particular, as we will see, this is an iff relationship.

Lemma 2 (Permutation decomposition). *Let* $\mathbf{z}_1, \ldots, \mathbf{z}_k$ *be vectors in* $\mathbb{F}^m$ *and let* $\mathbf{z} = \sum_{i=1}^k \mathbf{z}_i \in \mathbb{F}^m$*. Then* $([m, \sigma], \mathsf{com}(\mathbf{z}), \mathbf{z}) \in \mathcal{R}_{\mathcal{P}}$ *if and only if there exist* $\mathbf{h}_1, \ldots, \mathbf{h}_k \in \mathbb{F}^m$ *such that*

$$\sum_{j=1}^k \mathbf{h}_j = \mathbf{0} \text{ and } ([m, \sigma], (\mathsf{com}(\mathbf{z}_j), \mathsf{com}(\mathbf{h}_j)), (\mathbf{z}_j, \mathbf{h}_j)) \in \mathcal{R}_{\mathcal{P}}^r \text{ for all } j = 1, \ldots, k\,.$$

Proof. Let $\mathbf{z} = \sum_{i=1}^k \mathbf{z}_i$. It is $([m, \sigma], \mathsf{com}(\mathbf{z}), \mathbf{z}) \in \mathcal{R}_{\mathcal{P}}$ if and only if for all $i \in [m]$ we have $\sum_{j=1}^k (\mathbf{z}_j[i] - \mathbf{z}_j[\sigma(i)]) = 0$. If we set $\mathbf{h}_j[i] = \mathbf{z}_j[i] - \mathbf{z}_j[\sigma(i)]$ the above is equivalent to saying that there exist $\mathbf{h}_j$ such that $\sum_{j=1}^k \mathbf{h}_j = \mathbf{0}$ and for all $i \in [m]$ and for all $j \in [k]$ it is $\mathbf{h}_j[i] = \mathbf{z}_j[i] - \mathbf{z}_j[\sigma(i)]$, which is equivalent to having $\mathbf{h}_1, \ldots, \mathbf{h}_k$ such that $\sum_{j=1}^k \mathbf{h}_j = \mathbf{0}$ and $([m, \sigma], (\mathsf{com}(\mathbf{z}_j), \mathsf{com}(\mathbf{h}_j)), (\mathbf{z}_j, \mathbf{h}_j)) \in \mathcal{R}_{\mathcal{P}}^r$ for all $j = 1, \ldots, k$. □

5 Dynaverse: A $O(\sqrt{n}\log^2 n)$ dynamic zk-SNARK

In this section, we present Dynaverse, our first general-purpose dynamic zk-SNARK (i.e., for $\mathcal{R}_{\mathcal{C}}$ and index $\mathtt{i} = [n, n_0, \sigma]$). Dynaverse is using the Dynamo SNARK from Section 4 and leverages the decomposition lemma—Lemma 2. Dynaverse has $O(n)$ setup time, $O(k \cdot \sqrt{n}\log^2 n)$ update time (where k is the Hamming distance between $\mathtt{x}||\mathtt{w}$ and $\mathtt{x}'||\mathtt{w}'$) and $O(1)$ proof size. Dynaverse's update algorithm can be fully parallelized.

Background and problem with PLONK approach. Recall that for any fixed index $\mathtt{i} = [n, n_0, \sigma]$ describing a circuit $\mathcal{C}$, an instance of public inputs $\mathtt{x} \in \mathbb{F}^{n_0}$, and a witness $\mathtt{w} \in \mathbb{F}^{6n}$, we set $m = 6n + n_0$ and $\mathbf{z} = [\mathtt{w}; \mathtt{x}] \in \mathbb{F}^m$. We have $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathcal{C}}$ if and only if the following hold.

- *Copy constraint:* $([m, \sigma], \mathsf{com}(\mathbf{z}), \mathbf{z}) \in \mathcal{R}_{\mathcal{P}}$.
- *Gate constraint:* $\forall i \in [n], \mathbf{z}[i]+\mathbf{z}[n+i] = \mathbf{z}[2n+i], \mathbf{z}[3n+i]\cdot\mathbf{z}[4n+i] = \mathbf{z}[5n+i]$.

Let us focus on updatability of gate constraints and will come back to copy constraints later. Dynaverse will be using a similar technique with PLONK [16]. In particular one can write $\mathbf{z}$ as $[\mathbf{s}_1\ \mathbf{s}_2\ \mathbf{s}_3\ \mathbf{s}_4\ \mathbf{s}_5\ \mathbf{s}_6\ \mathbf{s}_7]$, where $\mathbf{s}_1$ holds the left inputs of all addition gates, $\mathbf{s}_2$ holds the right inputs of all addition gates, $\mathbf{s}_3$ holds the outputs of all addition gates, $\mathbf{s}_4$ holds the left inputs of all multiplication gates, $\mathbf{s}_5$ holds the right inputs of all multiplication gates, $\mathbf{s}_6$ holds the outputs of all multiplication gates, and $\mathbf{s}_7$ is of size n_0 and holds the public statement. Let $s_t(X)$ (for $t = 1, \ldots, 6$) be the polynomials that the prover uses to commit to those subvectors (We use θ to denote the n-th root of unity used in those Lagrange polynomials.) Clearly, to prove that the committed polynomials $s_t(X)$ satisfy the gate constraints we need to prove that $s_1(X) + s_2(X) = s_3(X)$ and $s_4(X) \cdot s_5(X) = s_6(X)$ for all $X = \theta, \ldots, \theta^n$.

Note that the addition constraint is easy to check due to the fact that $[s_t(X)]$'s are additively homomorphic. Similarly, the prover can update the commitments in $O(1)$ time when a value changes. However, the same does not hold for the multiplication constraint. In particular note that checking the multiplication constraint requires a zero-check for the polynomial $s_4(X)\cdot s_5(X) - s_6(X)$ on the set $\Theta = \{\theta, \ldots, \theta^n\}$ which can be done via a commitment to the polynomial

$$A(X) = \frac{s_4(X) \cdot s_5(X) - s_6(X)}{X^n - 1}\,. \tag{9}$$

However, as opposed to the addition constraint, if a single entry of say, $\mathbf{s}_4$, changes, the quotient polynomial $A(X)$ changes completely and must be recomputed from scratch. Unfortunately, this takes at least linear time.

Our main technique: Anchor statement. As we discussed in the introduction, the prover will compute a proof for an anchor statement $\mathbf{z}$ that proves that multiplication and addition gates are satisfied in $\mathbf{z}$ (The "gate" part of this proof includes the commitments to the polynomials $s_t(X)$ for $t \in [1, 6]$ and $A(X)$ as described above.) Our goal now is to compute a proof for an arbitrary statement

$\mathbf{z}'$ such that $\mathbf{z}' = \mathbf{z} + \mathbf{z}^*$, where $\mathbf{z}^*$ denotes the δ vector representing the update. For addition gates this is particularly easy, due to addition homomorphism.

For multiplication gates, let $I \subseteq [n]$ denote the set of indices of *all* the multiplication gates whose values differ from the initial witness. Define the polynomials $r'_4(X), r'_5(X), r'_6(X)$ of degree $|I| - 1$ where

$$r'_t(\theta^i) = \mathbf{z}'[(t-1)n + i], \text{ for all } t \in \{4, 5, 6\}, i \in I\,,$$

storing the *updated values* of the new witness vector. The prover then will provide a proof checking the multiplication gates in I via a zero-check for $r'_4(X) \cdot r'_5(X) - r'_6(X)$ on the set Θ^I, through a commitment to the quotient polynomial

$$A'(X) = \frac{r'_4(X) \cdot r'_5(X) - r'_6(X)}{\prod_{i \in I}(X - \theta^i)}\,.$$

Importantly, the prover can compute r'_4, r'_5, r'_6 in $O(|I| \log^2 |I|)$ time using fast interpolation [17], and $A'(X)$ in $O(|I| \log |I|)$ time using FFT (Note that for efficient verification, the prover also needs to provide the commitment to $A_I(X) = \prod_{i \in I}(X - \theta^i)$ and show its validity. This can be done through providing another commitment to $B_I(X) = (X^n - 1)/A_I(X)$.)

Now, by providing a commitment to the "anchor" quotient $A(X)$ and a commitment to $A'(X)$, the prover can prove that all multiplication gates in $\mathbf{z}$ are satisfied and all multiplication gates in $\mathbf{z}'$ *with respect to* I are also satisfied. The problem of stopping here is that that there might be other gates in $\mathbf{z}'$ that are *not* satisfied. To deal with this problem, the prover also needs to provide a commitment to $\mathbf{z}^*$ such that $\mathbf{z}' = \mathbf{z}^* + \mathbf{z}$ and a proof that all indices in $\mathbf{z}^*$ that are *not* in I are zero. This is done with a batch proof—see Eq. (10).

Copy constraints. The next task to be performed by the prover is to convince the verifier that $\mathbf{z}'$ is also consistent with σ. To do that, let $\mathbf{w} = [\mathbb{w}; \mathbf{0}]$, $\mathbf{w}^* = [\mathbb{w}' - \mathbb{w}; \mathbf{0}]$, and consider the following decomposition of $\mathbf{z}'$, i.e.,

$$\mathbf{z}' = [\mathbb{w}'; \mathbb{x}'] = [\mathbb{w}; \mathbf{0}] + [\mathbb{w}' - \mathbb{w}; \mathbf{0}] + [\mathbf{0}; \mathbb{x}'] = \mathbf{w} + \mathbf{w}^* + [\mathbf{0}; \mathbb{x}']\,.$$

Now, according to Lemma 2, we can do that by using a Dynamo proof for

$$([m, \sigma], (\mathsf{com}(\mathbf{w}), \mathsf{com}(\mathbf{h_w})), (\mathbf{w}, \mathbf{h_w})) \in \mathcal{R}^r_{\mathcal{P}} \text{ for some } \mathbf{h_w}\,,$$

and another Dynamo proof for

$$([m, \sigma], (\mathsf{com}(\mathbf{w}^*), \mathsf{com}(\mathbf{h}_{\mathbf{w}^*})), (\mathbf{w}^*, \mathbf{h}_{\mathbf{w}^*})) \in \mathcal{R}^r_{\mathcal{P}} \text{ for some } \mathbf{h}_{\mathbf{w}^*}\,,$$

as well as batch proofs $[q_{\mathbb{x}}], [q^*_{\mathbb{x}}]$ for $\mathsf{com}(\mathbf{w}), \mathsf{com}(\mathbf{w}^*)$ of 0s at $[6n + 1, 6n + n_0]$. Suppose $w(X), w^*(X), h_w(X), h_{w^*}(X)$ are polynomials encoding $\mathbf{w}, \mathbf{w}^*, \mathbf{h_w}, \mathbf{h}_{\mathbf{w}^*}$ respectively and we use $[w], [w^*], [h_w], [h_{w^*}]$ as their commitments.

Final step: Showing consistency between $[s_t], [r'_t]$ and $[w], [w^*]$. Note that the Dynamo SNARK provides relaxed permutation proofs for commitments $\mathsf{com}(\mathbf{w}) = [w]$ and $\mathsf{com}(\mathbf{w}^*) = [w^*]$, but checking the gate relationships works on commitments $[s_t]$ and $[r'_t]$. To show the relationship between them, the prover needs the following steps:

– The prover shows $(([w],[s_t]),(w,s_t))$ and $(([w^*],[s_t^*]),(w^*,s_t^*))$ are instances in $\mathcal{R}_{m,n,(t-1)n+1}$ where $\mathcal{R}_{m,n,k}$ is defined as

$$\mathcal{R}_{m,n,k} = \left\{ \begin{pmatrix} \mathbb{x} = ([a],[b]), \\ \mathbb{w} = (a(X), b(X)) \end{pmatrix} : \begin{array}{l} a(\omega^j) = b(\theta^{j-k+1}),\ \forall j \in [k, k+n-1] \\ \text{where } \omega, \theta \text{ are } m, n\text{-th roots of} \\ \text{unity respectively} \end{array} \right\}.$$

We present a protocol in Section D to show an argument for this relation.

– The prover then computes the commitments to the batch opening proofs of each $[s_t], [s_t^*]$ at I:

$$q_t(X) = \frac{s_t(X) - r_t(X)}{A_I(X)}, \quad q_t^*(X) = \frac{s_t^*(X) - r_t^*(X)}{A_I(X)}, \quad t \in \{4,5,6\}$$

so that for each $t \in \{4,5,6\}$, the verifier should be able to compute $[r_t'] = [r_t] \cdot [r_t^*]$. Furthermore, the prover computes the commitments to the batch opening proofs of zeros for each $[s_t^*]$ at $[n] \setminus I$:

$$\overline{q}_t^*(X) = \frac{s_t^*(X)}{B_I(X)}, \quad t \in \{4,5,6\} \tag{10}$$

The prover can compute all the commitments to such quotient polynomials in $O(|I| \log^2 |I|)$ time based on the discussion in Section A.

Proof and verification. The final proof consists of 2 Dynamo proofs, 12 consistency proofs and commitments to the polynomials mentioned above. To verify the final proof the verifier verifies the Dynamo proofs, the consistency proofs and the quotient polynomials $A, A', q_t, q_t^*, \overline{q}_t^*$. Then the verifier needs to verify consistency between the public statement $\mathbb{x}$ and the final proof. If we use the non-universal version of the Dynamo, then the verifier computes a commitment to the following polynomial corresponding to the public statement, i.e.,

$$h_{\mathbb{x}}(Y) = \sum_{i=6n+1}^{6n+n_0} (\mathbf{z}[i] - \mathbf{z}[\sigma^{-1}(i)]) \cdot L_i(Y) = \sum_{i=6n+1}^{6n+n_0} \mathbf{z}[i] \left(L_i(Y) - L_{\sigma^{-1}(i)}(Y)\right)$$

and checks whether $[h_{\mathbb{x}}(Y)] \cdot [h_w] \cdot [h_{w^*}] = 1_{\mathbb{G}}$, where $[h_w], [h_{w^*}]$ are in the public statements of the Dynamo instances for $\mathbf{w}, \mathbf{w}^*$. Our final protocol is in Fig. 4.

Efficient updates. We can now discuss the time to update the proof when $\mathbf{w}^*$ has k non-zeros. For gate constraints, $[s_t^*]$ and $[r_t'], [A']$ can be computed in $O(k \log^2 k)$ time. For copy constraints, the new Dynamo proof can be computed in $O(k)$ time. The proof for consistency check can also be computed in $O(k \log^2 k)$ time based on Sections 2 and D. Overall, the update time will be $O(k \log^2 k)$. To achieve amortized $\tilde{O}(\sqrt{n})$ update time, we can use $O(k \log^2 k)$ time to update the proof until $k \geq \sqrt{n}$. Then we recompute the whole proof (a new initial proof) in $O(n \log n)$ time. This can be de-amortized using standard techniques.

Theorem 4 (Dynaverse). *The protocol of Fig. 4 is a dynamic SNARK (per Definition 1) for $\mathcal{R}_{\mathcal{C}}$ based on q-DLOG (see Assumption 1) in the AGM model. Its complexities are as follows.*

1. *$\mathcal{G}$ runs in $O(n + n_0)$ time, outputs pk of $O(n)$ size, vk of $O(n_0)$ size;*
2. *$\mathcal{P}$ runs in $O(n \log n)$ time and outputs a proof π of $O(1)$ size;*
3. *$\mathcal{U}$ runs in $O(k\sqrt{n}\log^2 n)$ time, where k is the Hamming distance of $\mathbb{w}, \mathbb{w}'$;*
4. *$\mathcal{V}$ runs in $O(n_0)$ time.*

Just like Dynamo, Dynaverse can be made universal by using the random oracle and zero-knowledge using standard techniques. Please see Section J.

5.1 Aero: Our sparse zk-SNARK

Dynaverse naturally gives a way to build a sparse zk-SNARK protocol. More specifically, we can call the prover algorithm of Dynaverse with an all-zero anchor statement in the setup phase. When receiving the whole witness with sparsity k, the prover calls the update algorithm to get the updated proof within $O(k \log^2 k)$ time. As opposed to Dynaverse, Aero is not required to rebuild the proof since it is a zk-SNARK, not a dynamic zk-SNARK. We now have the following corollary.

Corollary 1 (Aero). *There is a sparse zk-SNARK (per Definitions 2 and 3) for $\mathcal{R}_{\mathcal{C}}$ based on q-DLOG (see Assumption 1) in the AGM model. Its complexities are as follows.*

1. *$\mathcal{G}$ runs in $O(n \log n + n_0)$ time and outputs public parameters of $O(n)$ size;*
2. *$\mathcal{P}$ runs in $O(k \log^2 k)$ time and outputs a proof π of $O(1)$ size, where k is the number of non-zero entries in $\mathbf{z}$;*
3. *$\mathcal{V}$ runs in $O(n_0)$ time.*

Similarly, we can make such construction zero-knowledge and universal. We omit the corresponding corollaries due to space limitations.

6 Dynalog: A $O(\log^3 n)$ dynamic zk-SNARK

In this section, we introduce Dynalog, a new dynamic zk-SNARK with polylogarithmic update time. We will only consider multiplication gates for notational convenience in this section, but it is easy to extend the idea for general case with both addition and multiplication gates. Recall that for any $\mathbb{i} = [n, n_0, \sigma]$ describing a circuit $\mathcal{C}$, public input $\mathbb{x} \in \mathbb{F}^{n_0}$, and witness $\mathbb{w} \in \mathbb{F}^{3n}$, let $\mathbf{z} = [\mathbb{w}; \mathbb{x}] \in \mathbb{F}^{3n+n_0}$. We have $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathcal{C}}$ if and only if $([3n + n_0, \sigma], \mathsf{com}(\mathbf{z}), \mathbf{z}) \in \mathcal{R}_{\mathcal{P}}$ and $\forall i \in [n]$, $\mathbf{z}[i] \cdot \mathbf{z}[n + i] = \mathbf{z}[2n + i]$. From now on we set $m = 3n + n_0$.

Core hierarchical data structure. Our approach is based on a standard "waterfall" hierarchical data structure that has also been used before in cryptography, e.g., [27,26,10]. Let us set $\ell = \log n$. Our data structure is as follows.

- We maintain $\ell + 1$ buffers $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(\ell)}$.

– $\mathcal{G}(1^\lambda, [n, n_0, \sigma]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - Set $m = 6n + n_0$. Pick random τ_X, τ_Y from $\mathbb{F}$ for X, Y respectively.
  - Call $\mathcal{D}.\mathcal{G}_{\tau_X,\tau_Y}(1^\lambda, [m, \sigma]) \to (\mathsf{pk}^{\mathcal{D}}, \mathsf{vk}^{\mathcal{D}})$.
  - Call $\mathcal{S}.\mathcal{G}_{\tau_X,\tau_Y}(1^\lambda, [m, n, (t-1)n+1]) \to (\mathsf{pk}_t^{\mathcal{S}}, \mathsf{vk}_t^{\mathcal{S}})$ for $t \in [1, 6]$.
  - Set $\mathsf{vk} = \{\mathsf{vk}^{\mathcal{D}}, \{\mathsf{vk}_t^{\mathcal{S}}\}_t, [L_i(Y)]_i, [X^n], [I_{\mathbb{x}}(X)]\}$ where $I_{\mathbb{x}}(X) = \prod_{i\in[6n+1,m]}(X - \omega^i)$ and $\mathsf{upk} = \mathsf{pk} = \{[X^i]_i, \mathsf{pk}^{\mathcal{D}}, \{\mathsf{pk}_t^{\mathcal{S}}\}_t\}$.

– $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{aux})$:
  - Parse $\mathbb{w}$ as $\{\mathbf{s}_t\}_{t=1,\ldots,6}$. Let $\mathbf{w} = [\mathbb{w}; \mathbf{0}]$. Compute $\mathbf{h_w}$ from $\mathbf{w}$.
  - Interpolate polynomials s_t, w, h_w from $\mathbf{s}_t, \mathbf{h}, \mathbf{h_w}$.
  - Call $\mathcal{D}.\mathcal{P}(\mathsf{pk}^{\mathcal{D}}, ([w], [h_w]), (\mathbf{w}, \mathbf{h_w})) \to \pi^{\mathcal{D}}$.
  - Compute $[q_{\mathbb{x}}]$ where $q_{\mathbb{x}}(X) = \frac{w(X)}{\prod_{i\in[6n+1,m]}(X-\omega^i)}$.
  - For $t \in [1, 6]$, call $\mathcal{S}.\mathcal{P}(\mathsf{pk}_t^{\mathcal{S}}, ([w], [s_t]), (w, s_t)) \to \pi_t^{\mathcal{S}}$.
  - Compute the commitment $[A(X)]$ as in Eq. (9).
  - For $t \in \{4, 5, 6\}, i \in [n]$, compute $[q_{t,i}(X)]$ as openings of $[s_t(X)]$ at $X = \theta^i$.
  - Set $\pi^{\mathsf{ini}} = (\{[s_t], \pi_t^{\mathcal{S}}\}_t, [w], [h_w], [A], \pi^{\mathcal{D}}, [q_{\mathbb{x}}])$, $\pi^{\mathsf{upd}} = \emptyset$, $\pi = (\pi^{\mathsf{ini}}, \pi^{\mathsf{upd}})$.
  - Set $\mathsf{aux} = (\mathbb{w}, \{[q_{i,t}]\}_{i\in[n],t\in\{4,5,6\}})$.

– $\mathcal{U}(\mathsf{upk}, \mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}, \pi, \mathsf{aux}) \to (\pi', \mathsf{aux}')$:
  - Parse $\pi^{\mathsf{ini}} = (\{[s_t], \pi_t^{\mathcal{S}}\}_t, [w], [h], [A], \pi^{\mathcal{D}})$ from π.
  - Parse $\mathsf{aux} = \{\mathbb{w}^{(0)}, \{[q_{i,t}^{(0)}]\}_{i\in[n],t\in\{4,5,6\}}\}$. Let $\mathbf{w}^* = [\mathbb{w}' - \mathbb{w}^{(0)}; \mathbf{0}]$.
  - If $\mathbf{w}^*$ has at least $\sqrt{n}$ non-zeros, return $(\pi, \mathsf{aux}) \leftarrow \mathcal{P}(\mathsf{pk}, \mathbb{x}', \mathbb{w}')$.
  - Derive r_4', r_5', r_6' from $\mathbb{w}'$ and compute $A'(X)$.
  - Calculate $I, \mathbf{s}_t^*, \mathbf{h}_{\mathbf{w}^*}$. Call $\mathcal{D}.\mathcal{P}(\mathsf{pk}^{\mathcal{D}}, ([w^*], [h_{w^*}]), (\mathbf{w}^*, \mathbf{h}_{\mathbf{w}^*})) \to (\pi^{\mathcal{D}})^*$.
  - Compute $[q_{\mathbb{x}}^*]$ where $q_{\mathbb{x}}^*(X) = \frac{w^*(X)}{\prod_{i\in[6n+1,m]}(X-\omega^i)}$.
  - For $t \in [1, 6]$, call $\mathcal{S}.\mathcal{P}(\mathsf{pk}_t^{\mathcal{S}}, ([w^*], [s_t^*]), (w^*, s_t^*)) \to (\pi_t^{\mathcal{S}})^*$.
  - For $t \in \{4, 5, 6\}$, compute r_t, r_t^* from $\mathbf{w}^{(0)}, \mathbf{w}^*$. Compute $[q_t], [q_t^*], [\overline{q}_t^*]$.
  - Set $\pi^{\mathsf{upd}} = (\{[s_t^*], (\pi_t^{\mathcal{S}})^*\}_{t\in[1,6]}, [w^*], [h_{w^*}], \{[r_t], [r_t^*], [q_t], [q_t^*], [\overline{q}_t^*]\}_{t\in\{4,5,6\}}, [A_I], [B_I], [A'], (\pi^{\mathcal{D}})^*, [q_{\mathbb{x}}^*])$.
  - Set $\pi' = (\pi^{\mathsf{ini}}, \pi^{\mathsf{upd}})$, $\mathsf{aux}' = \mathsf{aux}$.

– $\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$:
  - Check $[A_I]$ represent a valid set I: $e([A_I], [B_I]) = e([X^n - 1], g)$.
  - Check the addition constraints: $[s_1(X)] \cdot [s_2(X)] = [s_3(X)]$, $[s_1^*(X)] \cdot [s_2^*(X)] = [s_3^*(X)]$.
  - Check the multiplication constraints, i.e.,

$$e([s_4(X)], [s_5(X)]) \cdot e([-s_6(X)], g) = e([A(X)], [X^n - 1]),$$

$$e([r_4 + r_r^*], [r_5 + r_5^*]) \cdot e([-(r_6 + r_6^*)], g) = e([A'(X)], [A_I(X)]).$$

  - Check the batch proofs, i.e., $e([q_{\mathbb{x}}], [I_{\mathbb{x}}]) = e([w], g)$, $e([q_{\mathbb{x}}^*], [I_{\mathbb{x}}]) = e([w^*], g)$ and for $t \in \{4, 5, 6\}$, $e([q_t], [A_I]) = e([s_t - r_t], g)$, $e([q_t^*], [A_I]) = e([s_t^* - r_t^*], g)$, $e([\overline{q}_t^*], [B_I]) = e([s_t^*], g)$.
  - Check the Dynamo proofs $\pi^{\mathcal{D}}, (\pi^{\mathcal{D}})^*$ and consistency proofs $\pi_t^{\mathcal{S}}, (\pi_t^{\mathcal{S}})^*$.
  - Compute $h_{\mathbb{x}}(Y)$. Check $[h_{\mathbb{x}}(Y)] \cdot [h_w] \cdot [h_{w^*}] = 1_{\mathbb{G}}$.

Fig. 4. Dynaverse dynamic SNARK using Dynamo SNARK $\mathcal{D}$ and SNARK $\mathcal{S}$ for consistency check (Section D) as a black box. The red part is removed when used to build Aero, our sparse zk-SNARK.

- Every buffer $\mathsf{buf}^{(i)}$ has at most 2^i slots and is represented by a vector $\mathbf{z}_i \in \mathbb{F}^m$ of at most $3 \cdot 2^i$ non-zero entries.
- Every buffer stores "gate update" information: For instance, if, for gate j, the left input is increased by a and the output is increased by b, while the right input remains the same as before, then we store $(j, a, 0, b)$ to some $\mathsf{buf}^{(k)}$. This maps to a vector $\mathbf{z}_k$ such that $\mathbf{z}_k[j] = a$, $\mathbf{z}_k[n+j] = 0$ and $\mathbf{z}_k[2n+j] = b$ and the rest of the entries are 0.
- Witness vector $\mathbf{z}$ will be distributed among buffers such that $\mathbf{z} = \sum_{i=0}^{\ell} \mathbf{z}_i$.
- We denote with $S^{(k)}$ the non-zero entries of $\mathbf{z}_k$.

Final component of the data structure: The AMT tree. The final component of our core data structure is the AMT tree $tree_i$ [37]. In particular, for every level i of the data structure, we maintain a KZG commitment $\mathsf{com}(\mathbf{z}_i)$ along with an AMT tree for $\mathbf{z}_i$ as described in the Section E.

1. $\mathsf{AMT.Setup}(1^\lambda, m) \to (pk, vk)$.
2. $\mathsf{AMT.Commit}(pk, \mathbf{z}) \to \mathsf{com}(\mathbf{z})$.
3. $\mathsf{AMT.Prove}(pk, \mathbf{z}, i) \to \pi_i$.
4. $\mathsf{AMT.Verify}(vk, \mathsf{com}(\mathbf{z}), (i, \mathbf{z}[i]), \pi) \to \{0, 1\}$

for efficiently maintaining vector commitment proofs, equipped with the following additional algorithms.

1. An algorithm $\mathsf{AMT.ComputeAllProofs}(pk, \mathbf{z}) \to \{\pi_1, \ldots, \pi_m, tree\}$ that runs in $\tilde{O}(m')$ time where m' is the number of non-zero elements of $\mathbf{z}$. Along with the proofs, this algorithm outputs a data structure $tree$.
2. An algorithm $\mathsf{AMT.ProveZeroSet}(pk, \mathbf{z}, tree) \to \{S, \pi_S\}$ that runs in $\tilde{O}(m')$ time where m' is the number of non-zero elements of $\mathbf{z}$. This algorithm outputs a proof π_S for the set of indices S that map to zero values in vectors $\mathbf{z}$. Note that this algorithm has a respective verification algorithm $\mathsf{AMT.VerifyZeroSet}(vk, \mathsf{com}(\mathbf{z}), S, \pi_S) \to \{0, 1\}$.

Finally, the AMT vector commitment scheme is *proof homomorphic*, in the sense that not only you can add commitments to retrieve the commitment to the sum of the vectors, but also you can add proofs across *different* vectors but for the *same* indices: Given vectors $\mathbf{z}$ and $\mathbf{z}'$, an index i and proofs π_i and π'_i, the proof for index i for $\mathbf{z}+\mathbf{z}'$ can be computed as $\pi_i \cdot \pi'_i$. See Fig. 14 for a simple illustration of our core data structure.

Permutation check for z in the hierarchical data structure. Given $\mathbf{z}$ is distributed in the buffers of our hierarchical data structure as described above, there is a natural way to check that it satisfies the permutation σ using our Dynamo SNARK for a relaxed permutation relation from Section 4. In particular since $\mathbf{z} \in \mathbb{F}^n$ can be expressed as $\sum_{i=0}^{\ell} \mathbf{z}_i$ where $\mathbf{z}_i \in \mathbb{F}^n$ as well, we can use the decomposition lemma (Lemma 2). Therefore, for every buffer $\mathsf{buf}^{(i)}$, we provide a proof $\pi\mathsf{perm}^{(i)}$ that proves the statement $\mathsf{com}(\mathbf{z}_i), \mathsf{com}(\mathbf{h}_i)$ with respect to σ. Then, to verify that $\mathbf{z}$ satisfies σ, all proofs $\pi\mathsf{perm}^{(i)}$ for $i = 0, \ldots, \ell$ must verify. In addition, it needs to be the case that $\prod_{i=0}^{\ell} \mathsf{com}(\mathbf{h}_i) = 1_{\mathbb{G}}$.

Gate check for z in the hierarchical data structure, Perhaps the most challenging part of our approach is to perform the gate check: Given $\mathbf{z}$ is distributed in the buffers of our hierarchical data structure as described above, there are various ways to check the real values of $\mathbf{z}$ satisfy the gate constraints. For example, one can just use a *global check*, where all buffers ($\mathbf{z}_i$'s) are combined ("added") together, and then the gate check can be performed on the derived $\mathbf{z}$. However, for Dynalog, we use a *prefix check*, that is amenable to efficiently handling updates. In particular, we proceed in the following $\ell + 1$ steps.

1. check_0: Compute $\mathbf{z} = \sum_{i=0}^{\ell} \mathbf{z}_i$. Then perform the gate check on $\mathbf{z}$ only for the gates $j \in S^{(0)}$, i.e., check that $\mathbf{z}[j] \cdot \mathbf{z}[n+j] = \mathbf{z}[2n+j]$ for all $j \in S^{(0)}$.
2. check_1: Compute $\mathbf{z} = \sum_{i=1}^{\ell} \mathbf{z}_i$. Then perform the gate check on $\mathbf{z}$ only for the gates $j \in S^{(1)}$, i.e., check that $\mathbf{z}[j] \cdot \mathbf{z}[n+j] = \mathbf{z}[2n+j]$ for all $j \in S^{(1)}$.
3. ...
4. $\mathsf{check}_{\ell-1}$: Compute $\mathbf{z} = \sum_{i=\ell-1}^{\ell} \mathbf{z}_i$. Then perform the gate check on $\mathbf{z}$ only for the gates $j \in S^{(\ell-1)}$, i.e., check that $\mathbf{z}[j] \cdot \mathbf{z}[n+j] = \mathbf{z}[2n+j]$ for all $j \in S^{(\ell-1)}$.
5. check_ℓ: Compute $\mathbf{z} = \mathbf{z}_\ell$. Then perform the gate check on $\mathbf{z}$ only for the gates $j \in S^{(\ell)}$, i.e., check that $\mathbf{z}[j] \cdot \mathbf{z}[n+j] = \mathbf{z}[2n+j]$ for all $j \in S^{(\ell)}$.

Note that, check_0 is not a global check, since it combines *only* the indices in $S^{(0)}$ across all buffers, and not *all* indices in $[n]$. Now we have the following lemma showing that it is enough for us to perform the prefix check to check all gates constraints. The proof can be found in Section I.

Lemma 3 (Prefix and global check). *Let* $\mathbf{z}$ *be distributed in the hierarchical data structure* $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(\ell)}$*. We now have the following.*

1. *If a prefix check succeeds on* $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(\ell)}$*, then a global check succeeds also.*
2. *Assume for any* $i < k$*, all updates in* $\mathsf{buf}^{(i)}$ *occur after those in* $\mathsf{buf}^{(k)}$*. Then, if a global check succeeds on* $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(\ell)}$*, a prefix check succeeds also.*

Based off the above lemma, we will be maintaining a proof $\pi\mathsf{gate}^{(i)}$ for each $\mathsf{buf}^{(i)}$ that will be proving the prefix check. Informally speaking, the public statement of this proof will contain $(\mathsf{com}(\mathbf{z}_i), \ldots, \mathsf{com}(\mathbf{z}_\ell))$ such that for *all* indices j in $S^{(i)}$, the gate constraints hold for the combination of buffers $\mathsf{buf}^{(i)}, \ldots, \mathsf{buf}^{(\ell)}$.

Updating the hierarchical data structure efficiently. We now describe how we can use our data structure to handle updates. First, we initialize our data structure by storing the initial values of the witness $\mathbf{z}$ in $\mathsf{buf}^{(\ell)}$, while all other buffers are kept empty.

Without loss of generality, we will consider a single gate update u that produces a new satisfying witness $\mathbb{w}'$ (In the general case more gates might have to change to produce a new satisfying witness but it is enough to consider a single update for our analysis.) Let $\mathsf{buf}^{(k)}$ be the first empty buffer from level 0 onwards. Collect all the non-empty slots from $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(k-1)}$, and, together with the new update, store all of them into $\mathsf{buf}^{(k)}$, and then empty $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(k-1)}$.

- Public Input: Commitments $c_i, \ldots, c_\ell$.
- Witness 1: Indices $S^{(i)}$ of non-zeros in $\mathbf{z}_i$, where $\mathbf{z}_i$ is the vector committed to by c_i.
  Witness 2: The AMT proof $\pi_{[m]\setminus S^{(i)}}$ for $[m] \setminus S^{(i)}$.
  Witness 3: Values $\{\mathbf{z}[j] : j \in S^{(i)}\}$ where $\mathbf{z}$ is the vector committed to by $c_i \cdot \ldots \cdot c_\ell$.
  Witness 4: The AMT proofs $\{\pi_j : j \in S^{(i)}\}$ for $\{\mathbf{z}[j] : j \in S^{(i)}\}$.
- Computation:
  - Check $\mathsf{AMT.VerifyZeroSet}(vk, c_i, [m] \setminus S^{(i)}, \pi_{[m]\setminus S^{(i)}})$.
  - For all $j \in S^{(i)}$, check $\mathsf{AMT.Verify}(vk, c_i \cdot \ldots \cdot c_\ell, (j, \mathbf{z}[j]), \pi_j)$.
  - Let S contain all indices $j \in [n]$ such that at least one of j, $n+j$, $2n+j$ is in $S^{(i)}$.
  - For all indices $j \in S$ check $\mathbf{z}[j] \cdot \mathbf{z}[n+j] = \mathbf{z}[2n+j]$.

Fig. 5. Circuit $\mathcal{C}\mathsf{gate}^{(i)}$.

Note that when combining the slots from $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(k-1)}$, there might be multiple slots for the same gate. We can just combine them into one slot by adding their values so that there is at most one slot for each gate in $\mathsf{buf}^{(k)}$. Finally, while updating the buffers, we also update the respective vectors $\mathbf{z}_i$. At the end of this procedure, let B be the set of new buffers that have been created. For every new buffer $\mathsf{buf}^{(i)}$ in B, we must compute the new proofs for $\pi\mathsf{perm}^{(i)}$ and $\pi\mathsf{gate}^{(i)}$.

Update time complexity. Updating the buffers as described above can be performed in $O(\log n)$ amortized time since every buffer $\mathsf{buf}^{(i)}$ is guaranteed to have at most 2^i slots (Standard techniques can be used to deamortize this procedure as well.) Updating the AMT tree $tree_i$ can be done in $\tilde{O}(2^i)$ time by running $\mathsf{AMT.ComputeAllProofs}$. What remains to show to now is how to efficiently compute $\pi\mathsf{perm}^{(i)}$ and $\pi\mathsf{gate}^{(i)}$ for all the new buffers in B. In particular as long as we can compute $\pi\mathsf{perm}^{(i)}$ and $\pi\mathsf{gate}^{(i)}$ in $\tilde{O}(2^i)$ time, then the overall update time will be polylogarithmic, as desired.

Computing $\pi\mathsf{perm}^{(i)}$ in $O(2^i)$ time. Given a vector $z_i \in \mathbb{F}^m$, it is easy to see that $\pi\mathsf{perm}^{(i)}$ can be computed in $O(2^i)$ time—independent of m. This is because $\mathbf{z}_i$ has at most 2^i entries non-zero entries, and, by Dynamo Theorem 3, the relaxed permutation proof can be computed in $O(m')$ time, where m' is the number of non-zero vector entries. Computing $\pi\mathsf{gate}^{(i)}$ in time $\tilde{O}(2^i)$ is more complex and we analyze this part in detail in the following.

Computing $\pi\mathsf{gate}^{(i)}$ in $\tilde{O}(2^i)$ time. We now provide the details on how to compute proof $\pi\mathsf{gate}^{(i)}$ in $\tilde{O}(2^i)$ time. Recall that $\pi\mathsf{gate}^{(i)}$ must prove the i-th part of the prefix check which is the following.

"*On input commitments $c_i, \ldots, c_\ell$, let $S^{(i)}$ be the set of non-zero indices in the vector $\mathbf{z}_i$ committed to by c_i. Then check that the multiplication relation holds for the gate indices in $S^{(i)}$ in the vector committed to by $c_i \cdot \ldots \cdot c_\ell$.*"

Note that one could use a zk-SNARK directly to prove the relation above. This however would not achieve our efficiency goal since the witnesses are vectors of

`ComputeGateProof`(i)

1. Run $\mathsf{AMT.ProveZeroSet}(pk, \mathbf{z}_i, tree_i) \to \{[m] \setminus S^{(i)}, \pi_{[m]\setminus S^{(i)}}\}$.
2. For all $k = i+1$ to ℓ do the following.
   (a) Using $tree_k$, retrieve the AMT proofs $\{p_j : j \in S^{(i)}\}$ for $\{\mathbf{z}_k[j] : j \in S^{(i)}\}$.
   (b) Set $\pi_j = \pi_j \cdot p_j$ and $\mathbf{z}[j] = \mathbf{z}[j] + \mathbf{z}_k[j]$ for all $j \in S^{(i)}$.
3. Return a proof using IPAs (see Section F.2) for $\mathcal{C}\mathsf{gate}^{(i)}$ of Fig. 5 with public input $\mathsf{com}(\mathbf{z}_i), \ldots, \mathsf{com}(\mathbf{z}_\ell)$ and witnesses $S^{(i)}$, AMT proof $\pi_{S^{(i)}}$, values $\{\mathbf{z}[j] : j \in S^{(i)}\}$ and AMT proofs $\{\pi_j : j \in S^{(i)}\}$.

Fig. 6. Computing the proof $\pi\mathsf{gate}^{(i)}$ in $\tilde{O}(2^i)$ time.

size $m = 3n + n_0$ and therefore the prover will run in $O(m)$ time, instead of $\tilde{O}(2^i)$ time. Instead, we are using the AMT tree data structure to achieve our goal. The algorithm is described in Fig. 6. The main insight is to leverage the AMT data structure to compute the witnesses required for Fig. 5 in $\tilde{O}(2^i)$ time, so that we can use this information to build IPA instances (see Section F.2). Because of that Algorithm `ComputeGateProof` from Fig. 6 runs in time $\tilde{O}(2^i)$, as desired. This is because all complexities are proportional to $|S^{(i)}|$ which is $O(2^i)$. The complete algorithms of Dynalog are given in Fig. 15 (see Section K).

Lemma 4. *Algorithm* `ComputeGateProof`(i) *from Fig. 6 runs in* $\tilde{O}(2^i)$ *time.*

Theorem 5 (Dynalog)**.** *The protocol of Fig. 15 is a dynamic SNARK (per Definition 1) for* $\mathcal{R}_\mathcal{C}$ *based on q-DLOG (see Assumption 1), q-ASDBP (see Assumption 2), the ROM and the AGM. Its complexities are as follows.*

1. $\mathcal{G}$ *runs in* $O(n \log^3 n)$ *time, outputs* pk *of* $O(n \log^3 n)$ *size,* vk *of* $O(\log^2 n)$ *size;*
2. $\mathcal{P}$ *runs in* $O(n \log^2 n)$ *time and outputs a proof* π *of* $O(\log^3 n)$ *size;*
3. $\mathcal{U}$ *runs in amortized* $O(k \log^3 n)$ *time, where* k *is the Hamming distance of* $\mathbb{w}, \mathbb{w}'$*;*
4. $\mathcal{V}$ *runs in* $O(n_0 + \log^3 n)$ *time.*

Based on the universal Dynamo protocol (see Section 4), we can also derive a universal version of the Dynalog protocol. Based on the zero-knowledge Dynamo protocol (see Sections 4 and J) and the zero-knowledge property of IPAs (Theorem 12), we can derive a zero-knowledge version of the Dynalog protocol. We omit the statements due to space limitations.

De-amortization of update algorithm. We note that the update algorithm in Fig. 15 can be de-amortized to achieve $O(\log^3 n)$ update time in the worse case. This can be done via standard techniques, i.e., through decomposition of the prove algorithm for $\mathcal{C}\mathsf{gate}^{(i)}$ into 2^i steps. The data structures for buffers $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(i-1)}$ will be kept until the computation for $\pi\mathsf{gate}^{(i)}$ finishes.

7 From dynamic zk-SNARKs to recursion-free BIVC

In this section we show how to build a recursion-free BIVC scheme $(\mathcal{G}, \mathcal{P}, \mathcal{U})$ from a dynamic zk-SNARK. The definition of BIVC is given in the Appendix

– $\mathcal{G}(1^\lambda, \mathsf{F}, N) \to (\mathsf{pk}, \mathsf{vk})$.
  1. Let F_N be the N-sized circuit wrt to F from Figure 16;
  2. Let $\mathbb{i} = (\mathbb{x}, \mathbb{w})$ be the NP relation of F_N, i.e., $\mathbb{x} = (i = [b_1, \ldots, b_N], z_i, z_0)$ and $\mathbb{w} = (w_0, z_1, w_1, \ldots, z_{N-1}, w_{N-1}, z_N, w\mathsf{F})$;
  3. Run $\mathsf{DS}.\mathcal{G}(1^\lambda, \mathbb{i}) \to (\mathsf{DS.pk}, \mathsf{DS.upk}, \mathsf{DS.vk})$;
  4. Set $\mathbb{x} = ([0 \ldots 0], 0, z_0)$, $\mathbb{w} = (0, (\mathsf{F}(0,0), 0) \ldots, (\mathsf{F}(0,0), 0), \mathsf{F}(0,0), w\mathsf{F})$;
  5. Run $\mathsf{DS}.\mathcal{P}(\mathsf{DS.pk}, \mathbb{x}, \mathbb{w}) \to (\Pi_0, \mathsf{aux}_0)$;
  6. Output $\mathsf{DS.pk}$, $\mathsf{DS.upk}$ and (Π_0, aux_0) as pk and $\mathsf{DS.vk}$ as vk.
– $\mathcal{P}(i, z_0, \pi_{i-1}, z_{i-1}, w_{i-1}, z_i, \mathsf{pk}) \to \pi_i$.
  1. Parse pk as $\mathsf{DS.pk}$, $\mathsf{DS.upk}$ and (Π_0, aux_0);
  2. Let $\pi_{i-1} = (\Pi_{i-1}, \mathsf{aux}_{i-1})$ be a valid proof for $\mathbb{x} = ([b_1, \ldots, b_N], z_{i-1}, z_0)$, where $[b_1, \ldots, b_N]$ is the unary representation of $i-1$. Let also
  $$\mathbb{w} = (w_0, z_1, w_1, \ldots, z_{N-1}, w_{N-1}, z_N, w\mathsf{F})$$
  be the corresponding witness (For $i = 1$, $\pi_{i-1} = (\Pi_0, \mathsf{aux}_0)$ and can be retrieved from pk.)
  3. Consider the statement $\mathbb{x}' = ([b_1, \ldots, b_{i-1}, 1, b_{i+1}, \ldots, b_N], z_i, z_0)$ which differs only in the counter's bit i from $\mathbb{x}$ as well as in z_i and z_{i-1}, in that $z_i = \mathsf{F}(z_{i-1}, w_i)$;
  4. Let $\mathbb{w}'$ be the same as $\mathbb{w}$ with the only difference being $z_i = \mathsf{F}(z_{i-1}, w_{i-1})$ and w_i are now set, as opposed to $\mathsf{F}(0,0)$ and 0 respectively. Also note, due to z_i being computed with $\mathsf{SUMTREE}$, only a logarithmic number of additional circuit wires change.
  5. Run $\mathsf{DS}.\mathcal{U}(\mathsf{DS.upk}, \mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}, \Pi_{i-1}, \mathsf{aux}_{i-1}) \to (\Pi_i, \mathsf{aux}_i)$;
  6. Output π_i as (Π_i, aux_i).
– $\mathcal{V}(\mathsf{vk}, (i, z_i, z_0), \pi_i) \to 0/1$:
  1. Parse vk as $\mathsf{DS.vk}$ and π_i as (Π_i, aux_i);
  2. Output $\mathsf{DS}.\mathcal{V}(\mathsf{DS.vk}, (i, z_i, z_0), \Pi_i)$.

Fig. 7. Constructing a BIVC scheme from a dynamic zk-SNARK.

(see Definition 4). Our main idea is the following: We represent the BIVC computation with an $(N \cdot \mathsf{F})$-sized circuit F_N (see Figure 16) whose public input is (i, z_0, z_i)—the same with the public input of a plain IVC scheme. However, the circuit is constructed so that when the public statement changes from $(i-1, z_0, z_{i-1})$ to $(i+1, z_0, \mathsf{F}(z_{i-1}, w_{i-1}))$ only $|\mathsf{F}|$ as well as a logarithmic number of wires change. We will use a dynamic zk-SNARK DS, as in Definition 1, to build an IVC scheme I as in Definition 4. Recall the dynamic zk-SNARK API:

– $\mathsf{DS}.\mathcal{G}(1^\lambda, \mathbb{i}) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$;
– $\mathsf{DS}.\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{aux})$;
– $\mathsf{DS}.\mathcal{U}(\mathsf{upk}, \mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}, \pi, \mathsf{aux}) \to (\pi', \mathsf{aux}')$;
– $\mathsf{DS}.\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$.

The algorithms of our BIVC scheme are provided in Fig. 7.

Dealing with linear-size public statement. Due to the fact the the counter i must be passed in unary (to ensure a few wires change between neighboring statement), the public statement has linear size. We can deal with this in the

following way: Instead of exposing all N bits of the counter as a public statement, we can expose an easily-updatable hash of the sequence (e.g., MU-HASH [2]). The hash then will have to be computed using a binary tree inside F_N.

Theorem 6 (Recursion-free BIVC from dynamic zk-SNARKs). *The protocol of Figure 7 is a recursion-free BIVC scheme (per Definition 4) for N iterations of function F, assuming a dynamic zk-SNARK DS with algorithms $(\mathcal{G},\mathcal{P},\mathcal{U},\mathcal{V})$ (per Definition 1). Its complexities are as follows.*

1. *$\mathcal{G}$ runs in time $|\mathsf{DS}.\mathcal{G}|+|\mathsf{DS}.\mathcal{P}|$ where $\mathsf{DS}.\mathcal{G}$ runs on an index $\mathtt{i}$ defined by the $N\cdot|\mathsf{F}|$-sized circuit F_N from Figure 2. It outputs pk of size $|\mathsf{DS.pk}|+|\mathsf{DS.upk}|$ and vk of size and $|\mathsf{DS.vk}|$;*
2. *$\mathcal{P}$ runs in $|\mathsf{DS}.\mathcal{U}(|\mathsf{F}|)|$ time, outputs a proof π of $|\mathsf{DS}.\pi|+|\mathsf{DS.aux}|$ size, where $|\mathsf{DS}.\mathcal{U}(|\mathsf{F}|)|$ denotes the time complexity of $\mathsf{DS}.\mathcal{U}$ when the Hamming distance of the source and target statements is $|\mathsf{F}|$;*
3. *$\mathcal{V}$ runs in $|\mathsf{DS}.\mathcal{V}|$ time.*

Data structure size of our BIVC scheme. Our BIVC scheme uses the state aux (that is required by the underlying dynamic SNARK to perform the next update) as state. When Dynavold is used to implement the BIVC scheme, this state or data structure, for the case of the sequential updates of F_N is *sublinear* (This is not the case in general, for example, when arbitrary updates must be supported or when we use Dynalog, even with sequential updates.) In particular, recall that Dynavold, for proving multiplication constraints, bucketizes the witness in $\sqrt{n}$-sized buckets. Because updates in BIVC are monotonically increasing, only the bucket needs to be appended as state, which keeps the data structure sublinear. We now have the following Theorem (See Section I for its proof.)

Lemma 5 (Sublinear-state, recursion-free BIVC from Dynavold). *There exists a recursion-free BIVC scheme (per Definition 4) for N iterations of function F, assuming q-DLOG (see Assumption 1) in the AGM. Its complexities are as follows, where $M = N|\mathsf{F}|$.*

1. *$\mathcal{G}$ runs in time $O(M)$ and outputs $O(M)$-size pk and $O(\sqrt{M})$-size vk;*
2. *$\mathcal{P}$ runs in $O(\sqrt{M}\log M)$ time, outputs a proof π of $O(\sqrt{M})$ size;*
3. *$\mathcal{V}$ runs in $O(\sqrt{M})$ time.*

7.1 The Dynavold dynamic zk-SNARK

Briefly speaking, Dynavold is a dynamic zk-SNARK construction which uses sub-vectors to address the bottleneck of multiplication gates we have in Dynaverse and Dynalog. In particular, we decompose the witness vector $\mathbf{z}$ into $\sqrt{n}$ subvectors $\mathbf{z}_1,\ldots,\mathbf{z}_{\sqrt{n}}$ of size $O(\sqrt{n})$, each containing the wires for $O(\sqrt{n})$ gates. For each sub-vector, we maintain a constant-size proof for gate constraints and a Dynamo proof for copy constraints (using the decomposition of permutation relations, see

Lemma 2). For a single update appearing in $\mathbf{z}_j$, we only need to update its gate proof and Dynamo proof in $\tilde{O}(\sqrt{n})$ time. The proof will have $O(\sqrt{n})$ size.

One interesting finding for Dynavold is that it can be plugged in to our BIVC construction to achieve sub-linear data structure size. More specifically, note that in our BIVC circuit, the wires will be updated in a monotonic order and each sub-circuit of F computation will only be updated once. Therefore, we can let the first sub-vector contain the first $\sqrt{N \cdot |\mathsf{F}|}$ wires, and the second contain the next $\sqrt{N \cdot |\mathsf{F}|}$ wires, and so on. At any time, we only need to maintain the data structure for one single sub-vector (where the wires of the current iteration lie in) and discard the data structure for sub-vectors of previous F iterations. See the details of Dynavold in Section H and Fig. 17 for a figure illustration. See Section J for how to make this sub-linear data structure zero-knowledge.

References

1. Ananth, P., Deshpande, A., Kalai, Y.T., Lysyanskaya, A.: Fully homomorphic NIZK and NIWI proofs. In: Hofheinz, D., Rosen, A. (eds.) Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II. vol. 11892, pp. 356–385 (2019)
2. Bellare, M., Micciancio, D.: A new paradigm for collision-free hashing: Incrementality at reduced cost. In: Fumy, W. (ed.) Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding. vol. 1233, pp. 163–192 (1997)
3. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for r1cs. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019. pp. 103–128. Springer International Publishing, Cham (2019)
4. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for snarks and proof-carrying data. In: Recursive composition and bootstrapping for SNARKS and proof-carrying data. p. 111–120. New York, NY, USA (2013)
5. Bünz, B., Maller, M., Mishra, P., Tyagi, N., Vesely, P.: Proofs for inner pairing products and applications. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021. pp. 65–97. Cham (2021)
6. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 315–334 (2018)
7. Campanelli, M., Fiore, D., Han, S., Kim, J., Kolonelos, D., Oh, H.: Succinct zero-knowledge batch proofs for set accumulators. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022 (2022)
8. Campanelli, M., Nitulescu, A., Ràfols, C., Zacharakis, A., Zapico, A.: Linear-map vector commitments and their practical applications. In: Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part IV (2022)

9. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Kurosawa, K., Hanaoka, G. (eds.) Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings (2013)
10. Chandran, N., Kanukurthi, B., Ostrovsky, R.: Locally updatable and locally decodable codes. In: Lindell, Y. (ed.) Theory of Cryptography. pp. 489–514. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
11. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable proof systems and applications. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings. vol. 7237, pp. 281–300 (2012)
12. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 499–530. Cham (2023)
13. Chen, M., Chiesa, A., Gur, T., O'Connor, J., Spooner, N.: Proof-carrying data from arithmetized random oracles. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14005, pp. 379–404. Springer (2023)
14. Chiesa, A., Guan, Z., Samocha, S., Yogev, E.: Security bounds for proof-carrying data from straightline extractors. In: Boyle, E., Mahmoody, M. (eds.) Theory of Cryptography - 22nd International Conference, TCC 2024, Milan, Italy, December 2-6, 2024, Proceedings, Part II. vol. 15365, pp. 464–496 (2024)
15. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018. pp. 33–62. Cham (2018)
16. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953 (2019)
17. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, 3 edn. (2013)
18. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: Sedgewick, R. (ed.) Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA. pp. 291–304. ACM (1985)
19. Groth, J.: On the size of pairing-based non-interactive arguments. In: Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II. pp. 305–326 (2016)
20. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-Size Commitments to Polynomials and Their Applications. In: ASIACRYPT'10 (2010)
21. Labs, L.: Lagrange prover network, https://app.lagrange.dev/zk-coprocessor/explore/dashboard
22. Lee, H., Seo, J.H.: On the security of nova recursive proof system. IACR Cryptol. ePrint Arch. p. 232 (2024)
23. Luo, Z., Jia, Y., Ospina Gracia, A.V., Kate, A.: Cauchyproofs: Batch-updatable vector commitment with easy aggregation and application to stateless blockchains. In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 1947–1963 (2025)

24. Miltersen, P.B., Subramanian, S., Vitter, J.S., Tamassia, R.: Complexity models for incremental computation. Theor. Comput. Sci. **130**(1), 203–236 (Aug 1994)
25. Nguyen, W., Datta, T., Chen, B., Tyagi, N., Boneh, D.: Mangrove: A scalable framework for folding-based snarks. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024. pp. 308–344. Cham (2024)
26. Ostrovsky, R.: Efficient computation on oblivious rams. In: Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing. p. 514–523. STOC '90, Association for Computing Machinery, New York, NY, USA (1990)
27. Ostrovsky, R.: An efficient software protection scheme. In: Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. Lecture Notes in Computer Science, vol. 435, pp. 610–611. Springer (1989)
28. Papamanthou, C., Srinivasan, S., Gailly, N., Hishon-Rezaizadeh, I., Salumets, A., Golemac, S.: Reckle Trees: Updatable Merkle Batch Proofs with Applications. In: Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (2024)
29. Rosenberg, M., Mopuri, T., Hafezi, H., Miers, I., Mishra, P.: Hekaton: Horizontally-scalable zksnarks via proof aggregation. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. p. 929–940. CCS '24, Association for Computing Machinery, New York, NY, USA (2024)
30. Sefranek, M.: How (not) to simulate PLONK. Cryptology ePrint Archive, Paper 2024/848 (2024)
31. Setty, S.: Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In: Advances in Cryptology – CRYPTO 2020 (2020)
32. Setty, S., Thaler, J.: Twist and Shout: Faster memory checking arguments via one-hot addressing and increments. Cryptology ePrint Archive, Paper 2025/105 (2025)
33. Setty, S., Thaler, J., Wahby, R.: Unlocking the Lookup Singularity with Lasso. In: Advances in Cryptology – EUROCRYPT 2024 (2024)
34. Srinivasan, S., Chepurnoy, A., Papamanthou, C., Tomescu, A., Zhang, Y.: Hyperproofs: Aggregating and maintaining proofs in vector commitments. In: 31st USENIX Security Symposium (USENIX Security 22). Boston, MA (Aug 2022)
35. Tamassia, R.: Authenticated data structures. In: Di Battista, G., Zwick, U. (eds.) Algorithms - ESA 2003. pp. 2–5. Berlin, Heidelberg (2003)
36. Team, P.Z.: Plonky 2: Improved Plonk with Fast Verification and Universal SRS (2022), https://github.com/mir-protocol/plonky2/blob/main/plonky2/plonky2.pdf
37. Tomescu, A.: How to Keep a Secret and Share a Public Key (Using Polynomial Commitments). Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, USA (2020)
38. Tomescu, A., Abraham, I., Buterin, V., Drake, J., Feist, D., Khovratovich, D.: Aggregatable Subvector Commitments for Stateless Cryptocurrencies. In: Galdi, C., Kolesnikov, V. (eds.) Security and Cryptography for Networks. pp. 45–64. Cham (2020)
39. Tyagi, N., Fisch, B., Zitek, A., Bonneau, J., Tessaro, S.: Versa: Verifiable registries with efficient client audits from rsa authenticated dictionaries. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 2793–2807. CCS '22 (2022)
40. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) Theory of Cryptography, Fifth The-

ory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. vol. 4948, pp. 1–18 (2008)
41. Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. In: Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV (2022)

A Additional preliminaries

Algebraic group model and assumptions. For our security analysis, we will use the *algebraic group model* from [15]. In our protocols, by an *algebraic adversary* $\mathcal{A}$ we refer to a PPT algorithm which satisfies the following: Given lists of initial group elements $\mathbf{L} \in \mathbb{G}^n$, whenever $\mathcal{A}$ outputs a group element $g \in \mathbb{G}$, it also outputs a vector $\mathbf{g} \in \mathbb{F}^n$ such that $g = \prod_{j\in[n]} \mathbf{L}[j]^{\mathbf{g}[j]}$. Finally, as in [15,16], our security also rests on the q-DLOG assumption, which we present in the following.

Assumption 1 (q-DLOG) *For any PPT adversary $\mathcal{A}$, given $\mathsf{pp}_{\mathsf{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$ and $(g, g^\tau, \ldots, g^{\tau^q})$ where $\tau \xleftarrow{\$} \mathbb{F}$ and q is poly(λ), the probability of $\mathcal{A}$ outputting τ is $\mathsf{negl}(\lambda)$.*

Here, we present the necessary assumption for IPAs (see Section F, the *q-Auxiliary Structured Double Pairing (q-ASDBP* assumption in $\mathbb{G}_1$ and $\mathbb{G}_2$. First, we present this assumption in $\mathbb{G}_2$ below

Assumption 2 (q-ASDBP) *For any PPT adversary $\mathcal{A}$,*

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

We note that the $\mathbb{G}_1$ variant of q-ASDBP is defined similarly by swapping $\mathbb{G}_2$ with $\mathbb{G}_1$.

KZG commitments. Let $(\mathbb{p}, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$ be pairing parameters and let secret $\tau \in \mathbb{F}$ be chosen at random. A trusted party outputs the elements $g, g^\tau, \ldots, g^{\tau^q}$ for some polynomially-large q. For univariate polynomial $f(X)$ over variable X, the KZG commitment [20] of f is $g^{f(\tau)}$, which we write as $[f(X)]$. To prove a *single-point evaluation* $f(w) = z$, the prover computes the quotient commitment $[q(X)] = [(f(X) - z)/(X - w)]$ which can be verified with a pairing as $e([f(X)] \cdot g^{-z}, g) = e([q(X)], [X - w])$. In addition to single-point evaluation, the KZG commitment [20] allows a prover to run a *zero-check*, i.e., prove that the committed polynomial satisfies $f(x_i) = 0$ for a set of points $x_1, \ldots, x_t$. To do that, the prover computes the quotient $q(X) = f(X)/\prod_{i\in[t]}(X - x_i)$ and outputs

the commitment $[q(X)]$ as a proof. To verify, the verifier uses the bilinear map to check that

$$e([f(X)], g) = e\left([q(X)], \left[\prod_{i\in[t]}(X - x_i)\right]\right).$$

Our protocols rely heavily on the KZG zero-check. We finally note that we will be using KZG commitments on bivariate polynomials $f(X, Y)$ as well, where the trusted setup outputs $\{g^{\tau_X^i \tau_Y^j}\}_{i,j=0,\ldots,q}$, for random τ_X and τ_Y, or $\{[X^i Y^j]\}_{i,j=0,\ldots,q}$.

Efficient batching openings for KZG commitments. Suppose $f(X)$ is interpolated from a vector $\mathbf{z} \in \mathbb{F}^n$, i.e., $f(\omega^i) = \mathbf{z}[i]$ for all $i \in [n]$. Given the commitment $[f(X)]$, to show $f(\omega^i) = \mathbf{z}[i]$ for each $i \in I$ where $I \subseteq [n]$ is arbitrary subset, the prover can compute the commitment to a polynomial $q_I(X)$ as follows

$$q_I(X) = \frac{f(X) - r_I(X)}{\prod_{i\in I}(X - \omega^i)}$$

where $r_I(X)$ is a degree-$(|I| - 1)$ polynomial interpolated over the values in I:

$$r_I(\omega^i) = \mathbf{z}[i], \quad \forall i \in [i].$$

We discuss the efficient computation of $[q_I(X)]$ in two scenarios:

1. The prover knows $\mathbf{z}$ in advance and is allowed to pre-compute a linear size data structure. According to [38], the prover can pre-compute the commitments to $q_i(X) = \frac{f(X)-\mathbf{z}[i]}{X-\omega^i}$ for each $i \in [n]$. When I is given, the prover can compute $[q_I(X)]$ using the following equation:

$$q_I(X) = \sum_{i\in I} \frac{1}{A_I'(\omega^i)} \cdot q_i(X)$$

   where $A_I(X) = \prod_{i\in I}(X - \omega^i)$ and $A_I'(X)$ is the derivative of $A_I(X)$. Given $[q_i(X)]$, the prover can compute $[q_I(X)]$ in $O(|I| \log^2 |I|)$ time.
2. $\mathbf{z}$ is all 0s outside I, i.e., $\mathbf{z}[i] = 0$ for all $i \in [n] \setminus I$. According to Theorem 5.1 in [23], we can start with an all-zero vector with $q_i(X) = 0$ for all $i \in [n]$, and then modify each location in I to the value of $\mathbf{z}$ and update all the commitments for $[q_i(X)]$ in $O(|I| \log^2 |I|)$ time. Finally, $[q_I(X)]$ can be computed from each $q_i(X)$ for $i \in I$ using the method above within $O(|I| \log^2 |I|)$ time.
   Also, we may need to compute another batch proof to show that $\mathbf{z}$ is all 0s at $[n] \setminus I$. In particular, what we want is the commitment to the following polynomial:

$$\overline{q}_I(X) = \frac{f(X)}{\prod_{i\in[n]\setminus I}(X - \omega^i)} = \frac{f(X) \cdot \prod_{i\in I}(X - \omega^i)}{X^n - 1}.$$

Since $\overline{q}_I(X)$ has degree $|I|-1$, based on the tricks from [38,17], we can compute its evaluations at $\{\omega^i \mid i \in I\}$ in $O(|I|\log^2|I|)$ time and interpolate its coefficients using FFT. Therefore, the commitment to such $\overline{q}_I(X)$ can also be computed in $O(|I|\log^2|I|)$ time.

Extractors. Following [19] we write $(a;b) \leftarrow (\mathcal{A}||\mathcal{E}_{\mathcal{A}})(x)$ to indicate that adversary $\mathcal{A}$ and extractor $\mathcal{E}$ are given the same input x and output a and b respectively. We write $\mathcal{E}_{\mathcal{A}}$ to indicate that $\mathcal{E}$ also takes as input $\mathcal{A}$'s state, including any random coins.

zk-SNARKs. We now present the definition of circuit-specific zk-SNARKs [19]. For zk-SNARKs with universal setup, algorithm $\mathcal{G}$ below is separated into two algorithms, a universal generation $\mathcal{G}(1^\lambda, |\mathtt{i}|) \to \mathsf{pp}$ and an indexer $\mathcal{I}(\mathsf{pp}, \mathtt{i}) \to (\mathsf{pk}, \mathsf{vk})$. To avoid complexity in our presentation, all our constructions are presented as circuit-specific, but we show how to turn them into universal. In both circuit-specific and universal zk-SNARKs, algorithm $\mathcal{G}$ must be trusted.

Definition 3 (zk-SNARKs). *A zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) for indexed relation $\mathcal{R}$ is a tuple of the following PPT algorithms $\mathsf{S} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$:*

- $\mathcal{G}(1^\lambda, \mathtt{i}) \to (\mathsf{pk}, \mathsf{vk})$: *Given 1^λ and indexed relation $\mathtt{i}$, outputs prover key pk and verifier key vk.*
- $\mathcal{P}(\mathsf{pk}, \mathtt{x}, \mathtt{w}) \to \pi$: *Given prover key pk, instance $\mathtt{x}$, and witness $\mathtt{w}$, outputs proof π.*
- $\mathcal{V}(\mathsf{vk}, \mathtt{x}, \pi) \to 0/1$: *Given verifier key vk, instance $\mathtt{x}$, and a proof π, outputs accept or reject.*

A zk-SNARK S should have polylog-sized proofs and satisfy the following properties.

- Completeness*: Let $\mathcal{G}(1^\lambda, \mathtt{i}) \to (\mathsf{pk}, \mathsf{vk})$. We say that S satisfies* completeness *if for any $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$, if $\mathcal{P}(\mathsf{pk}, \mathtt{x}, \mathtt{w}) \to \pi$, then $\mathcal{V}(\mathsf{vk}, \mathtt{x}, \pi) \to 1$.*
- Knowledge Soundness*: We say that S satisfies* knowledge soundness *if for any PPT adversary $\mathcal{A}$ and for any $\mathtt{i}$ there exists a PPT extractor $\mathcal{E}_{\mathcal{A}}$ such that*

$$\Pr\left[\begin{array}{c}\mathcal{G}(1^\lambda, \mathtt{i}) \to (\mathsf{pk}, \mathsf{vk}); ((\mathtt{x}, \pi); \mathtt{w}) \leftarrow (\mathcal{A}||\mathcal{E}_{\mathcal{A}})(\mathsf{pk}, \mathsf{vk}) \\ : \\ \mathcal{V}(\mathsf{vk}, \mathtt{x}, \pi) \to 1 \wedge (\mathtt{i}, \mathtt{x}, \mathtt{w}) \notin \mathcal{R}\end{array}\right]$$

  is negligible.
- Zero Knowledge*: Fix $\mathtt{i}$ and $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$. Let $\mathcal{D}$ be π's distribution output by the experiment below.*
  1. *$\mathcal{G}(1^\lambda, \mathtt{i}) \to (\mathsf{pk}, \mathsf{vk})$.*
  2. *$\mathcal{P}(\mathsf{pk}, \mathtt{x}, \mathtt{w}) \to \pi$.*

  We say that S satisfies zero-knowledge *if there exists a PPT simulator $\mathcal{S}$ such that the distribution $\tilde{\mathcal{D}}$ of $\tilde{\pi}$ output by the following experiment is computationally-indistinguishable from $\mathcal{D}$.*

1. $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{S}(1^\lambda, \mathbb{i})$.
2. $\tilde{\pi} \leftarrow \mathcal{S}(\mathsf{pk}, \mathsf{vk}, \mathbb{x})$.

The zero-knowledge definition naturally extends to statistical/perfect zero-knowledge.

Bounded-step IVC definition.

Definition 4 (BIVC). *A bounded incremental verifiable computation scheme (BIVC) scheme is a tuple of PPT algorithms* $\mathsf{I} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ *with the following interface:*

- $\mathcal{G}(1^\lambda, \mathsf{F}, N) \rightarrow (\mathsf{pk}, \mathsf{vk})$. *On input security parameter, the function* F *and the number of iterations* N*, it outputs a prover key* pk *and verification key* vk.
- $\mathcal{P}(\mathsf{pk}, i, z_0, \pi_{i-1}, z_{i-1}, w_{i-1}, z_i) \rightarrow \pi_i$. *On input a counter* i*, initial input* z_0*, a proof* π_{i-1} *and value* z_{i-1}*, auxiliary input* w_{i-1} *and value* z_i *and the prover key* pk*, it outputs a new proof* π_i.
- $\mathcal{V}(\mathsf{vk}, (i, z_i, z_0), \pi_i) \rightarrow 0/1$: *On input verification key* vk*, counter* i*, output* z_i*, initial input* z_0 *and proof* π_i*, outputs accept or reject.*

A BIVC scheme I *should satisfy the following properties.*

- Completeness*: Let* N *be polynomially-bounded and let* $\mathcal{G}(1^\lambda, \mathsf{F}, N) \rightarrow (\mathsf{pk}, \mathsf{vk})$ *for some function* F*. We say that* I *satisfies* completeness *if for all* $i \leq N$*, for all* $z_0, z_1, \ldots, z_i$ *and for all* $w_0, w_1, \ldots, w_{i-1}$ *such that* $F(z_0, w_0) = z_1, \ldots, F(z_{i-1}, w_{i-1}) = z_i$ *it is:*
  For all $1 \leq j \leq i$*, if we have* $\mathcal{P}(\mathsf{pk}, j, z_0, \pi_{j-1}, z_{j-1}, w_{j-1}, z_j) \rightarrow \pi_j$*, then* $\mathcal{V}(\mathsf{vk}, (i, z_j, z_0), \pi_j) \rightarrow 1$.
- Knowledge Soundness*: We say that* I *satisfies* knowledge soundness *if for all for all* N*, for all* $i \leq N$*, for all PPT adversary* $\mathcal{A}$ *and for all functions* F *there exists a PPT extractor* $\mathcal{E}_\mathcal{A}$ *such that*

$$\Pr\left[\begin{array}{c} \mathcal{G}(1^\lambda, \mathsf{F}, N) \rightarrow (\mathsf{pk}, \mathsf{vk}); \\ (((i, z_i, z_0), \pi), z_0, w_0, \ldots, z_{i-1}, w_{i-1}, z_i) \leftarrow (\mathcal{A} || \mathcal{E}_\mathcal{A})(\mathsf{pk}, \mathsf{vk}) \\ \vdots \\ \mathcal{V}(\mathsf{vk}, (i, z_i, z_0), \pi) \rightarrow 1 \wedge \exists j \in [1, i] : z_j \neq F(z_{j-1}, w_{j-1}) \end{array}\right]$$

  is negligible.
- Zero Knowledge*: Fix* N*, any iteration number* $i \leq N$*, any* F *and some valid tuple* $(i, z_0, w_0, \ldots, z_{i-1}, w_{i-1}, z_i)$*. Let* $\mathcal{D}$ *be the distribution of* π *as output by the experiment below.*
  1. $\mathcal{G}(1^\lambda, \mathsf{F}, N) \rightarrow (\mathsf{pk}, \mathsf{vk})$*;*
  2. *For* $j = 1$ *to* i *output* $\mathcal{P}(j, z_0, \pi_{j-1}, z_{j-1}, w_{j-1}, z_j, \mathsf{pk}) \rightarrow \pi_j$.

  We say that I *satisfies* zero-knowledge *if there exists a PPT simulator* $\mathcal{S}$ *such that the distribution* $\tilde{\mathcal{D}}$ *of* $\tilde{\pi}_j$ *output by the following experiment is computationally-indistinguishable from* $\mathcal{D}$.
  1. $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{S}(1^\lambda, \mathsf{F}, N)$.
  2. $(\tilde{\pi}_1, \ldots, \tilde{\pi}_i) \leftarrow \mathcal{S}(\mathsf{pk}, \mathsf{vk}, (i, z_0, z_i))$.

The zero-knowledge definition naturally extends to statistical/perfect zero-knowledge.

Just like in SNARKs, for BIVC with universal setup, algorithm $\mathcal{G}$ below is separated into two algorithms, a universal generation $\mathcal{G}(1^\lambda, |\mathsf{F}|, N) \to \mathsf{pp}$ and an indexer $\mathcal{I}(\mathsf{pp}, \mathsf{F}) \to (\mathsf{pk}, \mathsf{vk})$. To avoid complexity in our presentation, all our constructions are presented as specific to F, but we show how to turn them into universal. In both circuit-specific and universal IVC, $\mathcal{G}$ must be trusted.

B Auxiliary lemmas

Lemma 6. *Suppose $f(X_1, \ldots, X_s) \in \mathbb{F}_{\le d}[X_1, \ldots, X_s]$ is a non-zero s-variate polynomial over variable $X_1, \ldots, X_s$ such that every variable has degree at most d (total degree at most sd). Pick $r_1, \ldots, r_s \xleftarrow{\$} \mathbb{F}$. Then the univariate polynomial $g(X) := f(r_1 X, \ldots, r_s X)$ is zero polynomial with probability at most $d/|\mathbb{F}|$.*

Proof. Group the terms of $f(X_1, \ldots, X_s)$ by the same total degree into the following form:

$$f(X_1, \ldots, X_s) = \sum_{l=0}^{sd} \sum_{(i_1,\ldots,i_s): i_1+\ldots+i_s=l} a_{i_1,\ldots,i_s} X_1^{i_1} \ldots X_s^{i_s} .$$

Then we have

$$g(X) := f(r_1 X, \ldots, r_s X) = \sum_{l=0}^{sd} X^l \sum_{(i_1,\ldots,i_s): i_1+\ldots+i_s=l} a_{i_1,\ldots,i_s} r_1^{i_1} \ldots r_s^{i_s} .$$

Since $f(X_1, \ldots, X_s)$ is non-zero, we assume $a_{i'_1,\ldots,i'_s} \neq 0$ and let $l' = i'_1 + \ldots + i'_s$. Consider the following multivariate polynomial:

$$h(X_1, \ldots, X_s) := \sum_{(i_1,\ldots,i_s): i_1+\ldots+i_s=l'} a_{i_1,\ldots,i_s} X_1^{i_1} \ldots X_s^{i_s} .$$

The number of roots $(a_1, \ldots, a_s) \in \mathbb{F}^s$ of $h(X_1, \ldots, X_s)$ is at most $|\mathbb{F}|^{s-1} \cdot d = d|\mathbb{F}|^{s-1}$, thus

$$\Pr\left[h(r_1, \ldots, r_s) = 0 \;\middle|\; r_1, \ldots, r_s \xleftarrow{\$} \mathbb{F} \right] \le \frac{d|\mathbb{F}|^{s-1}}{|\mathbb{F}|^s} = \frac{d}{|\mathbb{F}|} .$$

Finally, we have

$$\begin{aligned} &\Pr\left[g(X) \text{ is zero polynomial} \;\middle|\; r_1, \ldots, r_s \xleftarrow{\$} \mathbb{F} \right] \\ \le &\Pr\left[\sum_{(i_1,\ldots,i_s): i_1+\ldots+i_s=l'} a_{i_1,\ldots,i_s} r_1^{i_1} \ldots r_s^{i_s} = 0 \;\middle|\; r_1, \ldots, r_s \xleftarrow{\$} \mathbb{F} \right] \\ = &\Pr\left[h(r_1, \ldots, r_s) = 0 \;\middle|\; r_1, \ldots, r_s \xleftarrow{\$} \mathbb{F} \right] \le \frac{d}{|\mathbb{F}|} . \end{aligned}$$

□

The following lemma is Lemma 1 from [30] that is helpful to prove the zero-knowledge property. We refer to [30] to see its formal proof.

Lemma 7 ([30]). *Let $S \subset \mathbb{F}$ and $Z_S(X) := \prod_{a\in S}(X-a)$. Fix a polynomial $f \in \mathbb{F}[X]$ and any distinct values $x_1, \ldots, x_k \in \mathbb{F} \setminus S$. Then the following distribution is uniform in $\mathbb{F}^k$:*

1. *Choose a random polynomial $\rho \leftarrow \mathbb{F}^{(\leq k-1)}[X]$ of degree $k-1$ and define*
$$\tilde{f}(X) := f(X) + Z_S(X)\rho(X)\,.$$
2. *Output $(\tilde{f}(x_1), \ldots, \tilde{f}(x_k)) \in \mathbb{F}^k$.*

C Algebraic Group Model

Pairings for polynomial check. In our protocols, we may need to check the following is a zero polynomial

$$\sum_{i\in[t]} f_{1,i}(X_1, \ldots, X_s) \cdot f_{2,i}(X_1, \ldots, X_s) \equiv 0 \qquad \text{(Polynomial check)}$$

for polynomials $f_{1,i}, f_{2,i}$ $(i \in [t])$ over variables $X_1, \ldots, X_s$. Instead of sending the whole polynomials to the verifier, the prover computes

$$[f_{1,i}(X_1, \ldots, X_s)], [f_{2,i}(X_1, \ldots, X_s)], \forall i \in [t]$$

so that the verifier checks if

$$\prod_{i\in[t]} e([f_{1,i}(X_1, \ldots, X_s)], [f_{2,i}(X_1, \ldots, X_s)]) = 1 \qquad \text{(Pairing check)}$$

The following lemma states that it suffices to use pairing checks instead of polynomial checks.

Lemma 8. *For any PPT algebraic adversary $\mathcal{A}$, given $\mathsf{pp}_{\mathsf{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$ and $\mathbf{L} = \{g^{h_i(\alpha_1,\ldots,\alpha_s)}\}_h$ as the initial list ($h_i(X_1, \ldots, X_s)$ are some pre-defined public polynomials), the following probability is negligible under q-DLOG assumption:*

$$\Pr\left[\begin{array}{l} C_{l,i} = [f_{l,i}(X_1, \ldots, X_s)],\ \forall i \in [t], l \in \{1,2\} \\ \wedge \sum_{i\in[t]} f_{1,i}(X_1, \ldots, X_s) \cdot f_{2,i}(X_1, \ldots, X_s) \not\equiv 0 \\ \wedge \prod_{i\in[t]} e(C_{1,i}, C_{2,i}) = 1 \end{array} : \begin{array}{r} \{C_{l,i}\}_{i\in[t],l\in\{1,2\}} \\ \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{bl}}, \mathbf{L}) \end{array}\right]$$

Proof. Suppose $\mathcal{A}$ is an adversary as described in the lemma statement. Here, we construct another adversary $\mathcal{A}^*$ for q-DLOG assumption:

$\mathcal{A}^*(\mathsf{pp}_{\mathsf{bl}}, (g, g^\tau, \ldots, g^{\tau^q}))$:

1. Pick $r_1, \ldots, r_s \xleftarrow{\$} \mathbb{F}$. Let $\alpha_1 := r_1\tau,\ \ldots,\ \alpha_s := r_s\tau$. Compute
$$\mathbf{L} = \left\{ g^{h_i(\alpha_1,\ldots,\alpha_s)} = g^{h_i(r_1\tau,\ldots,r_s\tau)} \right\}_h$$
and sends $\mathsf{pp}_{\mathsf{bl}}$ and $\mathbf{L}$ to $\mathcal{A}$.
2. Receive $\{C_{l,i}\}_{i\in[t],l\in\{1,2\}}$ from $\mathcal{A}$. Note that since $\mathcal{A}$ is algebraic, $\mathcal{A}$ should also outputs vectors to show how each group element in $(C_{1,i}, C_{2,i})_{i\in[t]}$ can be computed from $\mathbf{L}$. Thus $\mathcal{A}^*$ can reconstruct $f_{l,i}(X_1, \ldots, X_s)$ such that
$$C_{l,i} = [f_{l,i}(X_1, \ldots, X_s)], \quad \forall i \in [t], l \in \{1, 2\}.$$
3. If the following holds:
$$\sum_{i\in[t]} f_{1,i}(X_1, \ldots, X_s) \cdot f_{2,i}(X_1, \ldots, X_s) \not\equiv 0 \ \wedge$$
$$\prod_{i\in[t]} e([f_{1,i}(X_1, \ldots, X_s)], [f_{2,i}(X_1, \ldots, X_s)]) = 1\,,$$
then $\mathcal{A}^*$ knows that $\sum_{i\in[t]} f_{1,i}(X_1, \ldots, X_s)\cdot f_{2,i}(X_1, \ldots, X_s)$ is a non-zero polynomial which evaluates 0 on $(\alpha_1, \ldots, \alpha_s)$. According to Lemma 6, $g(X) := \sum_{i\in[t]} f_{1,i}(r_1X, \ldots, r_sX) \cdot f_{2,i}(r_1X, \ldots, r_sX)$ is a zero polynomial with probability at most $d/|\mathbb{F}|$ (d the maximum degree of any variable in $\sum_{i\in[t]} f_{1,i}(X_1, \ldots, X_s) \cdot f_{2,i}(X_1, \ldots, X_s)$), which is negligible. Factor $g(X)$ and output the root τ.

Therefore, if $\mathcal{A}$ can success with non-negligible probability, then $\mathcal{A}^*$ can also break q-DLOG with non-negligible probability. □

Degree check. Suppose for $\mathsf{pp} = \{g^{h_i(\alpha_1,\ldots,\alpha_s)}\}_h$, $d_1, \ldots, d_s$ are the maximum degree of $X_1, \ldots, X_s$ among $h_i(X_1, \ldots, X_s)$, i.e.,
$$d_i = \max_i \mathsf{deg}_{X_i} h_i(X_1, \ldots, X_s), \quad \text{for } i \in [s]$$
If we want to check that a polynomial $f(X_1, \ldots, X_s)$ is only over variables $X_2, X_3, \ldots, X_s$ without X_1, i.e., $\mathsf{deg}_{X_1} f(X_1, \ldots, X_s) = 0$, then the prover can computes $[f(X_1, \ldots, X_s)]$ and $[f(X_1, \ldots, X_s)X_1^{d_1}]$ so that the verifier can check if
$$e([f(X_1, \ldots, X_s)], [X_1^{d_1}]) = e([f(X_1, \ldots, X_s)X_1^{d_1}], g) \qquad \text{(Degree check)}$$
Similarly, we can check the following to ensure f has degree at most t over X_3:
$$e([f(X_1, \ldots, X_s)], [X_3^{d_3-t}]) = e([f(X_1, \ldots, X_s)X_3^{d_3-t}], g)$$
The following lemma states that it suffices to use degree checks to limit the degree of variables in some polynomial f.

Lemma 9. *For any PPT algebraic adversary $\mathcal{A}$, given $\mathsf{pp}_{\mathsf{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$ and $\mathbf{L} = \{g_l^{h_i(\alpha_1,\ldots,\alpha_s)}\}_h$ as the initial list ($h_i(X_1,\ldots,X_s)$ are pre-defined public polynomials) where $d_1,\ldots,d_s$ are the maximum degree of $X_1,\ldots,X_s$ among $h_i(X_1,\ldots,X_s)$, the following probability is negligible under q-DLOG assumption:*

$$\Pr\left[\begin{array}{ll} C=[f(X_1,\ldots,X_s)], C'=[f'(X_1,\ldots,X_s)] & \\ \wedge \mathsf{deg}_{X_1} f(X_1,\ldots,X_s) > t & : (C,C') \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{bl}}, \mathbf{L}) \\ \wedge e([C,[X_1^{d_1-t}]_2) = e(C', g_2) & \end{array}\right]$$

Similar results apply for other variable(s).

Proof. We only consider the case for $l = 1$ and variable X_1. Suppose $\mathcal{A}$ is an adversary as described in the lemma statement. Here, we construct another adversary $\mathcal{A}^*$ for q-DLOG assumption:

$\mathcal{A}^*(\mathsf{pp}_{\mathsf{bl}}, (g_1, g_1^\tau, \ldots, g_1^{\tau^q}), (g_2, g_2^\tau, \ldots, g_2^{\tau^q})) :$

1. Pick $r_1,\ldots,r_s \xleftarrow{\$} \mathbb{F}$. Let $\alpha_1 := r_1\tau,\ \ldots,\ \alpha_s := r_s\tau$. Compute
$$\mathbf{L} = \left\{ g_l^{h_{l,i}(\alpha_1,\ldots,\alpha_s)} = g_l^{h_{l,i}(r_1\tau,\ldots,r_s\tau)} \right\}_{l\in\{1,2\},h}$$
and sends $\mathsf{pp}_{\mathsf{bl}}$ and $\mathbf{L}$ to $\mathcal{A}$.
2. Receive (C, C') from $\mathcal{A}$. Note that since $\mathcal{A}$ is algebraic, $\mathcal{A}$ should also outputs vectors to show how (C, C') can be computed from $\mathbf{L}$. Thus $\mathcal{A}^*$ can reconstruct $f(X_1,\ldots,X_s), f'(X_1,\ldots,X_s)$ such that
$$C = [f(X_1,\ldots,X_s)], C' = [f'(X_1,\ldots,X_s)].$$
Note that $\mathsf{deg}_{X_1} f(X_1,\ldots,X_s) \le d_1,\ \mathsf{deg}_{X_1} f'(X_1,\ldots,X_s) \le d_1$.
3. If the following holds:
$$\begin{gathered}\mathsf{deg}_{X_1} f(X_1,\ldots,X_s) = t \\ \wedge\ e([f(X_1,\ldots,X_s)], [X_1^{d_1-t}]) = e([f'(X_1,\ldots,X_s)], g)\,,\end{gathered}$$
then $\mathcal{A}^*$ knows that $f(X_1,\ldots,X_s)\cdot X_1^{d_1-t} - f'(X_1,\ldots,X_s)$ is a non-zero polynomial which evaluates 0 on $(\alpha_1,\ldots,\alpha_s)$. According to Lemma 6, the polynomial $g(X) := f(r_1X,\ldots,r_sX)\cdot(r_1X)^{d_1-t} - f'(r_1X,\ldots,r_sX)$ is a zero polynomial with probability at most $d/|\mathbb{F}|$ (d the maximum degree of any variable in $f(X_1,\ldots,X_s)\cdot X_1^{d_1-t} - f'(X_1,\ldots,X_s)$), which is negligible. Factor $g(X)$ and output the root τ.

Therefore, if $\mathcal{A}$ can success with non-negligible probability, then $\mathcal{A}^*$ can also break q-DLOG with non-negligible probability. □

D Consistency check between two polynomial commitments

We propose a protocol to prove the consistency between two polynomials. Say m, ℓ are powers of two and ω, φ are the m-th and ℓ-th roots of unity respectively. Note that $\varphi = \omega^{\frac{m}{\ell}}$. Let $L_i(X)$ be the Lagrange polynomial for size-m vectors and $L'_i(X)$ be the Lagrange polynomial for size-ℓ vectors. We build a protocol for the following relation (similar approach can be applied for $\mathcal{R}_{m,n,k}$ which only does not require $a(\omega^j) = 0$ for $j \in [m] \setminus [k, k+\ell-1]$):

$$\mathcal{R}^{(0)}_{m,\ell,k} = \left\{ \begin{pmatrix} (\mathbb{x} = ([a(X)], [b(X)]), \\ \mathbb{w} = (a(X), b(X))) \end{pmatrix} : \begin{array}{l} a(\omega^j) = b(\varphi^{j-k+1}),\ \forall j \in [k, k+\ell-1] \\ a(\omega^j) = 0,\ \forall j \in [m] \setminus [k, k+\ell-1] \end{array} \right\}.$$

For honestly interpolated $[a], [b]$, we show our protocol in the following steps:

1. Batch proof for $a(\omega^j) = 0$ for $j \in [m] \setminus [k, k+\ell-1]$:

$$a(X) = q_0(X) \prod_{i \in [m] \setminus [k, k+\ell-1]} (X - \omega^i)\,.$$

2. Define $c(X)$ as

$$c(X) = \sum_{i \in [k, k+\ell-1]} a(\omega^i) \cdot L'_{i-k+1}(\varphi^{1-k} X^{\frac{m}{\ell}}) = \sum_{i \in [\ell]} a(\omega^{i+k-1}) \cdot L'_i(\varphi^{1-k} X^{\frac{m}{\ell}})\,.$$

It is easy to see that for all $j \in [k, k+\ell-1]$,

$$c(\omega^j) = \sum_{i \in [k, k+\ell-1]} a(\omega^i) \cdot L'_{i-k+1}(\varphi^{j-k+1}) = a(\omega^j)\,.$$

The following equation checks this:

$$a(X) - c(X) = q_{ac}(X) \prod_{j \in [k, k+\ell-1]} (X - \omega^j)\,. \tag{11}$$

3. Show that $b(\varphi^{j-k+1}) = c(\omega^j)$ for $j \in [k, k+\ell-1]$. This can be done by introducing a new variable Y and doing variable substitution. Define $d(Y) := c(Y)$ and show that

$$c(X) - d(Y) = q_{cd}(X, Y)(X - Y)\,, \tag{12}$$

$$b(X) - d(Y) = q_{bd}(X, Y)(X - \varphi^{1-k} Y^{\frac{m}{\ell}})\,.$$

Briefly speaking, the first equation shows that $c(\omega^j) = d(\omega^j)$ for all j, and the second equation shows that $d(\omega^j) = b(\varphi^{j-k+1})$ for all j, which concludes that $b(\varphi^{j-k+1}) = d(\omega^j) = c(\omega^j) = a(\omega^j)$.

Wrapping up. We can combine Eqs. (11) and (12) and remove $c(X)$ through the following check:

$$a(X) - d(Y) = q_{ac}(X) \prod_{j\in[k,k+\ell-1]} (X - \omega^j) + q_{cd}(X,Y)(X - Y)\,.$$

The final proof will be KZG commitments to $q_0, q_{ac}, q_{cd}, q_{bd}, d$ as well as degree checks for a, b, d:

$$A(X,Y) = a(X)\cdot Y^m,\ B(X,Y) = b(X)\cdot Y^m,\ D(X,Y) = d(Y)\cdot X^m\,.$$

In fact, we can pre-process some polynomials in the setup phase so that the commitment to each polynomial can be computed in $O(\ell)$ time. We briefly present relevant theorems and the full protocol for $\mathcal{R}^{(0)}_{m,\ell,k}$ in the following.

Theorem 7 (Efficient proof computation in $\mathcal{R}^{(0)}_{m,\ell,k}$). *Let*

$$\mathcal{F} = \{q_0, q_{ac}, q_{cd}, q_{bd}, d, A, B, D\}$$

be the set of eight polynomials contained in the $\mathcal{R}^{(0)}_{m,\ell,k}$ proof. Every $f \in \mathcal{F}$ can be expressed as

$$f = \sum_{i\in[\ell]} f_i \cdot b(\varphi^i)\,,$$

where $\{f_1, \ldots, f_\ell\}_{f\in\mathcal{F}}$ are six fixed sets of ℓ polynomials each defined for every $f \in \mathcal{F}$.

Proof. For each polynomial in $\mathcal{F}$,

- $d(Y), A(X,Y), B(X,Y), D(X,Y)$. From its definition, $d_i(Y) = L'_i(\varphi^{1-k}Y^{\frac{m}{\ell}})$. A_i, B_i, D_i can be easily derived from the definition of a, b, d.
- $q_0(X), q_{ac}(X)$. Also from definition,

$$(q_0)_i(X) = \frac{L_{i+k-1}(X)}{\prod_{j\in[m]\setminus[k,k+\ell-1]}(X-\omega^j)},$$

$$(q_{ac})_i(X) = \frac{L_{i+k-1}(X) - L'_i(\varphi^{1-k}X^{\frac{m}{\ell}})}{\prod_{j\in[k,k+\ell-1]}(X-\omega^j)}\,.$$

  Here, $(q_{ac})_i(X)$ is a polynomial because $L_{i+k-1}(X)$ and $L'_i(\varphi^{1-k}X^{\frac{m}{\ell}})$ agree on all $X = \omega^j$, $j \in [k, k+\ell-1]$. In particular, when $j \neq i+k-1$, $j \in [k, k+\ell-1]$,

$$L_{i+k-1}(\omega^j) = 0, \quad L'_i(\varphi^{1-k}(\omega^j)^{\frac{m}{\ell}}) = L'_i(\varphi^{j-k+1}) = 0\,.$$

  When $j = i+k-1$,

$$L_{i+k-1}(\omega^j) = 1, \quad L'_i(\varphi^{1-k}(\omega^j)^{\frac{m}{\ell}}) = L'_i(\varphi^i) = 1\,.$$

– $q_{cd}(X,Y), q_{bd}(X,Y)$.

$$(q_{cd})_i(X,Y) = \frac{L'_i(\varphi^{1-k}X^{\frac{m}{\ell}}) - L'_i(\varphi^{1-k}Y^{\frac{m}{\ell}})}{X-W}\,.$$

$$(q_{bd})_i(X,Y) = \frac{L'_i(X) - L'_i(\varphi^{1-k}Y^{\frac{m}{\ell}})}{X-\varphi^{1-k}Y^{\frac{m}{\ell}}}\,.$$

□

– $\mathcal{G}(1^\lambda,[m,\ell,k]) \to (\mathsf{pk},\mathsf{vk})$:
  - Let $\mathcal{F} = \{q_0, q_{ac}, q_{cd}, q_{bd}, d, A, B, D\}$.
  - Pick random τ_X, τ_Y from $\mathbb{F}$ for variables X and Y respectively.
  - Set pk to contain the following KZG commitments, defined in Theorem 7, and computed using τ_X and τ_Y directly

$$\{[f_1],\ldots,[f_\ell]\}_{f\in\mathcal{F}}\,.$$

  - Set $\mathsf{vk} = \{g, [z_{\mathsf{in}}(X)], [z_{\mathsf{out}}(X)], [X^m], [Y^m], [X-Y], [X-\varphi^{1-k}Y^{\frac{m}{\ell}}]\}$ where $z_{\mathsf{in}}(X) = \prod_{i\in[k,k+\ell-1]}(X-\omega^i)$, $z_{\mathsf{out}}(X) = \prod_{i\in[m]\setminus[k,k+\ell-1]}(X-\omega^i)$.

– $\mathcal{P}(\mathsf{pk},\mathbb{x},\mathbb{w}) \to \pi$:
  - Parse $\mathbb{x}$ as $([a],[b])$ and $\mathbb{w}$ as a,b.
  - Following Theorem 7, output five commitments as π, i.e., $\forall f\in\mathcal{F}$ output

$$[f] = \prod_{i\in[\ell]}[f_i]^{b(\varphi^i)}\,.$$

– $\mathcal{V}(\mathsf{vk},\mathbb{x},\pi) \to 0/1$:
  - Parse vk as $\{g,[z_{\mathsf{in}}],[z_{\mathsf{out}}],[X^m],[Y^m],[X-Y],[X-\varphi^{1-k}Y^{\frac{m}{\ell}}]\}$, $\mathbb{x}$ as $([a],[b])$ and π as $\{[q_0],[q_{ac}],[q_{cd}],[q_{bd}],[d],[A],[B],[D]\}$.
  - Output 1 if and only if all the following relations hold:

$$e([a],g) = e([q_0],[z_{\mathsf{out}}]).$$
$$e([a-d],g) = e([q_{ac}],[z_{\mathsf{in}}])e([q_{cd}],[X-Y]).$$
$$e([b-d],g) = e([q_{bd}],[X-\varphi^{1-k}Y^{\frac{m}{\ell}}]).$$
$$e([a],[Y^m]) = e([A],g).$$
$$e([b],[Y^m]) = e([B],g).$$
$$e([d],[X^m]) = e([D],g).$$

Fig. 8. The protocol for $\mathcal{R}^{(0)}_{m,\ell,k}$ SNARK.

Theorem 8 (SNARK for $\mathcal{R}^{(0)}_{m,\ell,k}$). *The protocol of Fig. 8 is a SNARK (per Definition 3) for $\mathcal{R}^{(0)}_{m,\ell,k}$ based on q-DLOG (see Assumption 1) in the AGM. Its complexities are as follows.*

1. *$\mathcal{G}$ runs in $O(\ell+\log m)$ time, outputs pk of $O(\ell)$ size and vk of $O(1)$ size;*

2. $\mathcal{P}$ *runs in* $O(\ell)$ *time and outputs a proof* π *of* $O(1)$ *size.*
3. $\mathcal{V}$ *runs in* $O(1)$ *time.*

Proof. Completeness and complexities follow naturally from the protocol and Theorem 7. For knowledge soundness, we use the following extractor and show that the extracted witness $\mathbb{w}$ satisfies the permutation σ:

1. Run the algebraic adversary $(\mathbb{x}, \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk})$.
2. Let $\mathbb{x} = ([a], [b])$. Note that $\mathcal{A}$ is algebraic, it also outputs vectors of scalars showing how $[a], [b]$ are computed from $(\mathsf{pk}, \mathsf{vk})$. $\mathcal{E}_{\mathcal{A}}$ reconstructs $\widetilde{a}(X), \widetilde{b}(X)$ such that $[a] = [\widetilde{a}(X)], [b] = [\widetilde{b}(X)]$.
3. Output $\mathbb{w} = (\widetilde{z}(X), \widetilde{b}(X))$.

Similar to the proof of Theorem 2, parse $\pi = \{[f]\}_{f \in \mathcal{F}}$ and reconstruct $\{\widetilde{f}(X, Y)\}_{f \in \mathcal{F}}$ such that $[f] = [\widetilde{f}(X, Y)]$. The degree checks ensure that $\widetilde{a}, \widetilde{b}, \widetilde{d}$ are $\widetilde{a}(X), \widetilde{b}(X), \widetilde{d}(Y)$ respectively with overwhelming probability.

From the first check by $\mathcal{V}$, we have

$$\widetilde{a}(\omega^j) = 0, \quad \forall j \in [m] \setminus [k, k + \ell - 1]\,. \tag{13}$$

From the second check by $\mathcal{V}$,

$$\widetilde{a}(\omega^j) = \widetilde{d}(\omega^j),\ \forall j \in [k, k + \ell - 1]\,.$$

From the last check, pick $W = \omega^j$ and $X = \varphi^{1-k} Y^{\frac{m}{\ell}} = \varphi^{j-k+1}$,

$$\widetilde{b}(\varphi^{j-k+1}) = \widetilde{d}(\omega^j)\,.$$

Thus we have for all $j \in [k, k + \ell - 1]$,

$$\widetilde{b}(\varphi^{j-k+1}) = \widetilde{d}(\omega^j) = \widetilde{c}(\omega^j) = \widetilde{a}(\omega^j)\,,$$

together with Eq. (13), the output $\mathbb{w}$ of the extractor should be a valid witness. □

Our protocol in Fig. 8 can also be zero-knowledge. We have the following lemma and its proof can be found in Section J.

Lemma 10 (Zero-knowledge SNARK for $\mathcal{R}^{(0)}_{m,\ell,k}$). *There is a zero-knowledge version of* $\mathcal{R}^{(0)}_{m,\ell,k}$ *SNARK with the same complexities.*

Similarly, we can build SNARKs for $\mathcal{R}_{m,\ell,k}$ where the only difference is that $a(\omega^j)$ could be non-zero when $j \in [m] \setminus [k, k + \ell - 1]$. Hence, we only need to remove q_0 in its protocol. We have the following theorems.

Theorem 9 (SNARK for $\mathcal{R}_{m,\ell,k}$). *There is a SNARK for* $\mathcal{R}_{m,\ell,k}$ *based on q-DLOG (see Assumption 1) in the AGM. Its complexities are as follows.*

1. $\mathcal{G}$ *runs in* $O(m + \log m)$ *time, outputs* pk *of* $O(m)$ *size and* vk *of* $O(1)$ *size;*
2. $\mathcal{P}$ *runs in* $O(m')$ *time and outputs a proof* π *of* $O(1)$ *size.* m' *is the number of non-zeros in* $\{a(\omega^j)\}_{j \in [m]}$.
3. $\mathcal{V}$ *runs in* $O(1)$ *time.*

Lemma 11 (Zero-knowledge SNARK for $\mathcal{R}_{m,\ell,k}$). *There is a zero-knowledge version of* $\mathcal{R}_{m,\ell,k}$ *SNARK with the same complexities.*

D.1 Universal consistency check

Now we briefly show how to make the protocol in Fig. 8 universal to avoid a quadratic setup. We will not present a detailed protocol here but only a sketched idea. This is similar to build universal Dynamo, namely introducing a random oracle and replacing Y with a random point. More specifically, we still have $c(X)$ as defined before

$$c(X) = \sum_{i\in[\ell]} a(\omega^{i+k-1}) \cdot L'_i(\varphi^{1-k} X^{\frac{m}{\ell}}) .$$

Note that

$$b(\varphi^{1-k} X^{\frac{m}{\ell}}) = \sum_{i\in[\ell]} b(\varphi^i) \cdot L'_i(\varphi^{1-k} X^{\frac{m}{\ell}}) .$$

If we would like to show that $b(\varphi^i) = c(\omega^{i+k-1})$ for $i \in [\ell]$, it is equivalent to check that

$$b(\varphi^{1-k} X^{\frac{m}{\ell}}) - c(X) = q_{bc}(X) \prod_{i\in[k,k+\ell-1]} (X - \omega^i) . \tag{14}$$

Let $r = \mathsf{H}([a]||[b]||[c]||[q_{bc}])$ be a random point from the random oracle. Then the verifier can just check that

$$b(\varphi^{1-k} r^{\frac{m}{\ell}}) - c(r) = q_{bc}(r) \prod_{i\in[k,k+\ell-1]} (r - \omega^i)$$

by verifying the openings of $[b]$ at $\varphi^{1-k} r^{\frac{m}{\ell}}$, $[c]$ at r and $[q_{bc}]$ at r.

We have the following theorem for the universal protocol.

Theorem 10 (Universal SNARK for $\mathcal{R}^{(0)}_{m,\ell,k}$). *Assuming a random oracle $\mathsf{H} : \{0,1\}^* \to \mathbb{F}$, there is a universal SNARK (per Definition 3) for $\mathcal{R}^{(0)}_{m,\ell,k}$ based on q-DLOG (see Assumption 1) in the AGM. Its complexities are as follows.*

1. *$\mathcal{G}$ runs in $O(m)$ time and outputs pp of $O(m)$ size;*
2. *$\mathcal{I}$ runs in $O(\ell \log m)$ time, outputs pk of $O(\ell)$ size and vk of $O(1)$ size;*
3. *$\mathcal{P}$ runs in $O(\ell \log \ell)$ time and outputs a proof π of $O(1)$ size.*
4. *$\mathcal{V}$ runs in $O(1)$ time.*

We can also make the universal SNARK zero-knowledge.

Lemma 12 (Universal zero-knowledge SNARK for $\mathcal{R}^{(0)}_{m,\ell,k}$). *There is a zero-knowledge version of $\mathcal{R}^{(0)}_{m,\ell,k}$ universal SNARK with the same complexities.*

Similarly, we have the following theorems for $\mathcal{R}_{m,\ell,k}$.

Theorem 11 (Universal SNARK for $\mathcal{R}_{m,\ell,k}$). *Assuming a random oracle $\mathsf{H} : \{0,1\}^* \to \mathbb{F}$, there is a universal SNARK (per Definition 3) for $\mathcal{R}_{m,\ell,k}$ based on q-DLOG (see Assumption 1) in the AGM. Its complexities are as follows.*

1. $\mathcal{G}$ *runs in* $O(m)$ *time and outputs* pp *of* $O(m)$ *size;*
2. $\mathcal{I}$ *runs in* $O(m \log m)$ *time, outputs* pk *of* $O(m)$ *size and* vk *of* $O(1)$ *size;*
3. $\mathcal{P}$ *runs in* $O(m' \log m)$ *time and outputs a proof* π *of* $O(1)$ *size.* m' *is the number of non-zeros in* $\{a(\omega^j)\}_{j\in[m]}$.
4. $\mathcal{V}$ *runs in* $O(1)$ *time.*

Lemma 13 (Universal zero-knowledge SNARK for $\mathcal{R}_{m,\ell,k}$). *There is a zero-knowledge version of* $\mathcal{R}_{m,\ell,k}$ *universal SNARK with the same complexities.*

E AMT

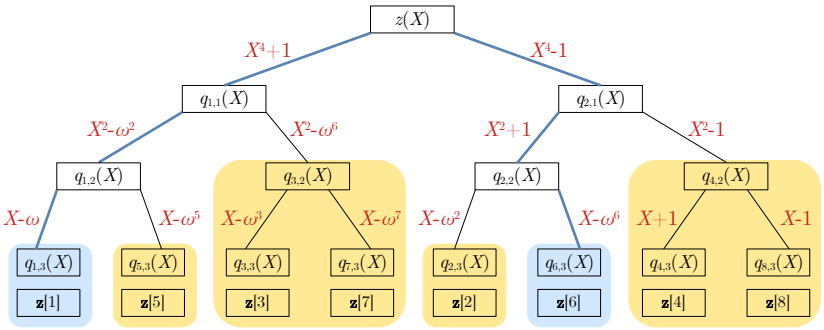

Fig. 9. A sample AMT quotient tree with $m = 8$. The red polynomials on the edges are the divisors and the root of each sub-tree is a polynomial shared by all the leaves in the sub-tree. Suppose in this $\mathbf{z}$ only $\mathbf{z}[1]$ and $\mathbf{z}[6]$ (blue part) are non-zero, then all the yellow sub-trees are empty except for their roots.

In this section we present the details of AMT [37]. To fit AMT into our protocols, we will describe AMT using KZG commitments and pairing-check Section 2. AMT is a vector commitment based on KZG commitments [20]. Suppose m is a power of two and $l = \log m$. To commit to a vector $\mathbf{z} \in \mathbb{F}^m$, interpolate a polynomial $z(X)$ such that $z(\omega^i) = \mathbf{z}[i]$ for all $i \in [m]$, and the commitment is the KZG commitment to $z(X)$. From KZG, we can open the commitment at a location i by computing a polynomial $q_i(X)$ and checking that

$$z(X) = \mathbf{z}_i + q_i(X)(X - \omega^i)\,.$$

Alternatively, we can compute polynomials $\{q_{i,j}(X)\}_{j\in[l-1]}$ instead and check the following equation:

$$z(X) = \mathbf{z}_i + \sum_{j=1}^{l} q_{i,j}(X)(X^{m/2^j} - (\omega^i)^{m/2^j})\,.$$

For such equation, there are two key observations:

– $\mathsf{AMT.Setup}(1^\lambda, m) \to (pk, vk)$:
  - Pick random τ_X from $\mathbb{F}$ for variable X.
  - Set $pk = [X^i]_{i\in[0,m]}$ and $vk = (g, [X^{2^j}]_{j\in[0,l]})$.

– $\mathsf{AMT.Commit}(pk, \mathbf{z}) \to \mathsf{com}(\mathbf{z})$:
  - Compute $z(X)$ and output $[z]$.

– $\mathsf{AMT.Prove}(pk, \mathbf{z}, i) \to \pi_i$:
  - Compute $z(X)$ from $\mathbf{z}$. Divide z by $X^{m/2^j} - (\omega^i)^{m/2^j}$ for each $j \in [l]$ and get $q_{i,j}(X)$.
  - Output π_i as $\{[q_{i,j}]\}_{j\in[l]}$.

– $\mathsf{AMT.Verify}(vk, \mathsf{com}(\mathbf{z}), (i, \mathbf{z}[i]), \pi) \to \{0, 1\}$:
  - Parse π as $(\pi_1, \ldots, \pi_l)$.
  - Output 1 if and only if

$$e(\mathsf{com}(\mathbf{z})/g^{\mathbf{z}[i]}, g) = \prod_{j=1}^{l} e(\pi_j, [X^{m/2^j} - (\omega^i)^{m/2^j}]) .$$

– $\mathsf{AMT.ComputeAllProofs}(pk, \mathbf{z}) \to \{\pi_1, \ldots, \pi_m, tree\}$:
  - Compute $z(X)$ from $\mathbf{z}$. Divide z by $X^{m/2^j} - (\omega^i)^{m/2^j}$ for each $j \in [l]$, $i \in [m]$ and get quotients $q_{i,j}(X)$. Compute each $[q_{i,j}]$ and build a tree as in Fig. 10.

– $\mathsf{AMT.ProveZeroSet}(pk, \mathbf{z}, tree) \to \{S, \pi_S\}$:
  - Let $S = \emptyset$ and $\pi_S = \emptyset$.
  - Traverse $tree$. Whenever there is a sub-tree $tree_t$ full of zero leaves, let i_t be the minimum index in this sub-tree. Denote $\pi_t = [q_{i_t,j}(X)]_{j\in[k_t]}$ as the commitments to the quotient polynomials on the path from the root to $tree_t$ and let $\pi_S \leftarrow \pi_S \cup \{\pi_t\}$.

– $\mathsf{AMT.VerifyZeroSet}(vk, \mathsf{com}(\mathbf{z}), S, \pi_S) \to \{0, 1\}$:
  - Rebuild each sub-tree full of zero leaves from S. For each such sub-tree $tree_t$, let i_t be the minimum index in this sub-tree, parse $\pi_t = [q_{i_t,j}(X)]_{j\in[k_t]}$ from π_S check if

$$e(\mathsf{com}(\mathbf{z}), g) = \prod_{j\in[k_t]} e([q_{i_t,j}], [X^{m/2^j} - (\omega^{i_t})^{m/2^j}]) .$$

Fig. 10. The protocol for AMT.

1. We can compute such $\{q_{i,j}(X)\}_{j\in[l-1]}$ through polynomial divisions. We first divide $z(X)$ by $X^{m/2}-(\omega^i)^{m/2}$ and put the quotient polynomial as $q_{i,1}(X)$. Then we can repeat the division on the remainder and get the following $q_{i,j}(X)$s.
2. Many locations share the same divisor $X^{m/2^j}-(\omega^i)^{m/2^j}$. More specifically, for any $j\in[l]$, every location of form $i+k\cdot 2^j$ where $i\in[2^j]$ and $k\in[m/2^j-1]$ share the same divisor:

$$X^{m/2^j}-(\omega^{i+k\cdot 2^j})^{m/2^j}=X^{m/2^j}-(\omega^i)^{m/2^j}\,.$$

   Thus if we compute $\{q_{i,j}(X)\}$ through polynomial divisions, then all the locations of form $i+k\cdot 2^j$ share the same first j divisors and thus the same first j quotient polynomials.

Based on this observation, we can build a tree for the quotient polynomials.

Proving zero evaluations from sub-trees. We can observe from the quotient tree that, if all the leaves of a sub-tree have zero values in $\mathbf{z}$, then all the quotients in this subtree are 0. Suppose this sub-tree has height k and the smallest index of the leaves is i, then all the leaves share the same evaluation check as follows:

$$z(X)=0+\sum_{j=1}^{l-k}q_{i,j}(X^{m/2^j}-(\omega^i)^{m/2^j})\,.$$

Therefore we can use the commitments to $\{q_{i,j}\}_{j\in[l-k]}$ as the proof of a zero sub-tree. If there are m' non-zero values in $\mathbf{z}$, then there are at most $O(m'l)=O(m'\log m)$ such zero sub-trees. We can fetch the $O(m'l^2)$ proof of all zero evaluations, i.e., KZG commitments of the quotient polynomials, from the tree directly.

We describe the algorithms for AMT in Fig. 10.

F Inner Product Arguments

Bünz et al. [5] give a non-interactive IPA which allows a prover to show that for $r\in\mathbb{F}$ (r could be $1^{\mathbb{F}}$) and $E_A,E_B,E_r\in\mathbb{G}_T$, they know $(\mathbf{A},\mathbf{B})\in\mathbb{G}_1^m\times\mathbb{G}_2^m$ such that E_A,E_B are pairing commitments to $\mathbf{A},\mathbf{B}$, and E_r is the inner pairing product with respect to $\mathbf{r}=[r^{2(i-1)}]_{i\in[m]}$:

$$E_r=\langle\mathbf{A}^{\mathbf{r}},\mathbf{B}\rangle=\prod_{i\in[m]}e(\mathbf{A}[i]^{r^{2(i-1)}},\mathbf{B}[i])\,.$$

More specifically, it is an argument for the following relation (we denote its language $\mathcal{L}^m_{\mathsf{IPA}}$):

$$\mathcal{R}^m_{\mathsf{IPA}} = \left\{ \begin{pmatrix} \mathbb{x} = (\mathsf{vk}_A = g_1^{\beta}, \mathsf{vk}_B = g_2^{\alpha}, r \in \mathbb{F}, \\ E_A, E_B, E_r \in \mathbb{G}_T), \\ \mathbb{w} = (\mathbf{r} = [r^{2(i-1)}]_{i\in[m]}, \mathbf{A} \in \mathbb{G}_1^m, \\ \mathbf{B} \in \mathbb{G}_2^m, \mathbf{v_A} = [g_2^{\beta^{2(i-1)}}]_{i\in[m]}, \\ \mathbf{v_B} = [g_1^{\alpha^{2(i-1)}}]_{i\in[m]})) \end{pmatrix} : \begin{array}{c} g_1 \xleftarrow{\$} \mathbb{G}_1, g_2 \xleftarrow{\$} \mathbb{G}_2, \\ \alpha, \beta \xleftarrow{\$} \mathbb{F} \\ \wedge E_A = \langle \mathbf{A}, \mathbf{v_A} \rangle \\ \wedge E_B = \langle \mathbf{v_B}, \mathbf{B} \rangle \\ \wedge E_r = \langle \mathbf{A^r}, \mathbf{B} \rangle \end{array} \right\}$$

We can also easily obtain a simplified variant of IPA to show that for $r \in \mathbb{F}, E_r \in \mathbb{G}_l$ ($l \in \{1,2\}$) and $E_A \in \mathbb{G}_T$, they know E_A is a pairing commitment to $\mathbf{A} \in \mathbb{G}_l^m$, and E_r is its combination with respect to $\mathbf{r} = [r^{2(i-1)}]_{i\in[m]}$:

$$E_r = \prod_{i\in[m]} \mathbf{A}[i]^{r^{2(i-1)}}.$$

We also define the corresponding relation here and denote its language as $\mathcal{L}^m_{\mathsf{EXP}}$.

$$\mathcal{R}^m_{\mathsf{EXP}} = \left\{ \begin{pmatrix} \mathbb{x} = (\mathsf{vk}_A = g_l^{\beta}, r \in \mathbb{F}, \\ E_A \in \mathbb{G}_T, E_r \in \mathbb{G}_l), \\ \mathbb{w} = (\mathbf{r} = [r^{2(i-1)}]_{i\in[m]}, \mathbf{A} \in \mathbb{G}_l^m, \\ \mathbf{v_A} = [g_{3-l}^{\beta^{2(i-1)}}]_{i\in[m]}) \end{pmatrix} : \begin{array}{c} g_1 \xleftarrow{\$} \mathbb{G}_1, g_2 \xleftarrow{\$} \mathbb{G}_2, \beta \xleftarrow{\$} \mathbb{F} \\ \wedge E_A = \langle \mathbf{A}, \mathbf{v_A} \rangle \\ \wedge E_r = \prod_{i\in[m]} \mathbf{A}[i]^{r^{2(i-1)}} \end{array} \right\}$$

Theorem 12 (IPA [5]). *There are interactive IPAs for $\mathcal{R}^m_{\mathsf{EXP}}$ and $\mathcal{R}^m_{\mathsf{IPA}}$ that have knowledge soundness and zero-knowledge (per Definition 3) under the q-ASDBP (see Assumption 2), the Algebraic Group Model and the Random Oracle Model. The prover takes $O(m)$ time, the verifier takes $O(\log m)$ time and the proof size is $O(\log m)$.*

F.1 Details of asymptotic optimization for Dynavold

Recall our example in Section H.1: Fix $t \in [1,6]$, the verifier needs to check for all $i \in [m]$,

$$e([\beta_{t,i}], [X^2]) = e([B_{t,i}], g).$$

If the verifier picks $r \xleftarrow{\$} \mathbb{F}$, then it can just check the following combination:

$$\prod_{i\in[m]} e([\beta_{t,i}]^{r^{2(i-1)}}, [X^2]]) = \prod_{i\in[m]} e([B_{t,i}]^{r^{2(i-1)}}, g),$$

which is equivalent to

$$e\left(\prod_{i\in[m]} [\beta_{t,i}]^{r^{2(i-1)}}, [X^2]\right) = e\left(\prod_{i\in[m]} [B_{t,i}]^{r^{2(i-1)}}, g\right). \tag{15}$$

Let $\boldsymbol{\beta}_t = [\beta_{t,i}]_{i\in[m]}$, $\boldsymbol{B}_t = [B_{t,i}]_{i\in[m]}$, $\mathbf{r} = [r^{2(i-1)}]_{i\in[m]}$. The prover can show that $E_{t,\beta,r}, E_{t,B,r} \in \mathbb{G}$ satisfy the following:

$$\begin{aligned} E_{\beta,t,r} &= \langle \boldsymbol{\beta}_t, \mathbf{r}\rangle = \prod_{i\in[m]} [\beta_{t,i}]^{r^{2(i-1)}}, \\ E_{B,t,r} &= \langle \boldsymbol{B}_t, \mathbf{r}\rangle = \prod_{i\in[m]} [B_{t,i}]^{r^{2(i-1)}}, \\ e(E_{\beta,t,r}, [X^2]) &= e(E_{B,t,r}, g)\,, \end{aligned}$$

where the first two equations can be proved by IPA. More specifically, using some fixed keys $\mathbf{v}_{\beta,t}, \mathbf{v}_{B,t} \in \mathsf{pk}$ and $\mathsf{vk}_{\beta,t}, \mathsf{vk}_{B,t} \in \mathsf{vk}$, the prover computes $E_{t,\beta}$ and $E_{t,B}$ and shows that

$$\begin{aligned} (\mathsf{vk}_{\beta,t}, r, E_{\beta,t}, E_{\beta,t,r}) &\in \mathcal{L}^m_{\mathsf{EXP}}\,, \\ (\mathsf{vk}_{B,t}, r, E_{B,t}, E_{B,t,r}) &\in \mathcal{L}^m_{\mathsf{EXP}}\,. \end{aligned}$$

We apply this trick for all the equations in the verification algorithm. Let

$$\mathcal{F}_{\mathsf{IPA}} = \{s_t, z_t, u_t, \alpha_t, \beta_t, h_t, Z_t, B_t, H_t\}_{t\in[1,6]} \cup \{A\}\,.$$

For each $f \in \mathcal{F}_{\mathsf{IPA}}$, define its bold form as $\mathbf{f} = [\,[f_i]\,]_{i\in[m]}$ and prepare IPA keys $\mathbf{v}_f \in \mathsf{pk}$, $\mathsf{vk}_f \in \mathsf{vk}$. After the verifier picks $r \in \mathbb{F}$, let $\mathbf{r} = [r^{2(i-1)}]$, the prover computes E_f for all $f \in \mathcal{F}_{\mathsf{IPA}}$. Now we go through all the checks in $\mathcal{V}$:

1. Addition constraint: $\forall i \in [\ell]$, $[s_{1,i}(X)] \cdot [s_{2,i}(X)] = [s_{3,i}(X)]$:
   - The prover computes $E_{s,1,r}, E_{s,2,r}, E_{s,3,r}$ such that
   $$E_{s,1,r} = \langle \mathbf{s}_1, \mathbf{r}\rangle, E_{s,2,r} = \langle \mathbf{s}_2, \mathbf{r}\rangle, E_{s,3,r} = \langle \mathbf{s}_3, \mathbf{r}\rangle.$$
   - The prover computes and sends the proof for the following statements of IPA:
   $$\begin{aligned} \pi_{s,1,r} &: (\mathsf{vk}_{s,1}, r, E_{s,1}, E_{s,1,r}) \in \mathcal{L}^m_{\mathsf{EXP}}, \\ \pi_{s,2,r} &: (\mathsf{vk}_{s,2}, r, E_{s,2}, E_{s,2,r}) \in \mathcal{L}^m_{\mathsf{EXP}}, \\ \pi_{s,3,r} &: (\mathsf{vk}_{s,3}, r, E_{s,3}, E_{s,3,r}) \in \mathcal{L}^m_{\mathsf{EXP}}. \end{aligned}$$
   - The verifier verifies $\pi_{s,1,r}, \pi_{s,2,r}, \pi_{s,3,r}$ and the following:
   $$E_{s,1,r} \cdot E_{s,2,r} = E_{s,3,r}.$$
2. Multiplication constraint: $\forall i \in [m]$, $e([s_{4,i}(X)], [s_{5,i}(X)]) = e([s_{6,i}(X)], g) \cdot e([A_i(X)], [X^m - 1])$
   - The prover computes $E_{s,4,5,r}, E_{s,6,r}, E_{A,r}$ such that
   $$\begin{aligned} E_{s,4,5,r} &= \langle \mathbf{s}_4^{\mathbf{r}}, \mathbf{s}_5\rangle, \\ E_{s,6,r} &= \langle \mathbf{s}_6, \mathbf{r}\rangle, \\ E_{A,r} &= \langle \mathbf{A}, \mathbf{r}\rangle. \end{aligned}$$

- The prover computes and sends the proof for the following statements of IPA:

$$\pi_{s,4,5,r} : (\mathsf{vk}_{s,4}, \mathsf{vk}_{s,5}, r, E_{s,4}, E_{s,5}, E_{s,4,5,r}) \in \mathcal{L}^m_{\mathsf{IPA}},$$
$$\pi_{s,6,r} : (\mathsf{vk}_{s,6}, r, E_{s,6}, E_{s,6,r}) \in \mathcal{L}^m_{\mathsf{EXP}},$$
$$\pi_{A,r} : (\mathsf{vk}_A, r, E_A, E_{A,r}) \in \mathcal{L}^m_{\mathsf{EXP}}.$$

- The verifier verifies $\pi_{s,4,5,r}, \pi_{s,6,r}, \pi_{A,r}$ and the following:

$$E_{s,4,5,r} = e(E_{s,6,r}, g) \cdot e(E_{A,r}, [X^m - 1])$$

3. Dynamo proof for $t \in [1,6], i \in [m]$, we only consider the first equation here (the rest follows the same technique) $e([u_{t,i}], [z_{t,i}]) = e([\alpha_{t,i}], [X^m - 1]) \cdot e([\beta_{t,i}], [X]) \cdot e([h_{t,i}], g^{1/m})$:
   - The prover computes $E_{u,z,t,r}, E_{\alpha,t,r}, E_{\beta,t,r}, E_{h,t,r}$ such that

$$E_{u,z,t,r} = \langle \mathbf{u}_t, \mathbf{z}_t \rangle, \quad E_{\alpha,t,r} = \langle \alpha_t, \mathbf{r} \rangle,$$
$$E_{\beta,t,r} = \langle \beta, \mathbf{r} \rangle, \quad E_{h,t,r} = \langle \mathbf{h}_t, \mathbf{r} \rangle .$$

   - The prover computes and sends the proof for the following statements of IPA:

$$\pi_{u,z,t,r} : (\mathsf{vk}_{u,t}, \mathsf{vk}_{z,t}, r, E_{u,t}, E_{z,t}, E_{u,z,t,r}) \in \mathcal{L}^m_{\mathsf{IPA}},$$
$$\pi_{\alpha,t,r} : (\mathsf{vk}_{\alpha,t}, r, E_{\alpha,t}, E_{\alpha,t,r}) \in \mathcal{L}^m_{\mathsf{EXP}},$$
$$\pi_{\beta,t,r} : (\mathsf{vk}_{\beta,t}, r, E_{\beta,t}, E_{\beta,t,r}) \in \mathcal{L}^m_{\mathsf{EXP}},$$
$$\pi_{h,t,r} : (\mathsf{vk}_{h,t}, r, E_{h,t}, E_{h,t,r}) \in \mathcal{L}^m_{\mathsf{EXP}}.$$

   - The verifier verifies $\pi_{u,z,t,r}, \pi_{\alpha,t,r}, \pi_{\beta,t,r}, \pi_{h,t,r}$ and the following:

$$E_{u,z,t,r} = e(E_{\alpha,t,r}, [X^m - 1]) \cdot e(E_{\beta,t,r}, [X]) \cdot e(E_{h,t,r}, g^{1/m}) .$$

4. Consistency proof is similar to the Dynamo proof.
5. Input consistency $[h_{\mathbb{x}}(Y)] \cdot \prod_{t=1}^{6} \prod_{i=1}^{m} [h_{t,i}(Y)] = 1_{\mathbb{G}}$:
   - The prover computes and sends the following for all $t \in [1,6]$:

$$E_{h,t,1} = \prod_{i \in [m]} h_{t,i}(Y) ,$$

$$\pi_{h,t,1} : \quad (\mathsf{vk}_{h,t}, 1, E_{h,t}, E_{h,t,1}) \in \mathcal{L}^m_{\mathsf{EXP}} .$$

   - The verifier verifies $\pi_{h,t,1}$ for all $t \in [1,6]$ and the following:

$$[h_{\mathbb{x}}(Y)] \cdot \prod_{t=1}^{6} E_{h,t,1} = 1_{\mathbb{G}} .$$

Overall, there will be $O(1)$ number of IPA proofs. Thus the proof size and verification time can be reduced to $O(\log n)$.

F.2 IPA for $\mathcal{C}\mathsf{gate}^{(i)}$

In this section, we introduce how to build a protocol for $\mathcal{C}\mathsf{gate}^{(i)}$ using GIPA instead of black-box using another SNARK.

First, we need to translate the objects appeared in $\mathcal{C}\mathsf{gate}^{(i)}$ into vectors suitable for GIPA. For ease of notations, we consider using $\mathbf{z}_L, \mathbf{z}_R, \mathbf{z}_O$ instead of $\mathbf{z}[1:n], \mathbf{z}[n+1:2n], \mathbf{z}[2n+1:3n]$, and all the objects related to $\mathbf{z}$ (including the commitments and proofs) will be naturally separated into three instances of form $(\cdot)_L, (\cdot)_R, (\cdot)_O$. We take a look at each of them in details. WLOG assume that n is a power of two and θ is the n-the root of unity.

Commitments. The new commitments in the public statement will be

$$(c_\Delta)_i, \ldots, (c_\Delta)_\ell, \text{ where } \Delta \in \{L, R, O\}\,.$$

Set of indices. It suffices to use only S which contains all $j \in [n]$ such that at least one of $j, n+j, 2n+j$ is in $S^{(i)}$. If using S, we need to check that for all $j \in S$, $\mathbf{z}_L[j] \cdot \mathbf{z}_R[j] = \mathbf{z}_O[j]$ and $\mathbf{z}_L, \mathbf{z}_R, \mathbf{z}_O$ have zero values outside S, i.e., $z_\Delta[j] = 0$ for all $j \in [n] \setminus S, \Delta \in \{L, R, O\}$.

Let $s = |S|$ and $\mathbf{t} = [t_1 \ldots t_s] \in [n]^s$ as the vector including all the unique indices in S. We use the following vector of group elements to represent such S:

$$\mathbf{S} := ([X - \theta^{t_j}])_{j \in [s]}\,.$$

For the set $[n] \setminus S$, we consider zero sub-trees in the AMT quotient tree of size n. See Fig. 9 for an example. In this case, $n = 8$ and $S = \{1, 6\}, \mathbf{S} = ([X-\theta], [X-\theta^6])$. We use the following vector to represent $[n] \setminus S$, which contains the commitments to all the divisors of the yellow zero sub-trees:

$$\overline{\mathbf{S}} = ([X - \theta^5], [X^2 - \theta^6], [X - \theta^2], [X^2 - 1])\,.$$

Let $s^c = |\overline{\mathbf{S}}|$. Note that $s^c \leq \ell \cdot s$. We can check the validity of $\mathbf{S}$ and $\overline{\mathbf{S}}$ by the following equation:

$$\left(\prod_{j \in [s]} \mathbf{S}[j]\right) \cdot \left(\prod_{j \in [s^c]} \overline{\mathbf{S}}[j]\right) \stackrel{?}{=} [X^n - 1]\,.$$

However, it is not easy for the prover to show this equation efficiently. We can consider opening all the commitments in this equation at a random point α picked by the verifier. In particular, the prover computes vectors

$$\mathbf{S}^\alpha := (f(\alpha))_{[f(X)] \in \mathbf{S}}, \quad \overline{\mathbf{S}}^\alpha := (f(\alpha))_{[f(X)] \in \overline{\mathbf{S}}}\,,$$

and then show that

$$\left(\prod_{j \in [s]} \mathbf{S}^\alpha[j]\right) \cdot \left(\prod_{j \in [s^c]} \overline{\mathbf{S}}^\alpha[j]\right) \stackrel{?}{=} \alpha^n - 1\,,$$

as well as the proof that opening $\mathbf{S}[j], \overline{\mathbf{S}}[j]$ at $X = \alpha$ can get $\mathbf{S}^{\alpha}[j], \overline{\mathbf{S}}^{\alpha}[j]$. To show that opening $[f(X)] = [X^{2^k} - t]$ at $X = \alpha$ gets $v = \alpha^{2^k} - t$, we can show there exists $d_0, \ldots, d_{\ell-1} \in \{0,1\}$ such that

$$f(X) - v = \sum_{k=0}^{\ell-1} d_k \cdot (X^{2^k} - \alpha^{2^k}) .$$

Thus, the prover can compute $\mathbf{D}^k \in \{0,1\}^s$, $\overline{\mathbf{D}}^k \in \{0,1\}^{s^c}$ such that

$$\mathbf{S}[j]/g^{\mathbf{S}^{\alpha}[j]} = \prod_{k=0}^{\ell-1} [X^{2^k} - \alpha^{2^k}]^{\mathbf{D}^k[j]}, \quad \forall j \in [s] ,$$

$$\overline{\mathbf{S}}[j]/g^{\overline{\mathbf{S}}^{\alpha}[j]} = \prod_{k=0}^{\ell-1} [X^{2^k} - \alpha^{2^k}]^{\overline{\mathbf{D}}^k[j]}, \quad \forall j \in [s^c] ,$$

Values. We have the following vectors for values:

$$\mathbf{v}_{\Delta} = (\mathbf{z}_{\Delta}[t_j])_{j \in [s]}, \text{ where } \Delta \in \{L, R, O\} .$$

The equation to check all the gates in S will become

$$\mathbf{v}_L[j] \cdot \mathbf{v}_R[j] = \mathbf{v}_O[j], \quad \forall j \in [s] .$$

The AMT proofs. The new proofs should include the proof of $\mathbf{v}_{\Delta}$ values for $\prod_{k=i}^{\ell} (c_{\Delta})_k$ and the proof of zeros at $[n] \setminus S$ for $(c_{\Delta})_i$. In particular, for each $\Delta \in \{L, R, O\}$ and $k \in [\ell]$, denote

$$\mathbf{p}_{\Delta}^k = ([(q_{\Delta})_{j,k}(X)])_{j \in [s]} ,$$

$$\overline{\mathbf{p}}_{\Delta}^k = ([(\overline{q}_{\Delta})_{j,k}(X)])_{j \in [s^c]} ,$$

such that

$$(\mathbf{p}_{\Delta}^1[j], \mathbf{p}_{\Delta}^2[j], \ldots, \mathbf{p}_{\Delta}^{\ell}[j])$$

is the AMT proof of $\mathbf{v}_{\Delta}[j]$, and

$$(\overline{\mathbf{p}}_{\Delta}^1[j], \overline{\mathbf{p}}_{\Delta}^2[j], \ldots, \overline{\mathbf{p}}_{\Delta}^{\ell}[j])$$

is one of the proof of zeros (could be padded to ℓ group elements by adding 0s).

Let $\mathbf{A}^k = ([X^{n/2^k} - (\theta^{t_j})^{n/2^k}])_{j \in [s]}$, then the equation for AMT verification of $\mathbf{v}_{\Delta}[j]$ would be

$$e\left(\prod_{k=i}^{\ell} (c_{\Delta})_k, g\right) = e(g^{\mathbf{v}_{\Delta}[j]}, g) \cdot \prod_{k \in [\ell]} e\left(\mathbf{p}_{\Delta}^k[j], \mathbf{A}^k[j]\right) .$$

If we want to use this equation to verify the AMT proofs, we need to show the validity of $\mathbf{A}^k$. One natural way to do that is to show that there is another vector $\mathbf{B}^k$, defined as

$$\mathbf{B}^k = \left(\left[\frac{X^{n/2^k} - (\theta^{t_j})^{n/2^k}}{X - \theta^{t_j}} \right] \right)_{j \in [s]},$$

such that $e(\mathbf{B}^k[j], \mathbf{S}[j]) = e(\mathbf{A}^k[j], g)$ for each $j \in [s]$. Similarly, for proof of zeros, define $\overline{\mathbf{A}}^k$ and $\overline{\mathbf{B}}^k$ such that

$$e((c_\Delta)_i, g) = \prod_{k \in [\ell]} e\left(\overline{\mathbf{p}}^k_\Delta[j], \overline{\mathbf{A}}^k[j]\right), \quad \forall j \in [s^c],$$

$$e(\overline{\mathbf{B}}^k[j], \overline{\mathbf{S}}[j]) = e(\overline{\mathbf{A}}^k[j], g).$$

Note that $\overline{\mathbf{A}}^k[j]$ should have degree not less than $\overline{\mathbf{S}}[j]$. If $n/2^k$ is less than the degree of $\overline{\mathbf{S}}[j]$, then let $\overline{\mathbf{A}}^k[j] = \overline{\mathbf{S}}[j]$.

Wrapping up. We put all the necessary equations mentioned above in Fig. 11. To show for Eq. (16), we can consider the prefix product trick widely used in permutation checking [16,12,29].This would require $O(s \log s + s^c \log s^c)$ prover time and $O(1)$ proof size. To show for other equations, the prover can commit to all the vectors of size s and s^c, and build IPAs using the same technique from the previous subsection. This will result in $O(\ell)$ commitments and $O(\ell)$ IPA instances of both size s and size s^c. In summary, the prover time to show for all the equations in $O(s \cdot \ell^2)$ time and the proof size will be $O(\ell \log s)$.

G Universal Dynamo using random oracle

In this section we show how our Dynamo protocol can be turned into a universal one, introducing an $\mathcal{I}$ algorithm—see Definition 3. A natural way to do that is to compute the verifier key $[u]$ and update keys for v, α, β and B in $\mathcal{I}$. However this would require the trusted universal setup algorithm $\mathcal{G}$ to output keys of size $O(m^2)$ in pp, e.g., all the commitments

$$\{[L_i(X)L_j(Y)]\}_{i,j \in [m]}$$

so that we can encode an arbitrary permutation σ. We can overcome the quadratic complexity with the use of the *random oracle* function $\mathsf{H} : \{0,1\}^* \to \mathbb{F}$ which is an additional assumption that we are required to make in order to achieve universality. The central idea is to use a random point to replace variable Y. Following our starting point in the beginning of this section, it is easy to see that $([m, \sigma], (\mathsf{com}(\mathbf{z}), \mathsf{com}(\mathbf{h})), (\mathbf{z}, \mathbf{h})) \in \mathcal{R}^r_{\mathcal{P}}$ if and only if

$$\sum_{i \in [m]} (\mathbf{z}[i] - \mathbf{h}[i]) \cdot L_i(Y) = \sum_{i \in [m]} \mathbf{z}[i] \cdot L_{\sigma^{-1}(i)}(Y). \tag{24}$$

– Check the validity of $\mathbf{S} \in \mathbb{G}^s, \overline{\mathbf{S}} \in \mathbb{G}^{s^c}$ through a random α picked by the verifier, $\mathbf{S}^\alpha \in \mathbb{F}^s, \overline{\mathbf{S}}^\alpha \in \mathbb{F}^{s^c}$, and $\mathbf{D}^k \in \mathbb{F}^s, \overline{\mathbf{D}}^k \in \mathbb{F}^{s^c}$ for $k \in [0, \ell - 1]$:

$$\left(\prod_{j\in[s]} \mathbf{S}^\alpha[j]\right) \cdot \left(\prod_{j\in[s^c]} \overline{\mathbf{S}}^\alpha[j]\right) = \alpha^n - 1 \tag{16}$$

$$\mathbf{S}[j]/g^{\mathbf{S}^\alpha[j]} = \prod_{k=0}^{\ell-1} [X^{2^k} - \alpha^{2^k}]^{\mathbf{D}^k[j]}, \quad \forall j \in [s] \tag{17}$$

$$\overline{\mathbf{S}}[j]/g^{\overline{\mathbf{S}}^\alpha[j]} = \prod_{k=0}^{\ell-1} [X^{2^k} - \alpha^{2^k}]^{\overline{\mathbf{D}}^k[j]}, \quad \forall j \in [s^c] \tag{18}$$

– Check the AMT proofs $\mathbf{p}_\Delta^k \in \mathbb{G}^s$, $\overline{\mathbf{p}}_\Delta^k \in \mathbb{G}^{s^c}$ for each $k \in [\ell], \Delta \in \{L, R, O\}$:

$$e\left(\prod_{k=i}^{\ell} (c_\Delta)_k, g\right) = e(g^{\mathbf{v}_\Delta[j]}, g) \cdot \prod_{k\in[\ell]} e\left(\mathbf{p}_\Delta^k[j], \mathbf{A}^k[j]\right) \quad \forall j \in [s] \tag{19}$$

$$e((c_\Delta)_i, g) = \prod_{k\in[\ell]} e\left(\overline{\mathbf{p}}_\Delta^k[j], \overline{\mathbf{A}}^k[j]\right), \quad \forall j \in [s^c] \tag{20}$$

– Check the validity of $\mathbf{A}^k \in \mathbb{G}^s$, $\overline{\mathbf{A}}^k \in \mathbb{G}^{s^c}$ using $\mathbf{B}^k \in \mathbb{G}^s$, $\overline{\mathbf{B}}^k \in \mathbb{G}^{s^c}$ for $k \in [\ell]$:

$$e(\mathbf{B}^k[j], \mathbf{S}[j]) = e(\mathbf{A}^k[j], g), \ \forall j \in [s^c] \tag{21}$$

$$e(\overline{\mathbf{B}}^k[j], \overline{\mathbf{S}}[j]) = e(\overline{\mathbf{A}}^k[j], g), \ \forall j \in [s^c] \tag{22}$$

– Check the validity of $\mathbf{v}_L, \mathbf{v}_R, \mathbf{v}_O \in \mathbb{G}^s$:

$$\mathbf{v}_L[j] \cdot \mathbf{v}_R[j] = \mathbf{v}_O[j], \ \forall j \in [s] \tag{23}$$

Fig. 11. The equations to check for $\mathcal{C}\mathsf{gate}^{(i)}$.

To prove the public statement $(\mathsf{com}(\mathbf{z}), \mathsf{com}(\mathbf{h}))$, we will have a prover provide a proof that, when evaluated Y at a random point $r \in \mathbb{F}$, the above equation holds. Now we can avoid the use of variable Y and pick $\mathsf{com}(\mathbf{z}), \mathsf{com}(\mathbf{h})$ as the KZG commitments to $z(X), h(X)$ which encodes $\mathbf{z}, \mathbf{h}$ respectively using Lagrange interpolation.

Computing the proof. First, given KZG commitments $[z]$ and $[h]$ to vectors $\mathbf{z}$ and $\mathbf{h}$ (which are the public statement) the prover computes a random point r using a random oracle, i.e., $r = \mathsf{H}([z]||[h])$. Then the prover provides the following proofs.

1. A commitment to $v(X)$ which encodes $(\mathbf{z}[i] - \mathbf{h}[i]) \cdot L_i(r)$ for all $i \in [m]$, namely
$$v(X) = \sum_{i \in [m]} L_i(X) \cdot (\mathbf{z}[i] - \mathbf{h}[i]) \cdot L_i(r)\,.$$
To prove $v(X)$ is well-formed we observe that $v(\omega^i) = (z(\omega^i) - h(\omega^i)) \cdot \frac{\omega^i(r^m-1)}{m(r-\omega^i)}$ and therefore it is sufficient to check that
$$m \cdot v(\omega^i)(r - \omega^i) = (z(\omega^i) - h(\omega^i)) \cdot \omega^i(r^m - 1), \text{ for all } i \in [m]\,.$$
We can do that through the standard zero check
$$m \cdot v(X)(r - X) - (z(X) - h(X))X(r^m - 1) = \beta(X)(X^m - 1)\,.$$
2. A commitment to $v_\sigma(X)$ which encodes $\mathbf{z}[i] \cdot L_{\sigma^{-1}(i)}(r)$ ($v_\sigma(\omega^i) = \mathbf{z}[i] \cdot L_{\sigma^{-1}(i)}(r)$ for all $i \in [m]$), and the correctness proof for $v_\sigma(X)$. Similarly from the definition of Lagrange polynomials we have $v_\sigma(\omega^i) = z(\omega^i) \cdot \omega^{\sigma^{-1}(i)}(r^m - 1)/m(r - \omega^{\sigma^{-1}(i)})$. Let now $t(X)$ be a polynomial (computed in algorithm $\mathcal{I}$) such that $t(\omega^i) = \omega^{\sigma^{-1}(i)}$ for all $i \in [m]$. The commitment to $t(X)$ is provided as a verifier key. Then it is sufficient to check that
$$m \cdot v_\sigma(\omega^i)(r - t(\omega^i)) = z(\omega^i)t(\omega^i)(r^m - 1), \text{ for all } i \in [m]$$
through the standard zero check
$$m \cdot v_\sigma(X)(r - t(X)) - z(X)t(X)(r^m - 1) = \beta_\sigma(X)(X^m - 1)\,.$$
3. The proof for $\sum_{i \in [m]} v(\omega^i) - \sum_{i \in [m]} v_\sigma(\omega^i) = 0$. Similar as before, we use the univariate sum-check by showing that (i) $v(X), v_\sigma(X)$ have degree at most $m - 1$ over X; (ii) $v(0) - v_\sigma(0) = 0$ through opening $v(X) - v_\sigma(X)$ at $X = 0$ using the following quotient polynomial
$$\gamma(X) = \frac{v(X) - v_\sigma(X)}{X}\,.$$

The final universal Dynamo proof consists of 5 KZG commitments to $v(X)$, $\beta(X)$, $v_\sigma(X)$, $\beta_\sigma(X)$ and $\gamma(X)$ as defined above as well as additional KZG commitments to $V(X), V_\sigma(X)$ for degree check of $v(X), v_\sigma(X)$ (both with degree at most $m - 1$). We now have the following theorem for efficiency and main theorems (Theorem 3 and Lemma 1), whose proof can be found in Sections I and J.

– $\mathcal{G}(1^\lambda, m) \to \mathsf{pp}$:
  - Pick random $\tau_X \in \mathbb{F}$ for variable X.
  - Set $\mathsf{pp} = \{[X], \ldots, [X^m]\}$.

– $\mathcal{I}(\mathsf{pp}, [m, \sigma]) \to (\mathsf{pk}, \mathsf{vk})$:
  - Set $\mathsf{pk} = \{[L_i], [L_i(X) \cdot X], [\beta_i(X)], [(\beta_{\sigma,i}(X)], [\gamma_i(X)]\}_{i\in[m]}$ as in proof of Theorem 13.
  - Set $\mathsf{vk} = \{[t(X)], [X], [X^m]\}$, where $t(X) = \sum_{i\in[m]} L_i(X) \cdot \omega^{\sigma^{-1}(i)}$.

– $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to \pi$:
  - Parse $\mathbb{x}$ as $[z]$ and $[h]$ and $\mathbb{w}$ as $\mathbf{z}[1], \ldots, \mathbf{z}[m]$ and $\mathbf{h}[1], \ldots, \mathbf{h}[m]$.
  - Compute $r = \mathsf{H}([z]||[h])$.
  - Following Theorem 13, output π as the commitments to each $f \in \mathcal{F} = \{v, \beta, v_\sigma, \beta_\sigma, \gamma, V, V_\sigma\}$.

– $\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$:
  - Parse vk as $\{[t], [X], [X^m]\}$, $\mathbb{x}$ as $[z]$ and $[h]$ and π as commitments to all $f \in \mathcal{F}$.
  - Compute $r = \mathsf{H}([z]||[h])$.
  - Output 1 if and only if all the following relations hold:

$$e([v]^m, [r - X]) = e([z - h], [X]^{r^m - 1}) \cdot e([\beta], [X^m - 1]).$$
$$e([v_\sigma]^m, [r - t]) = e([z], [t]^{r^m - 1}) \cdot e([\beta_\sigma], [X^m - 1]).$$
$$e([v - v_\sigma], g) = e([\gamma], [X]).$$
$$e([v], [X]]) = e([V], g).$$
$$e([v_\sigma], [X]]) = e([V_\sigma], g).$$

Fig. 12. The universal Dynamo SNARK.

Theorem 13 (Efficient computation in universal Dynamo). *Let $\mathcal{F} = \{v, \beta, v_\sigma, \beta_\sigma, \gamma, V, V_\sigma\}$ be the set of seven polynomials contained in the* Dynamo *proof. Every $f \in \mathcal{F}$ can be computed in $O(m')$ time where m' is the number of non-zeros in $\mathbf{z}$.*

H Dynavold: A dynamic zk-SNARK with $O(\sqrt{n} \log n)$ update time

In this section, we present Dynavold, our first general-purpose dynamic zk-SNARK (i.e., for $\mathcal{R}_\mathcal{C}$ and index $\mathbb{i} = [n, n_0, \sigma]$). Dynavold is using the Dynamo SNARK from Section 4. Dynavold has $O(n)$ setup time, $O(k \cdot \sqrt{n} \log n)$ update time (where k is the Hamming distance between the statements) and $O(1)$ proof size. We note here that Dynavold's update algorithm is trivially parallelizable.

Our main technique: Enforcing gate constraints on subvectors. To address the linear update time of multiplication gate constraints, we follow a natural approach. We divide each vector $\mathbf{s}_t$ into the $\ell = \sqrt{n}$ successive subvectors $\mathbf{s}_{t,1}, \ldots, \mathbf{s}_{t,\ell}$ of $\sqrt{n}$ values each, with the goal being that an update will be addressed by only working on a $\sqrt{n}$-size subvector. To do that, the prover will

provide ℓ polynomial commitments for each vector $\mathbf{s}_t$ for a total of $6 \cdot \ell$ commitments, i.e., the commitments $[s_{t,1}(X)], \ldots, [s_{t,\ell}(X)]$ for $t = 1, \ldots, 6$. Importantly, for these commitments we are using the ℓ-th root of unity φ. Specifically, for $t \in [1, 6], i \in [\ell]$, $s_{t,i}(X)$ is a degree-$(\ell - 1)$ polynomial such that

$$s_{t,i}(\varphi^j) = \mathbf{s}_{t,i}[j], \quad j \in [\ell]\,.$$

To prove the multiplication constraint, the prover must now provide ℓ commitments to the following quotient polynomials

$$A_i(X) = \frac{s_{4,i}(X) \cdot s_{5,i}(X) - s_{6,i}(X)}{X^\ell - 1} \text{ for } i = 1, \ldots, \ell\,. \tag{25}$$

Enforcing copy constraints across subvectors. Note now that the final task to be performed by the prover is to convince the verifier that $[s_{t,1}(X)], \ldots, [s_{t,\ell}(X)]$, for $t = 1, \ldots, 6$, are consistent with σ and $\mathbf{s}_7 = \mathbb{x}$. To do that, we consider the following decomposition of $\mathbf{z}$, i.e.,

$$\mathbf{z} = \sum_{t=1}^{6} \sum_{i=1}^{\ell} \mathbf{z}_{t,i} + \mathbf{z}_7$$

such that

- For all $t = 1, \ldots, 6$ and for $i \in [\ell]$, $\mathbf{z}_{t,i} \in \mathbb{F}^m$ and $\mathbf{z}_7 \in \mathbb{F}^m$.
- For all $t = 1, \ldots, 6$ and for all $i \in [\ell]$, $\mathbf{z}_{t,i}[(t-1)n + (i-1)\ell + j] = \mathbf{s}_{t,i}[j]$ for $j \in [\ell]$. For all other locations, $\mathbf{z}_{t,i}$ has 0 values.
- $\mathbf{z}_7[6n + j] = \mathbf{s}_7[j]$ for $j \in [n_0]$. For all other locations, $\mathbf{z}_7$ has 0 values.

Now, for each $\mathbf{z}_{t,i} \in \mathbb{F}^m$ we can compute $\mathbf{h}_{t,i} \in \mathbb{F}^m$ such that

$$([m, \sigma], (\mathsf{com}(\mathbf{z}_{t,i}), \mathsf{com}(\mathbf{h}_{t,i})), (\mathbf{z}_{t,i}, \mathbf{h}_{t,i})) \in \mathcal{R}^r_{\mathcal{P}}\,.$$

By Lemma 2, if we also have that $\sum_{t=1}^{6} \sum_i \mathbf{h}_{t,i} = \mathbf{0}$, this will mean that $([m, \sigma], \mathsf{com}(\mathbf{z}), \mathbf{z}) \in \mathcal{R}_{\mathcal{P}}$. Therefore it is enough to provide Dynamo proofs for $((\mathsf{com}(\mathbf{z}_{t,i}), \mathsf{com}(\mathbf{h}_{t,i})), (\mathbf{z}_{t,i}, \mathbf{h}_{t,i}))$ for all $t = 1, \ldots, 6$ and $i = 1, \ldots, \ell$ and just check that the $\mathbf{h}$'s sum to $\mathbf{0}$. Recall that we use $[z_{t,i}(X)]$ for $\mathsf{com}(\mathbf{z}_{t,i})$ where $z_{t,i}(X)$ satisfies $z_{t,i}(\omega^j) = \mathbf{z}_{t,i}[j]$ for all $j \in [m]$.

Final step: Showing consistency between $[s_{t,i}]$ and $[z_{t,i}]$. Note that the Dynamo SNARK provides relaxed permutation proofs for commitments $\mathsf{com}(\mathbf{z}_{t,i}) = [z_{t,i}]$ to vectors $\mathbf{z}_{t,i} \in \mathbb{F}^m$, but checking the gate relationships work on commitments $[s_{t,i}]$ to vectors $\mathbf{s}_{t,i} \in \mathbb{F}^\ell$. Both commitments commit to the same elements in $\mathbb{F}$, but with different polynomials. In particular we want to show that $(([z_{t,i}], [s_{t,i}]), (z_{t,i}, s_{t,i})) \in \mathcal{R}^{(0)}_{m,\ell,(t-1)n+(i-1)\ell+1}$ where $\mathcal{R}^{(0)}_{m,\ell,k}$ is defined as

$$\mathcal{R}^{(0)}_{m,\ell,k} = \left\{ \begin{array}{ll} (\mathbb{x} = ([a], [b]), & a(\omega^j) = b(\varphi^{j-k+1}),\ \forall j \in [k, k+\ell-1]\,, \\ \ \mathbb{w} = (a(X), b(X))) & a(\omega^j) = 0,\ \forall j \in [m] \setminus [k, k+\ell-1] \end{array} \right\} : \left\} \right.\,.$$

– $\mathcal{G}(1^\lambda, [n, n_0, \sigma]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - Set $m = 6n + n_0$ and $\ell = \sqrt{n}$.
  - Pick random τ_X, τ_Y from $\mathbb{F}$ for X and Y respectively.
  - Call $\mathcal{D}.\mathcal{G}_{\tau_X,\tau_Y}(1^\lambda, [m, \sigma]) \to (\mathsf{pk}^{\mathcal{D}}, \mathsf{vk}^{\mathcal{D}})$.
  - Call $\mathcal{S}.\mathcal{G}_{\tau_X,\tau_Y}(1^\lambda, [m, \ell, (t-1)n + (i-1)\ell + 1]) \to (\mathsf{pk}^{\mathcal{S}}_{t,i}, \mathsf{vk}^{\mathcal{S}}_{t,i})$ for $t \in [1, 6]$, $i \in [\ell]$.
  - Set $\mathsf{upk} = \mathsf{pk} = \{[X^i]_i, \mathsf{pk}^{\mathcal{D}}, \{\mathsf{pk}^{\mathcal{S}}_{t,i}\}_{t,i}\}$ and $\mathsf{vk} = \{\mathsf{vk}^{\mathcal{D}}, \{\mathsf{vk}^{\mathcal{S}}_{t,i}\}_{t,i}, [L_i(Y)]_i, [X^\ell - 1]\}$.

– $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{aux})$:
  - Parse $\mathbb{x}$ as $\mathbf{s}_7$ and $\mathbb{w}$ as $\{\mathbf{s}_{t,1}, \ldots, \mathbf{s}_{t,\ell}\}_{t=1,\ldots,6}$.
  - For $t = 1, \ldots, 6$, for $i = 1, \ldots, \ell$ do the following.
    - Compute vectors $\mathbf{z}_{t,i}$ and $\mathbf{h}_{t,i} \in \mathbb{F}^m$.
    - Compute the corresponding KZG commitments $[z_{t,i}]$, $[s_{t,i}]$ and $[h_{t,i}]$.
    - Call $\mathcal{D}.\mathcal{P}(\mathsf{pk}^{\mathcal{D}}, ([z_{t,i}], [h_{t,i}]), (\mathbf{z}_{t,i}, \mathbf{h}_{t,i})) \to \pi^{\mathcal{D}}_{t,i}$.
    - Call $\mathcal{S}.\mathcal{P}(\mathsf{pk}^{\mathcal{S}}_{t,i}, ([z_{t,i}], [s_{t,i}]), (z_{t,i}, s_{t,i})) \to \pi^{\mathcal{S}}_{t,i}$.
  - For $i = 1, \ldots, \ell$, compute the commitments $[A_i(X)]$ as in Eq. (25).
  - Proof π contains $\{[s_{t,i}], [z_{t,i}], [h_{t,i}], \pi^{\mathcal{D}}_{t,i}, \pi^{\mathcal{S}}_{t,i}\}_{t,i}$ and $\{[A_i(X)]\}_i$. Set $\mathsf{aux} = \mathbf{z}$.

– $\mathcal{U}(\mathsf{upk}, \mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}, \pi, \mathsf{aux}) \to (\pi', \mathsf{aux}')$:
  - Parse $(\mathbb{x}, \mathbb{w})$ as $\mathbf{z}$ and $(\mathbb{x}', \mathbb{w}')$ as a new valid witness $\mathbf{z}'$.
  - Let C be the set of tuples (t, i) that correspond to updated subvectors $\mathbf{s}_{t,i}$.
  - For every $(t, i) \in C$ call $\mathcal{D}.\mathcal{P}$ to recompute its Dynamo proof $\pi^{\mathcal{D}'}_{t,i}$.
  - For every $(t, i) \in C$, call $\mathcal{S}.\mathcal{P}$ to recompute the consisitency proof $\pi^{\mathcal{S}'}_{t,i}$.
  - Let $I = \{i : (t, i) \in C \wedge t \geq 4\}$. For every $i \in I$ recompute $[A'_i(X)]$ as in Eq. (25).
  - Output new Dynamo and consistency proofs and new $[A'_i(X)]$ as π'. Set $\mathsf{aux}' = \mathbf{z}'$.

– $\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$:
  - Parse π as $\{[s_{t,i}], [z_{t,i}], [h_{t,i}], \pi^{\mathcal{D}}_{t,i}, \pi^{\mathcal{S}}_{t,i}\}_{t,i}$ and $\{[A_i(X)]\}_i$.
  - Check the addition constraints, i.e., that for all $i = 1, \ldots, \ell$ it is

$$[s_{1,i}(X)] \cdot [s_{2,i}(X)] = [s_{3,i}(X)] \,.$$

  - Check the multiplication constraints, i.e., that, for all $i = 1, \ldots, \ell$ it is

$$e([s_{4,i}(X)], [s_{5,i}(X)]) \cdot e([-s_{6,i}(X)], g) = e([A_i(X)], [X^\ell - 1]) \,.$$

  - Check the Dynamo and consistency proofs i.e., for all $t = 1, \ldots, 6$ and $i = 1, \ldots, \ell$ it is

$$\mathcal{D}.\mathcal{V}(\mathsf{vk}^{\mathcal{D}}, ([z_{t,i}], [h_{t,i}]), \pi^{\mathcal{D}}_{t,i}) \to 1 \,, \quad \mathcal{S}.\mathcal{V}(\mathsf{vk}^{\mathcal{S}}_{t,i}, ([z_{t,i}], [s_{t,i}]), \pi^{\mathcal{S}}_{t,i}) \to 1 \,.$$

  - Parse $\mathbb{x}$ as $\mathbf{z}[6n + 1 : 6n + n_0]$. Set $h_{\mathbb{x}}(Y) = \sum_{i=6n+1}^{6n+n_0} \mathbf{z}[i] \left(L_i(Y) - L_{\sigma^{-1}(i)}(Y)\right)$. Check

$$[h_{\mathbb{x}}(Y)] \cdot \prod_{t \in [1,6]} \prod_{i \in [\ell]} [h_{t,i}(Y)] = 1_{\mathbb{G}} \,.$$

Fig. 13. Dynavold dynamic SNARK using Dynamo SNARK $\mathcal{D}$ and SNARK $\mathcal{S}$ for consistency check (Section D) as a black box.

We present a protocol in Section D to show the consistency proof for this relation.

Proof and verification. The final proof consists of $6\sqrt{n}$ Dynamo proofs, $6\sqrt{n}$ consistency proofs and $\sqrt{n}$ commitments to the quotient polynomials from Eq. (25). To verify the final proof the verifier verifies the Dynamo proofs, the consistency proofs and the quotient polynomials A_i. Then the verifier needs to verify consistency between the public statement $\mathbb{x}$ and the final proof. If we use the non-universal version of the Dynamo, then the verifier computes a commitment to the following polynomial corresponding to the public statement, i.e.,

$$h_{\mathbb{x}}(Y) = \sum_{i=6n+1}^{6n+n_0} (\mathbf{z}[i] - \mathbf{z}[\sigma^{-1}(i)]) \cdot L_i(Y) = \sum_{i=6n+1}^{6n+n_0} \mathbf{z}[i] \left(L_i(Y) - L_{\sigma^{-1}(i)}(Y)\right)$$

and checks whether

$$[h_{\mathbb{x}}(Y)] \cdot \prod_{t=1}^{6} \prod_{i=1}^{m} [h_{t,i}(Y)] = 1_{\mathbb{G}} \,,$$

where $[h_{t,i}]$ is in the public statement of the Dynamo instance for the $\mathbf{z}_{t,i}$ vector. See Fig. 17 in Section K for pictorial description of Dynavold. Our final protocol is shown in Fig. 4.

Theorem 14 (Dynavold). *The protocol of Fig. 13 is a dynamic SNARK (per Definition 1) for $\mathcal{R}_{\mathcal{C}}$ based on q-DLOG (see Assumption 1) in the AGM model. Its complexities are as follows.*

1. *$\mathcal{G}$ runs in $O(n + n_0)$ time, outputs* pk *of $O(n)$ size,* vk *of $O(\sqrt{n} + n_0)$ size;*
2. *$\mathcal{P}$ runs in $O(n \log n)$ time and outputs a proof π of $O(\sqrt{n})$ size;*
3. *$\mathcal{U}$ runs in $O(k\sqrt{n} \log n)$ time, where k is the Hamming distance of $\mathbb{w}, \mathbb{w}'$;*
4. *$\mathcal{V}$ runs in $O(n_0 + \sqrt{n})$ time.*

Proof. Completeness and updatability follow naturally from the construction. For knowledge soundness, we can build an extractor by calling the extractor of Dynamo to extract the witness which satisfies the copy constraints. The knowledge soundness of $\mathcal{S}$ and the verification algorithm can also ensure that this extracted witness also satisfies the gate constraints. The complexities of $\mathcal{G}, \mathcal{P}, \mathcal{U}$ and $\mathcal{V}$ follow from the protocol. □

Lemma 14 (Zero-knowledge Dynavold). *There is a zero-knowledge version of Dynavold with the same complexities.*

Removing the consistency check by using a variant of relaxed permutation relation. Consider a variant of relaxed permutation relation with index $\mathbb{i} = [m, \sigma, b, \ell]$, where b, ℓ are parameters such that $b + \ell - 1 \leq m$ and meant to represent the subvector that we are interested in enforcing the permutation relation. This new relation contains those tuples

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = ([m, \sigma, b, \ell], (\mathsf{com}(\mathbf{s}), \mathsf{com}(\mathbf{h})), (\mathbf{s}, \mathbf{h}))$$

where $\mathsf{com}(\mathbf{s})$ and $\mathsf{com}(\mathbf{h})$ are commitments to $\mathbf{s} \in \mathbb{F}^{\ell}$ and $\mathbf{h} \in \mathbb{F}^{m}$ such that there is a vector $\mathbf{z} \in \mathbb{F}^{m}$ satisfying

- $\mathbf{z}[i] = \mathbf{s}[i - b + 1]$ for all $i \in [b, b + \ell - 1]$.
- $\mathbf{z}[i] = 0$ for all $i \in [m] \setminus [b, b + \ell - 1]$.
- $\mathbf{z}[i] = \mathbf{z}[\sigma(i)] + \mathbf{h}[i]$ for all $i \in [m]$.

In fact, we could use a variant of Dynamo to show for this relation, which will have exact the same asymptotics as current approach but allow Dynavoldto be consistency-check free.

Concrete proof size optimization. Recall that in Dynavold, we must run Dynamo for each one of the $6 \cdot \sqrt{n}$ subvectors. One Dynamo instance requires 7 group elements (2 for the public statement and 5 for the proof) and therefore the Dynavold proof has a total of $42 \cdot \sqrt{n}$ group elements. To reduce the proof size, we can provide only 3 Dynamo proofs for addition gates. This allows us to reduce the proof size to $4\sqrt{n} + O(1)$ group elements. See Section K for a figure of illustration.

H.1 Asymptotic reduction of Dynavold proof and verification to $O(1)$ or $O(\log n)$

We can reduce the proof size and verifier time of Dynavold from $O(\sqrt{n})$ to $O(1)$ by using a custom pairing-based argument system by Bünz et al. [5], as follows, increasing the proof size to $O(\log n)$ and introducing the random oracle assumption.

Using a pairing equations argument. Recall that the main bottleneck of Dynavold's verifier is processing $M = 6 \cdot \sqrt{n}$ commitments. For example, for every $t = 1, \ldots, 6$, one part of the Dynavold verification algorithm requires verifying $\sqrt{n}$ Dynamo proofs, where (in the first part of the Dynamo proof) the verifier receives commitments $[\beta_i]$ and $[B_i]$ to polynomials $\beta_i(X)$ and $B_i(X)$ respectively for $i = 1, \ldots, m$ and must check whether

$$e([\beta_i], [X^2]) = e([B_i], g) \text{ for } i = 1, \ldots, \sqrt{n}\,.$$

Note that all equations above hold with overwhelming probability over the random choice of a value r, if and only if, it is

$$e\left(\prod_{i \in [\ell]} [\beta_i \cdot r^i], [X^2]\right) = e\left(\prod_{i \in [\ell]} [B_i \cdot r^i], g\right)\,. \tag{26}$$

However, checking the above relation again requires access to individual β_i and B_i. Bünz et al. [5] propose a protocol to circumvent this problem. In particular, the Bünz et al. protocol allows the verifier (in $\log M$ time), on input a pairing-based commitment $\mathsf{com}(x)$ (of $[[x_1], \ldots, [x_{\sqrt{n}}]]$), E_r and r to be convinced that $E_r = \prod_{i \in [\sqrt{n}]} [x_i \cdot r^i]$. Our final protocol that uses Bünz's protocol as a black box is as follows.

1. The prover commits to $[[\beta_1], \ldots, [\beta_{\sqrt{n}}]]$ and $[[B_1], \ldots, [B_{\sqrt{n}}]]$, and sends commitments $\mathsf{com}(\beta)$ and $\mathsf{com}(B)$ respectively to the verifier.

2. The verifier picks a random $r \in \mathbb{F}$ and sends to the prover.
3. The prover and verifier run two instances of the GIPA protocol from Bünz et al. [5] with first public input $(\mathsf{com}(\beta), E_{\beta,r}, r)$ and first witness $[[\beta_1], \ldots, [\beta_{\sqrt{n}}]]$ and second public input $(\mathsf{com}(B), E_{B,r}, r)$ and second witness $[[B_1], \ldots, [B_{\sqrt{n}}]]$, allowing the verifier to check the validity of $E_{\beta,r}$ and $E_{B,r}$ and perform the final verification of Eq. (26).

As usual, it is easy to turn the protocol above into non-interactive via the Fiat-Shamir trick, introducing, however, the random oracle assumption. In addition the pairing protocol also requires an additional assumption—the q-ASDBP assumption, see [5]. All other equations in the verification algorithm of Dynavold can be cast into the above framework, allowing us to achieve the same asymptotic savings—see Section F.

I Security proofs for main theorems

I.1 Proof of Theorem 1

Proof. All the above f_i can be directly calculated from the definition of f. In particular,

- $Z_i(X,Y) = Y^m \cdot L_i(X)$;
- $H_i(X,Y) = X^m \cdot \big(L_i(Y) - L_{\sigma^{-1}(i)}(Y)\big)$;
- $\alpha_i(X,Y) = \frac{L_i(X)(\mathbf{y}_i(Y) - u(X,Y))}{X^m - 1}$;
- $\beta_i(X,Y) = \frac{\mathbf{y}_i(Y)(L_i(X) - L_i(0))}{X}$;
- $B_i(X,Y) = \mathbf{y}_i(Y)(L_i(X) - L_i(0)) \cdot X$.

□

I.2 Proof of Theorem 2

Proof. Completeness follows naturally from the construction and efficient proof computation follows from Theorem 1. The complexities of $\mathcal{P}$ and $\mathcal{V}$ follow naturally from the protocol. For the runtime of $\mathcal{G}$, since this algorithm has access to a and b, it can compute everything in linear time.

For knowledge soundness, we construct the following extractor $\mathcal{E}_{\mathcal{A}}(\mathsf{pk}, \mathsf{vk})$ outputting a witness $\mathbb{w}$ and show that, under the q-DLOG assumption in the AGM, if the verifier accepts, then $\mathbb{w}$ is a valid witness for $\mathbb{x}$ output by $\mathcal{A}$.

1. Run the algebraic adversary $(\mathbb{x}, \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk})$.
2. Let $\mathbb{x} = ([z], [h])$. Note that since $\mathcal{A}$ is algebraic, it also outputs vectors of scalars showing how $[z], [h]$ can be computed from $(\mathsf{pk}, \mathsf{vk})$. $\mathcal{E}_{\mathcal{A}}$ reconstructs $\widetilde{z}(X), \widetilde{h}(Y)$ such that $[z] = [\widetilde{z}(X)], [h] = [\widetilde{h}(Y)]$. Abort if $\deg_Y \widetilde{z} > 0$ or $\deg_X \widetilde{h} > 0$.
3. Output $\mathbb{w} = (\mathbb{w}_z, \mathbb{w}_h)$ where $\mathbb{w}_z[i] = \widetilde{z}(\omega^i), \mathbb{w}_h[i] = \widetilde{h}(\omega^i)$, $\forall i \in [m]$.

Clearly, the extractor runs in polynomial time. To conclude the proof we must show that $\mathbb{w}$ satisfies the permutation σ encoded in $u(X,Y)$. Parse $\pi = \{[f]\}_{f\in\mathcal{F}}$. Since $\mathcal{A}$ is algebraic, it also outputs vectors of scalars to show how $[f]$ can be computed from $(\mathsf{pk},\mathsf{vk})$, and then we can reconstruct $\{\widetilde{f}(X,Y)\}_{f\in\mathcal{F}}$ such that $[f] = [\widetilde{f}(X,Y)]$.

Since the second check in $\mathcal{V}$ passes (see Fig. 3), according to Lemma 9, assuming q-DLOG, the probability that $\deg_Y \widetilde{z} > 0$ is negligible, and thus $\mathcal{E}_{\mathcal{A}}$ will only abort with negligible probability. Now we can write $\widetilde{z}(X,Y)$ as $\widetilde{z}(X)$. Similarly from the last two checks in $\mathcal{V}$, we know that with overwhelming probability, $\widetilde{h}(X,Y)$ has degree 0 on X so we can write it as $\widetilde{h}(Y)$, and $\widetilde{\beta}(X,Y)$ has degree at most $m-2$ on X so $\widetilde{\beta}(X,Y)\cdot X$ has degree at most $m-1$ on X.

From the first check by $\mathcal{V}$ in Fig. 3, according to Lemma 8, assuming q-DLOG, the following equation

$$u(X,Y)\widetilde{z}(X) = \widetilde{\alpha}(X,Y)(X^m-1) + \widetilde{\beta}(X,Y)\cdot X + \frac{1}{m}\cdot\widetilde{h}(Y) \tag{27}$$

holds with overwhelming probability.

From Eq. (27), we have

$$u(\omega^i,Y)\cdot\widetilde{z}(\omega^i) - \frac{1}{m}\cdot\widetilde{h}(Y) = \widetilde{\beta}(\omega^i,Y)\cdot\omega^i\,. \tag{28}$$

The polynomial $\widetilde{\beta}(X,Y)\cdot X$ with degree at most $m-1$ on X now has evaluations $u(\omega^i,Y)\cdot\widetilde{z}(\omega^i) - \widetilde{h}(Y)/m$ when evaluating X on the roots of unity. Therefore, we can write $\widetilde{\beta}(X,Y)\cdot X$ as

$$\widetilde{\beta}(X,Y)\cdot X = \sum_{i\in[m]} L_i(X)\cdot\left(u(\omega^i,Y)\cdot\widetilde{z}(\omega^i) - \frac{1}{m}\cdot\widetilde{h}(Y)\right).$$

Evaluating this equation on $X=0$, we have

$$0 = \frac{1}{m}\sum_{i\in[m]}\left(u(\omega^i,Y)\cdot\widetilde{z}(\omega^i) - \frac{1}{m}\cdot\widetilde{h}(Y)\right),$$

$$\widetilde{h}(Y) = \sum_{i\in[m]} u(\omega^i,Y)\cdot\widetilde{z}(\omega^i) = \sum_{i\in[m]}\mathbf{y}_i(Y)\widetilde{z}(\omega^i)\,.$$

Note that $u(\omega^i,Y)$ and $\mathbf{y}_i(Y)$ are predefined with degree at most $m-1$ on Y, so is $\widetilde{h}(Y)$. Thus

$$\sum_{i\in[m]}\mathbf{y}_i(Y)\widetilde{z}(\omega^i) = \widetilde{h}(Y) = \sum_{i\in[m]}\widetilde{h}(\omega^i)\cdot L_i(Y)$$

which means the output $\mathbb{w}$ of $\mathcal{E}_{\mathcal{A}}$ should be a valid witness. □

I.3 Proof of Theorem 13

Proof. We consider each $f \in \mathcal{F}$ in details:

- The commitments to $v(X), V(X), v_\sigma(X), V_\sigma(X)$. $[v], [V], [v_\sigma], [V_\sigma]$ can be computed directly from $[L_i(X)]$ and $[X \cdot L_i(X)]$ in $O(m')$ time. Note that the honestly computed $\mathbf{h}$ has at most $2m'$ non-zero elements.
- The commitments to $\gamma(X)$. Provide the commitments to the following polynomials in the prover keys:

$$\gamma_i(X) = \frac{L_i(X) - L_i(0)}{X}\,.$$

- The commitment to $\beta(X), \beta_\sigma(X)$. For each $i \in [m]$, provide the commitment to the quotient polynomials $\beta_i(X), \beta_{\sigma,i}(X)$ from the polynomial divisions below:

$$L_i(X) \cdot X = \beta_i(X)(X^m - 1) + r_i(X)\,.$$
$$L_i(X) \cdot t(X) = (\beta_\sigma)_i(X)(X^m - 1) + (r_\sigma)_i(X)\,.$$

  Then $\beta(X), \beta_\sigma(X)$ can be written as

$$\beta(X) = \sum_{i\in[m]} (\mathbf{z}[i] - \mathbf{h}[i]) \cdot (-mL_i(r) + 1 - r^m)\beta_i(X)\,,$$

$$\beta_\sigma(X) = \sum_{i\in[m]} \mathbf{z}[i] \cdot (-mL_{\sigma^{-1}(i)}(r) + 1 - r^m)(\beta_{\sigma,i}(X)\,,$$

  whose commitment can be computed in $O(m' + \log m)$ time (including the computation for $L_i(r), L_{\sigma^{-1}(i)}(r)$ and r^m).

□

I.4 Proof of Theorem 3

Proof. Completeness and complexities follow naturally from the construction and Theorem 13. For knowledge soundness, we use the following extractor and show that the extracted witness $\mathbb{w}$ satisfies the permutation σ:

1. Run the algebraic adversary $(\mathbb{x}, \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk})$.
2. Let $\mathbb{x} = ([z], [h])$. Note that since $\mathcal{A}$ is algebraic, it also outputs vectors of scalars showing how $[z], [h]$ can be computed from $(\mathsf{pk}, \mathsf{vk})$. $\mathcal{E}_\mathcal{A}$ reconstructs $\widetilde{z}(X), \widetilde{h}(X)$ such that $[z] = [\widetilde{z}(X)], [h] = [\widetilde{h}(X)]$.
3. Output $\mathbb{w} = (\mathbb{w}_z, \mathbb{w}_h)$ where $\mathbb{w}_z[i] = \widetilde{z}(\omega^i), \mathbb{w}_h[i] = \widetilde{h}(\omega^i)$, $\forall i \in [m]$.

Similar to the proof of Theorem 2, parse $\pi = \{[f]\}_{f\in\mathcal{F}}$ and reconstruct $\{\widetilde{f}(X)\}_{f\in\mathcal{F}}$ such that $[f] = [\widetilde{f}(X)]$.

From the first and the second check by $\mathcal{V}$, we have

$$\widetilde{v}(\omega^i) = (\widetilde{z}(\omega^i) - \widetilde{h}(\omega^i)) \cdot \frac{\omega^i(r^m - 1)}{m(r - \omega^i)} = (\widetilde{z}(\omega^i) - \widetilde{h}(\omega^i)) \cdot L_i(r)\,,$$

$$\widetilde{v_\sigma}(\omega^i) = \widetilde{z}(\omega^i) \cdot \frac{t(\omega^i)(r^m - 1)}{m(r - t(\omega^i))} = \widetilde{z}(\omega^i) \cdot L_{\sigma^{-1}(i)}(r)\,,$$

Together with the next three checks by $\mathcal{V}$ for univariate sumcheck, we have

$$\sum_{i\in[m]} \widetilde{v}(\omega^i) - \sum_{i\in[m]} \widetilde{v_\sigma}(\omega^i) = \sum_{i\in[m]} (\widetilde{z}(\omega^i) - \widetilde{h}(\omega^i)) \cdot L_i(r) - \sum_{i\in[m]} \widetilde{z}(\omega^i) \cdot L_{\sigma^{-1}(i)}(r) = 0\,. \tag{29}$$

Since r is a random point from the random oracle, the output $\mathbb{w}$ of the extractor should be a valid witness. □

I.5 Proof of Theorem 4

Proof. Completeness and updatability follow naturally from the construction. For knowledge soundness, we can build an extractor by calling the extractor of Dynamo to extract the witness which satisfies the copy constraints. The knowledge soundness of $\mathcal{S}$ and the verification algorithm can also ensure that this extracted witness also satisfies the gate constraints. The complexities of $\mathcal{G}, \mathcal{P}, \mathcal{U}$ and $\mathcal{V}$ follow from the protocol and the discuss before. □

I.6 Proof of Lemma 3

Proof. For the first item, not that if check_k succeeds, then all gates in $\mathsf{buf}^{(k)}$ that do not appear in any $\mathsf{buf}^{(i)}$, for $i < k$ will satisfy the gate constraints. Therefore by the end of the prefix check, all gates will pass the gate check which implies that the prefix check implies global check. For the second item, note that if a global check succeeds a prefix check might fail because for the gates that appear both in $\mathsf{buf}^{(k)}$ and $\mathsf{buf}^{(i)}$ (call this set of gates W), check_k might fail. But if updates in $\mathsf{buf}^{(i)}$ come after updates in $\mathsf{buf}^{(k)}$, then it has to be the case that gates in W are satisfied individually both in $\mathsf{buf}^{(i)}$ and $\mathsf{buf}^{(k)}$ and therefore the prefix check succeeds. □

I.7 Proof of Theorem 5

Proof. Completeness and updatability follow naturally from the construction and Lemma 3. For the extractor for knowledge soundness, we directly use the extractor for Dynamo to extract each $\mathbf{z}_i$. Reconstruct $\mathbf{z}$ from $\sum_i \mathbf{z}_i$. It is natural to see that such $\mathbf{z}$ satisfy the copy constraints from the security proof of Dynamo. Now we argue that this extracted witness also satisfies the gate constraints.

Parse $\pi = (\mathsf{com}(\mathbf{z}_j), \mathsf{com}(\mathbf{h}_j), \pi\mathsf{gate}^{(j)}, \pi\mathsf{perm}^{(j)})_{j\in[0,l]}$. Now we argue that the gate constraints are satisfied, which is implied by Lemma 3. More specifically, consider any gate with index $\mathsf{id} \in [n]$, and suppose it first appears in $S^{(i)}$, i.e., for all $j \in [0, i-1]$, $\mathsf{id}, \mathsf{id}+n, \mathsf{id}+2n \notin S^{(j)}$. Then for any $j \in [0, i-1]$, either we have $\mathsf{buf}^{(j)}$ is empty ($\mathbf{z}_j = 0$), or we have a proof $\pi\mathsf{gate}^{(j)}$ showing

that as an index outside $S^{(j)}$, $\mathsf{com}(\mathbf{z}_j)$ opens to 0 at $\mathsf{id}, \mathsf{id}+n, \mathsf{id}+2n$, i.e., $\mathbf{z}_j[\mathsf{id}] = \mathbf{z}_j[\mathsf{id}+n] = \mathbf{z}_j[\mathsf{id}+2n] = 0$. Therefore we have

$$\mathbf{z}[\mathsf{id}] = \sum_{j\in[i,l]} \mathbf{z}_j[\mathsf{id}]\,, \quad \mathbf{z}[\mathsf{id}+n] = \sum_{j\in[i,l]} \mathbf{z}_j[\mathsf{id}+n]\,, \quad \mathbf{z}[\mathsf{id}+2n] = \sum_{j\in[i,l]} \mathbf{z}_j[\mathsf{id}+2n]\,.$$

Moreover, $\pi\mathsf{gate}^{(i)}$ shows that for any each gate j appearing in $S^{(i)}$, the values from opening $\prod_{k=i}^{l} c_k$ at j satisfy the multiplication relation, i.e.,

$$\left(\sum_{k\in[i,l]} \mathbf{z}_k[j]\right) \cdot \left(\sum_{k\in[i,l]} \mathbf{z}_k[j+n]\right) = \sum_{k\in[i,l]} \mathbf{z}_k[j+2n]\,,$$

which is also true for gate id. Finally, we have

$$\mathbf{z}[\mathsf{id}] \cdot \mathbf{z}[\mathsf{id}+n] = \mathbf{z}[\mathsf{id}+2n]\,,$$

which means the gate constraints are also satisfied for the extracted witness.

Our security proof is based on the security proofs of Dynamo (q-DLOG (see Assumption 1) in the AGM) and the IPA (see Theorem 12). According to the analysis in Section F.2, $\pi\mathsf{gate}^{(i)}$ requires $O(2^i \cdot \log^2 n)$ time to compute and the proof size is $O(\log^2 n)$. Therefore, in our final protocol, the proof size is $O(\log^3 n)$ and the amortized update time is $O(\log^3 n)$ (each $\mathsf{buf}^{(i)}$ will be recomputed after every $O(2^i)$ updates). □

I.8 Proof of Theorem 6

Proof. The completeness and soundness follow directly from those of the dynamic zk-SNARK DS. The complexities also follow from the complexities of DS. □

I.9 Proof of Lemma 5

Proof. The proof is similar to the proof of Theorem 6. We want to mention that although there are $O(\log M)$ wires changed for the SUMTREE, they are basically all from addition gates. The wires changed from multiplication gates are all from the computation F. □

J Adding zero-knowledge

In this section, we talk about how to add zero-knowledge for Dynamo, Dynaverse and Dynalog. Since one key part of their proofs is KZG commitments to interpolated polynomials, the central idea is to add mask polynomials. For example, to commit to $\mathbf{z} \in \mathbb{F}^m$, we compute the KZG commitment to the following polynomial $z(X)$:

$$z(X) = \sum_{i\in[m]} \mathbf{z}[i] \cdot L_i(X)$$

such that $z(\omega^i) = \mathbf{z}[i]$, for all $i \in [m]$. A standard way to mask this polynomial is to pick a random polynomial $\rho(X)$ and compute $z^{\mathsf{zk}}(X) = z(X)+\rho(X)\cdot(X^m-1)$. It is easy to see that we still have $z^{\mathsf{zk}}(\omega^i) = \mathbf{z}[i]$ for all $i \in [m]$. In fact, $[z^{\mathsf{zk}}(X)]$ is a hiding commitment for $\mathbf{z}$. Now in this section, commitments to vectors of form $\mathsf{com}(\mathbf{z})$ will all be regarded as hiding commitments with masks.

J.1 Proof of Lemma 1

Proof. We introduce how to add zero-knowledge for Dynamo here.
Non-universal version. We need to slightly modify the relaxed permutation relation here to accommodate the change of commitments. More specifically, now the indexed relation contains those tuples

$$(\mathbb{x}, \mathbb{w}) = (([z^{\mathsf{zk}}], [h^{\mathsf{zk}}]), (z^{\mathsf{zk}}, h^{\mathsf{zk}}))$$

such that

$$z^{\mathsf{zk}}(\omega^i) = z^{\mathsf{zk}}(\omega^{\sigma(i)}) + h^{\mathsf{zk}}(\omega^i),\ \forall i \in [m]\,.$$

We put the polynomials (possibly with masks) in the witness so that honest provers can make use of the random masks. However, we still only care about the evaluations on roots of unity.

Suppose the prover has the following two polynomials:

$$z^{\mathsf{zk}}(X) = z(X) + \rho_z(X)\cdot(X^m-1)\,,$$

$$h^{\mathsf{zk}}(Y) = h(Y) + \rho_h(Y)\cdot(Y^m-1)\,.$$

Introduce a mask for $v(X,Y)$ such that $v^{\mathsf{zk}}(\omega^i,\omega^j)$ still encodes $u(\omega^i,\omega^j)\cdot z(\omega^i)$:

$$v^{\mathsf{zk}}(X,Y) = v(X,Y) + \rho_v\cdot(Y^m-1), \quad \text{where } \rho_v \xleftarrow{\$} \mathbb{F}\,.$$

We would like to show that

$$u(X,Y)\cdot z^{\mathsf{zk}}(X) = v^{\mathsf{zk}}(X,Y) + q_v^{\mathsf{zk}}(X,Y)(Y^m-1) + \alpha^{\mathsf{zk}}(X,Y)(X^m-1)\,. \quad (30)$$

Pick $q_v^{\mathsf{zk}}(X,Y)$ as

$$q_v^{\mathsf{zk}}(X,Y) = -\rho_v + \rho_{q,v}\cdot(X^m-1), \quad \text{where } \rho_{q,v} \xleftarrow{\$} \mathbb{F}\,.$$

Then we can calculate $\alpha^{\mathsf{zk}}(X,Y)$ as

$$\alpha^{\mathsf{zk}}(X,Y) = \alpha(X,Y) + \rho_z(X)\cdot u(X,Y) - \rho_{q,v}\cdot(Y^m-1)\,.$$

For the relation between $v^{\mathsf{zk}}(X,Y)$ and $h^{\mathsf{zk}}(Y)$, we need the following check:

$$\sum_{i\in[m]} v^{\mathsf{zk}}(\omega^i,\omega^j) = h^{\mathsf{zk}}(\omega^j),\ \forall j \in [m]\,.$$

Pick for random $\rho_{q,h} \xleftarrow{\$} \mathbb{F}$ and define

$$q_h^{\mathsf{zk}}(X,Y) = -\frac{1}{m}\rho_h(Y) + \rho_v + \rho_{q,h} \cdot X\,,$$

following the univariate sumcheck idea in Section 4, we check

$$v^{\mathsf{zk}}(X,Y) = \beta^{\mathsf{zk}}(X,Y) \cdot X + \frac{1}{m}h^{\mathsf{zk}}(Y) + q_h^{\mathsf{zk}}(X,Y) \cdot (Y^m - 1)\,, \tag{31}$$

where

$$\beta^{\mathsf{zk}}(X,Y) = \beta(X,Y) - \rho_{q,h} \cdot (Y^m - 1)\,.$$

For the verification, replace the first check by $\mathcal{V}$ in Dynamo with Eqs. (30) and (31). Also, based on $z^{\mathsf{zk}}, h^{\mathsf{zk}}, \beta^{\mathsf{zk}}$, we can naturally derive corresponding polynomials for their degree checks.

Now we can use

$$\mathcal{F}^{\mathsf{zk}} = \{\alpha^{\mathsf{zk}}, \beta^{\mathsf{zk}}, Z^{\mathsf{zk}}, H^{\mathsf{zk}}, B^{\mathsf{zk}}, q_v^{\mathsf{zk}}, q_h^{\mathsf{zk}}\}$$

instead for Dynamo to achieve zero knowledge. We omit the redundant illustration for minor changes in the keys (basically we need $O(1)$ number of new prover keys to help update the group elements in $\mathcal{F}^{\mathsf{zk}}$) and the detailed protocol of zero-knowledge Dynamo.

We only show here a simulator for Zero-knowledge property:

$\mathcal{S}(1^\lambda, \mathbb{i}) \to (t, \mathsf{pk}, \mathsf{vk})$:
Follow every step of $\mathcal{G}(1^\lambda, [m, \sigma])$, and output $(t = (\tau_X, \tau_Y, u(X,Y)), \mathsf{pk}, \mathsf{vk})$.

$\mathcal{S}(t, \mathsf{pk}, \mathsf{vk}, \mathbb{x}_0, \ldots, \mathbb{x}_l) \to (\tilde{\pi}_0, \ldots, \tilde{\pi}_l)$:
For every $i \in [0, l]$,

(a) Parse $\mathbb{x}_i$ as $([z^{\mathsf{zk}}], [h^{\mathsf{zk}}])$.

(b) Pick $\tau_v, \tau_{q,v}, \tau_{q,h} \xleftarrow{\$} \mathbb{F}$ and let $\widetilde{[v^{\mathsf{zk}}]} = [\tau_v]$, $\widetilde{[q_v^{\mathsf{zk}}]} = [\tau_{q,v}]$ and $\widetilde{[q_h^{\mathsf{zk}}]} = [\tau_{q,h}]$.

(c) Compute $\widetilde{[Z^{\mathsf{zk}}]} = [z^{\mathsf{zk}}]^{\tau_Y^m}$, $\widetilde{[H^{\mathsf{zk}}]} = [h^{\mathsf{zk}}]^{\tau_X^m}$.

(d) Compute

$$\widetilde{[\alpha^{\mathsf{zk}}]} = \left(\frac{[z^{\mathsf{zk}}]^{u(\tau_X,\tau_Y)}}{\widetilde{[v^{\mathsf{zk}}]} \cdot \widetilde{[q_v^{\mathsf{zk}}]}^{\tau_Y^m - 1}} \right)^{\frac{1}{\tau_X^m - 1}}.$$

(e) Compute $\widetilde{[\beta^{\mathsf{zk}}]}, \widetilde{[B^{\mathsf{zk}}]}$ as following:

$$\widetilde{[\beta^{\mathsf{zk}}]} = \left(\frac{\widetilde{[v^{\mathsf{zk}}]}}{\widetilde{[q_h^{\mathsf{zk}}]}^{\tau_Y^m - 1} \cdot [h^{\mathsf{zk}}]^{1/m}} \right)^{\frac{1}{\tau_X}} \quad \widetilde{[B^{\mathsf{zk}}]} = \widetilde{[\beta^{\mathsf{zk}}]}^{\tau_X^2}$$

(f) Output all the group elements computed above in π_i.

Now we argue $\mathcal{S}$ correctly simulates a prover. Set $\Omega = \{0\} \cup \{\omega^i\}_{i\in[m]}$.

For fixed polynomial $v(\tau_X, Y)$ and a value τ_Y, if $\tau_Y \notin \Omega$ and $\rho_v \xleftarrow{\$} \mathbb{F}$, then according to Lemma 7, $v^{\mathsf{zk}}(\tau_X, \tau_Y) = v(\tau_X, \tau_Y) + \rho_v \cdot (\tau_Y^m - 1)$ is also uniform in $\mathbb{F}$.

For fixed polynomial $q_v(X, \tau_Y)$ and a value τ_X, if $\tau_X \notin \Omega$ and $\rho_{q,v} \xleftarrow{\$} \mathbb{F}$, then according to Lemma 7, $q_v^{\mathsf{zk}}(\tau_X, \tau_Y) = -\rho_v + \rho_{q,v} \cdot (\tau_X^m - 1)$ is also uniform in $\mathbb{F}$.

For fixed polynomial $q_h(X, \tau_Y)$ and a value τ_X, if $\tau_X \notin \Omega$ and $\rho_{q,h} \xleftarrow{\$} \mathbb{F}$, then according to Lemma 7, $q_h^{\mathsf{zk}}(\tau_X, \tau_Y) = -\frac{1}{m}\rho_h(\tau_Y) + \rho_v + \rho_{q,h} \cdot \tau_X$ is also uniform in $\mathbb{F}$.

Above all, as long as $\tau_X, \tau_Y \notin \Omega$ (which is of overwhelming probability),

$$(v^{\mathsf{zk}}(\tau_X, \tau_Y), q_v^{\mathsf{zk}}(\tau_X, \tau_Y), q_h^{\mathsf{zk}}(\tau_X, \tau_Y))$$

is uniform in $\mathbb{F}^3$ and thus $\mathcal{S}$ can perfectly simulate $([v^{\mathsf{zk}}], [q_v^{\mathsf{zk}}], [q_h^{\mathsf{zk}}])$. Based on the codes of the prover and the simulator, $[f^{\mathsf{zk}}]_{f\in\{\alpha,\beta,Z,B,H\}}$ are exactly determined by $[f^{\mathsf{zk}}]_{f\in\{z,h,v,q_v,q_h\}}$. Therefore, $\mathcal{S}$ can successfully simulate a prover.

Universal version. We follow the same trick from the non-universal version. Suppose the prover has the following two polynomials:

$$z^{\mathsf{zk}}(X) = z(X) + \rho_z(X) \cdot (X^m - 1)\,,$$

$$h^{\mathsf{zk}}(X) = h(X) + \rho_h(X) \cdot (X^m - 1)\,.$$

Introduce masks for $v(X), v_\sigma(X)$ such that $v^{\mathsf{zk}}(X), v_\sigma^{\mathsf{zk}}(X)$ still have the same evaluations on roots of unity respectively:

$$v^{\mathsf{zk}}(X) = v(X) + \rho_v \cdot (X^m - 1)\,,$$

$$v_\sigma^{\mathsf{zk}}(X) = v_\sigma(X) + \rho_\sigma \cdot (X^m - 1)\,,$$

where $\rho_v, \rho_\sigma \xleftarrow{\$} \mathbb{F}$.

Now we can derive calculate $\beta^{\mathsf{zk}}(X), \beta_\sigma^{\mathsf{zk}}(X)$ as:

$$\beta^{\mathsf{zk}}(X) = \beta(X) + m\rho_v \cdot (r - X) - (r^m - 1)X \cdot (\rho_z(X) - \rho_h(X))\,,$$

$$\beta_\sigma^{\mathsf{zk}}(X) = \beta_\sigma(X) + m\rho_\sigma \cdot (r - t(X)) - (r^m - 1)t(X)\rho_z(X)\,.$$

The next step is to check $\sum_{i\in[m]}(v^{\mathsf{zk}} - v_\sigma^{\mathsf{zk}})(\omega^i) = 0$. However, we cannot directly apply the univariate sumcheck idea directly, since $v^{\mathsf{zk}} - v_\sigma^{\mathsf{zk}}$ has degree m now. Showing the consistency between $v^{\mathsf{zk}} - v_\sigma^{\mathsf{zk}}$ and $v - v_\sigma$ and then doing the univariate sumcheck on $v - v_\sigma$ do not work either because without masks the simulator cannot know and simulate $v - v_\sigma$. To address this, we would consider another polynomial $u^{\mathsf{zk}}(X)$ with degree at most $m - 1$, which is very close to $v - v_\sigma$. In particular, let $\varphi = \omega^{m/4}$ denote the 4-th root of unity and $\{L_i'(X)\}_{i\in[1,4]}$ the Lagrange polynomials for vectors of size 4. For $\rho_1, \rho_2, \rho_3 \in \mathbb{F}$, define

$$u^{\mathsf{zk}}(X) = v(X) - v_\sigma(X) + \rho_1 \cdot L_{m/4}(X) + \rho_2 \cdot L_{m/2}(X) + \rho_3 \cdot L_{3m/4}(X)\,.$$

We can see that $u^{\mathsf{zk}}(X)$ agrees with $v(X) - v_\sigma(X)$ on all $X = \omega^i$ except for $i \in \{m/4, m/2, 3m/4\}$. If an honest prover does a univariate sumcheck on $u^{\mathsf{zk}}(X)$, then he can convince the verifier that $\sum_{i\in[m]} u^{\mathsf{zk}}(\omega^i) = \rho_{123}$ where $\rho_{123} = \rho_1 + \rho_2 + \rho_3$:

$$u^{\mathsf{zk}}(X) = q_u^{\mathsf{zk}}(X) \cdot X + \rho_{123}, \text{ plus using } U^{\mathsf{zk}}(X) \text{ to check } u^{\mathsf{zk}} \text{ has degree } < m\,. \tag{32}$$

Now based on $u^{\mathsf{zk}}(X)$, we can check the following to ensure the relation between u^{zk} and $v^{\mathsf{zk}} - v_\sigma^{\mathsf{zk}}$:

1. $u^{\mathsf{zk}}(X)$ and $v^{\mathsf{zk}}(X) - v_\sigma^{\mathsf{zk}}(X)$ agree on $\{\omega^i\}_{i\in[m]\setminus\{m/4,m/2,3m/4,m\}}$:
$$u^{\mathsf{zk}}(X) - (v^{\mathsf{zk}}(X) - v_\sigma^{\mathsf{zk}}(X)) = q_v^{\mathsf{zk}}(X) \cdot \frac{X^m - 1}{X^4 - 1}\,. \tag{33}$$
2. Let $w(X)$ be a polynomial that encodes the difference between $u^{\mathsf{zk}}(X)$ and $v^{\mathsf{zk}}(X) - v_\sigma^{\mathsf{zk}}(X)$ on $\{\omega^{m/4}, \omega^{m/2}, \omega^{3m/4}, \omega^m\} = \{\varphi, \varphi^2, \varphi^3, \varphi^4\}$, i.e., $w(X) = \rho_1 \cdot L_1'(X) + \rho_2 \cdot L_2'(X) + \rho_3 \cdot L_3'(X)$:
$$u^{\mathsf{zk}}(X) - (v^{\mathsf{zk}}(X) - v_\sigma^{\mathsf{zk}}(X)) = \gamma^{\mathsf{zk}}(X) \cdot (X^4 - 1) + w(X)\,.$$
Then apply univariate sumcheck on $w(X)$ to show that $\sum_{i\in[1,4]} w(\varphi^i) = \rho_{123}$:
$$w(X) = q_w^{\mathsf{zk}}(X) \cdot X + \rho_{123}, \quad \text{plus checking } w \text{ has degree less than } 4\,.$$
These two equations can be combined into the following one removing w (plus $W^{\mathsf{zk}}(X)$ for $\mathsf{deg} q_w^{\mathsf{zk}} < 3$):
$$u^{\mathsf{zk}}(X) - (v^{\mathsf{zk}}(X) - v_\sigma^{\mathsf{zk}}(X)) = \gamma^{\mathsf{zk}}(X) \cdot (X^4 - 1) + q_w^{\mathsf{zk}}(X) \cdot X + \rho_{123}\,. \tag{34}$$

Above all, to check $\sum_{i\in[m]}(v^{\mathsf{zk}} - v_\sigma^{\mathsf{zk}})(\omega^i) = 0$, the prover picks $\rho_1, \rho_2, \rho_3 \xleftarrow{\$} \mathbb{F}$ and computes ρ_{123}, u^{zk}, q_u^{zk}, q_v^{zk}, γ^{zk}, q_w^{zk}, U^{zk} and W^{zk} to show Eqs. (32) to (34). We omit the redundant illustration for efficient computing commitments to these polynomials and show here the simulator:

$\mathcal{S}(1^\lambda, \mathbb{i}) \to (t, \mathsf{pk}, \mathsf{vk})$:
Follow every step of $\mathcal{G}(1^\lambda, [m, \sigma])$, and output $((\tau_X, t(X)), \mathsf{pk}, \mathsf{vk})$.

$\mathcal{S}((\tau_X, t(X)), \mathsf{pk}, \mathsf{vk}, \mathbb{x}_0, \ldots, \mathbb{x}_l) \to (\tilde{\pi}_0, \ldots, \tilde{\pi}_l)$:
For every $i \in [0, l]$,
(a) Parse $\mathbb{x}_i$ as $([z^{\mathsf{zk}}], [h^{\mathsf{zk}}])$.
(b) Compute $r = \mathsf{H}([z^{\mathsf{zk}}]||[h^{\mathsf{zk}}])$.
(c) Pick $\tau_v, \tau_{v,\sigma}, \tau_u, \tau_{q,w}, \widetilde{\rho_{123}} \xleftarrow{\$} \mathbb{F}$ and let $\widetilde{[v^{\mathsf{zk}}]} = [\tau_v]$, $\widetilde{[v_\sigma^{\mathsf{zk}}]} = [\tau_{v,\sigma}]$, $\widetilde{[u^{\mathsf{zk}}]} = [\tau_u]$ and $\widetilde{[q_w^{\mathsf{zk}}]} = [\tau_{q,w}]$.
(d) Compute $\widetilde{[Z^{\mathsf{zk}}]} = [z^{\mathsf{zk}}]^{\tau_Y^m}$, $\widetilde{[H^{\mathsf{zk}}]} = [h^{\mathsf{zk}}]^{\tau_X^m}$.
(e) Compute
$$\widetilde{[\beta^{\mathsf{zk}}]} = \left(\frac{\widetilde{[v^{\mathsf{zk}}]}^{m(r-\tau_X)}}{([z^{\mathsf{zk}}]/[h^{\mathsf{zk}}])^{\tau_X(r^m-1)}}\right)^{\frac{1}{\tau_X^m-1}}, \widetilde{[\beta_\sigma^{\mathsf{zk}}]} = \left(\frac{\widetilde{[v_\sigma^{\mathsf{zk}}]}^{m(r-t(\tau_X))}}{[z^{\mathsf{zk}}]^{t(\tau_X)(r^m-1)}}\right)^{\frac{1}{\tau_X^m-1}}.$$

(f) Compute

$$\widetilde{[q_u^{\mathsf{zk}}]} = \left(\frac{\widetilde{[u^{\mathsf{zk}}]}}{\widetilde{\rho_{123}}}\right)^{\frac{1}{\tau_X}} \qquad \widetilde{[U^{\mathsf{zk}}]} = \widetilde{[u^{\mathsf{zk}}]}^{\tau_X}$$

$$\widetilde{[q_v^{\mathsf{zk}}]} = \left(\frac{\widetilde{[u^{\mathsf{zk}}]} \cdot \widetilde{[v_\sigma^{\mathsf{zk}}]}}{\widetilde{[v^{\mathsf{zk}}]}}\right)^{\frac{\tau_X^4-1}{\tau_X^m-1}}$$

$$\widetilde{[\gamma^{\mathsf{zk}}]} = \left(\frac{\widetilde{[u^{\mathsf{zk}}]} \cdot \widetilde{[v_\sigma^{\mathsf{zk}}]}}{\widetilde{[v^{\mathsf{zk}}]} \cdot \widetilde{[q_w^{\mathsf{zk}}]} \cdot \widetilde{\rho_{123}}}\right)^{\frac{1}{\tau_X^4-1}} \qquad \widetilde{[W^{\mathsf{zk}}]} = \widetilde{[q_w^{\mathsf{zk}}]}^{\tau_X^{m-2}}$$

(g) Output all the group elements computed above in π_i.

Now we argue $\mathcal{S}$ correctly simulates a prover. Following the idea from the proof of Lemma 1, what we left here is to argue that for randomly picked $\rho_1, \rho_2, \rho_3 \xleftarrow{\$} \mathbb{F}$, with overwhelming probability,

$$(u^{\mathsf{zk}}(\tau_X), q_w^{\mathsf{zk}}(\tau_X), \rho_{123})$$

is uniform in $\mathbb{F}^3$, which can easily concluded by their definition. □

J.2 Proof of Lemma 10

Proof. Suppose the prover has the following two polynomials:

$$a^{\mathsf{zk}}(X) = a(X) + \rho_a(X) \cdot (X^m - 1)\,,$$
$$b^{\mathsf{zk}}(X) = b(X) + \rho_b(X) \cdot (X^\ell - 1)\,.$$

We observe that

$$\begin{aligned} b^{\mathsf{zk}}(X) &= b(X) + \rho_b(X) \cdot (X^\ell - 1) \\ &= b(X) + \rho_b(X) \cdot ((\varphi^{1-k} Y^{\frac{m}{\ell}})^\ell - 1) + \rho_b(X) \cdot (X^\ell - (\varphi^{1-k} Y^{\frac{m}{\ell}})^\ell) \\ &= b(X) + \rho_b(X) \cdot (Y^m - 1) + \rho_b(X) \cdot g(X, \varphi^{1-k} Y^{\frac{m}{\ell}}, \ell) \cdot (X - \varphi^{1-k} Y^{\frac{m}{\ell}})\,, \end{aligned}$$

where we define polynomial

$$g(X, Y, i) := \frac{X^i - Y^i}{X - Y}\,.$$

First, we can calculate the batch proof $q_0^{\mathsf{zk}}(X)$ directly from $a^{\mathsf{zk}}(X)$:

$$q_0^{\mathsf{zk}}(X) = q_0(X) + \rho_a(X) \cdot \prod_{i \in [k, k+\ell-1]} (X - \omega^i)\,.$$

Next, we want to build $d^{\mathsf{zk}}(Y)$ for the following two checks:

$$a^{\mathsf{zk}}(X) - d^{\mathsf{zk}}(Y) = q_{ac}^{\mathsf{zk}}(X, Y) \prod_{j \in [k, k+\ell-1]} (X - \omega^j) + q_{cd}^{\mathsf{zk}}(X, Y)(X - Y)\,,$$

$$b^{\mathsf{zk}}(X) - d^{\mathsf{zk}}(Y) = q_{bd}^{\mathsf{zk}}(X,Y)(X - \varphi^{1-k}Y^{\frac{m}{\ell}}) + q_d^{\mathsf{zk}}(X,Y)(Y^m - 1)\,.$$

Pick $d^{\mathsf{zk}}(Y), q_{cd}^{\mathsf{zk}}(X,Y), q_d^{\mathsf{zk}}(X,Y)$ as

$$d^{\mathsf{zk}}(Y) = d(Y) + \rho_d \cdot (Y^m - 1) = d(Y) + \rho_d \cdot (X^m - 1) - \rho_d \cdot g(X,Y,m)(X - Y)\,,$$

$$q_{cd}^{\mathsf{zk}}(X,Y) = q_{cd}(X,Y) + \rho_d \cdot g(X,Y,m) + \rho_{cd}(X^m - 1)\,,$$

$$q_{bd}^{\mathsf{zk}}(X,Y) = q_{bd}(X,Y) + \rho_b(X) \cdot g(X, \varphi^{1-k}Y^{\frac{m}{\ell}}, \ell) + \rho_{bd}(Y^m - 1)\,,$$

where $\rho_d, \rho_{cd}, \rho_{bd} \xleftarrow{\$} \mathbb{F}$. It is easy to see that

$$(d^{\mathsf{zk}}(\tau_Y), q_{cd}^{\mathsf{zk}}(\tau_X, \tau_Y), q_{bd}^{\mathsf{zk}}(\tau_X, \tau_Y))$$

is uniform in $\mathbb{F}^3$ with overwhelming probability and $q_{ac}^{\mathsf{zk}}(X,Y), q_d^{\mathsf{zk}}(X,Y)$ can be computed from $d^{\mathsf{zk}}(Y), q_{cd}^{\mathsf{zk}}(X,Y), q_d^{\mathsf{zk}}(X,Y)$ and $a^{\mathsf{zk}}(X), b^{\mathsf{zk}}(X)$. We omit the details for the simulator here. □

J.3 Proof of Lemma 14

Proof. We briefly introduce how to add zero knowledge to Dynaverse and Dynavold here. The intuition is also to add mask polynomials. Take Dynavold as an example. In particular, we pick $z_{t,i}^{\mathsf{zk}}(X), h_{t,i}^{\mathsf{zk}}(Y), s_{t,i}^{\mathsf{zk}}(X)$ as

$$z_{t,i}^{\mathsf{zk}}(X) = z_{t,i}(X) + \rho_{t,i}^{z} \cdot (X^m - 1)\,,$$

$$h_{t,i}^{\mathsf{zk}}(Y) = h_{t,i}(Y) + \rho_{t,i}^{h} \cdot (Y^m - 1)\,,$$

$$s_{t,i}^{\mathsf{zk}}(X) = s_{t,i}(X) + \rho_{t,i}^{s}(X) \cdot (X^\ell - 1)\,,$$

where $\rho_{t,i}^{z}, \rho_{t,i}^{z}, \rho_{t,i}^{s} \xleftarrow{\$} \mathbb{F}$. Then we can call the zero-knowledge version of Dynamo and consistency check for $([z_{t,i}^{\mathsf{zk}}], [h_{t,i}^{\mathsf{zk}}])$ and $([z_{t,i}^{\mathsf{zk}}], [s_{t,i}^{\mathsf{zk}}])$ respectively, and show for the gate constraints and input consistency based on $s_{t,i}^{\mathsf{zk}}(X)$ and $h_{t,i}^{\mathsf{zk}}(Y)$. □

J.4 Ensuring the IVC transferred state is zero-knowledge

Recall that in the update algorithm of Dynavold (cf. Fig. 13), the only place we need the whole witnesses for one bucket is where we compute quotient polynomials $A_i'(X)$ from Eq. (25). This is because we need to compute the polynomial multiplication and division from scratch every time. Now we consider another approach to update $[A_i(X)]$ without revealing all the coefficients or evaluations of $A_i(X)$. Whenever we compute or update the proof, we also maintain the following in the state (we only care about and compute the quotients and ignore the remainders here, same thing for other quotients below):

$$\left[\frac{s_{4,i}(X) \cdot L_j(X)}{X^\ell - 1}\right], \left[\frac{s_{5,i}(X) \cdot L_j(X)}{X^\ell - 1}\right] \text{ for } i,j = 1,\ldots,\ell,$$

and we need to pre-compute $[l_{i,j}(X)]$ for $i, j \in [\ell]$ where

$$l_{i,j}(X) = \frac{L_i(X) \cdot L_j(X)}{X^\ell - 1}\,.$$

When we have an update for $s_{4,i}(X)$ or $s_{5,i}(X)$, we can update both $[A_i(X)]$ and the state in $O(m)$ time. For example, if $s_{4,i}(X)$ is updated as

$$s'_{4,i}(X) = s_{4,i}(X) + L_j(X) \cdot \delta\,,$$

then we have

$$A'_i(X) = A_i(X) + \frac{s_{5,i}(X) \cdot L_j(X)\delta}{X^\ell - 1}\,,$$

and we can update $[A_i(X)]$ with the help of the previous state. Also, we can update the whole state with the help of $[l_{i,j}(X)]$. All the group elements in the state can be zero-knowledge through simple masks.

We emphasize that we only need to maintain one state of $O(\ell) = O(\sqrt{m})$ size since updates in IVC are monotonically increasing.

K Figure illustrations

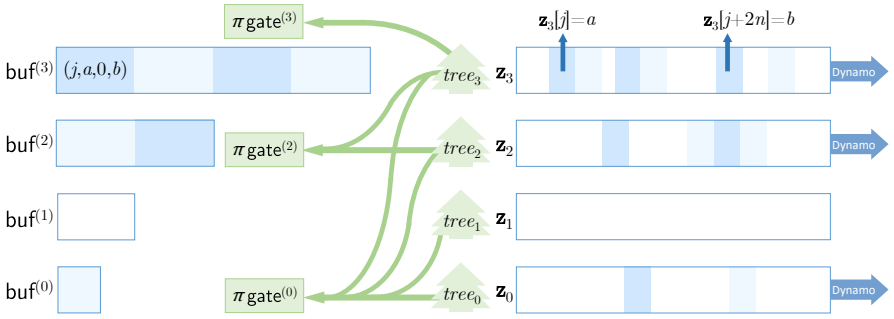

Fig. 14. Core data structure illustration for Dynalog. In this example, there are four buffers and $\mathsf{buf}^{(1)}$ is empty. Each buffer $\mathsf{buf}^{(k)}$ maps to a vector $\mathbf{z}_k$ and we build an AMT $tree_k$ on top of that.

- $\mathcal{G}(1^\lambda, [n, n_0, \sigma]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - Set $\ell = \log n$ and $m = 3n + n_0$.
  - Set $\mathsf{AMT.Setup}(1^\lambda, m) \to (pk, vk)$.
  - Set $(\mathsf{pk}g^{(i)}, \mathsf{vk}g^{(i)})$ for all $i = 0, \ldots, \ell$ as the necessary keys from IPAs used for $\pi\mathsf{gate}^{(i)}$.
  - Set $\mathcal{D}.\mathcal{G}(1^\lambda, [m, \sigma]) \to (\mathsf{pk}_\mathcal{D}, \mathsf{vk}_\mathcal{D})$.
  - Set $\mathsf{pk} = \mathsf{upk} = \{pk, \{\mathsf{pk}g^{(i)}\}_{i\in[\ell]}, \mathsf{pk}_\mathcal{D}\}$ and $\mathsf{vk} = \{vk, \{\mathsf{vk}g^{(i)}\}_{i\in[\ell]}, \mathsf{vk}_\mathcal{D}\}$.

- $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{aux})$:
  - Parse $\mathbb{x}||\mathbb{w}$ as $z_\ell \in \mathbb{F}^m$.
  - Set $\mathbf{z}_i = \mathbf{0} \in \mathbb{F}^m$ for all $i = 0, \ldots, \ell - 1$.
  - For each level $i = 0, \ldots, \ell$ store the following information as aux.
    1. Buffer $\mathsf{buf}^{(i)}$ storing the indices of non-zero gates.
    2. Commitment $\mathsf{com}(\mathbf{z}_i)$ and $\mathbf{z}_i$.
    3. AMT tree $tree_i$ computed using $\mathsf{AMT.ComputeAllProofs}(pk, \mathbf{z}_i)$.
  - For each level $i = 0, \ldots, \ell$ do the following.
    1. Output permutation proof as $\mathcal{D}.\mathcal{P}(\mathsf{pk}_\mathcal{D}, (\mathsf{com}(\mathbf{z}_i), \mathsf{com}(\mathbf{h}_i)), (\mathbf{z}_i, \mathbf{h}_i)) \to \pi\mathsf{perm}^{(i)}$.
    2. Output gate proof as $\mathtt{ComputeGateProof}(i) \to \pi\mathsf{gate}^{(i)}$.
  - Output proof π to contain $\mathsf{com}(\mathbf{z}_i)$, $\mathsf{com}(\mathbf{h}_i)$, $\pi\mathsf{gate}^{(i)}$ and $\pi\mathsf{perm}^{(i)}$ for all $i = 0, \ldots, \ell$.
  - Add $\pi\mathsf{gate}^{(i)}$ and $\pi\mathsf{perm}^{(i)}$ into aux.

- $\mathcal{U}(\mathsf{upk}, \mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}, \pi, \mathsf{aux}) \to (\pi', \mathsf{aux}')$:
  - Let $u = (j, a, b, c)$ be the gate update that produces $\mathbb{w}'$.
  - Let $\mathsf{buf}^{(k)}$ be the first empty buffer from level 0 onwards.
  - Merge $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(k-1)}$ into $\mathsf{buf}^{(k)}$. Also add u into $\mathsf{buf}^{(k)}$ and empty $\mathsf{buf}^{(i)}$ for $i < k$.
  - Set $\mathbf{z}_k[j] = a$, $\mathbf{z}_k[n + j] = b$ and $\mathbf{z}_k[2n + j] = c$. Set $\mathbf{z}_k = \mathbf{z}_k + \sum_{i=0}^{k-1} \mathbf{z}_i$.
  - Compute new commitments $\mathsf{com}(\mathbf{z}_i)$ and new AMT trees $tree_i$ for $i = 0, \ldots, k$.
  - For each level $i = 0, \ldots, k$ do the following.
    1. Output permutation proof as $\mathcal{D}.\mathcal{P}(\mathsf{pk}_\mathcal{D}, (\mathsf{com}(\mathbf{z}_i), \mathsf{com}(\mathbf{h}_i)), (\mathbf{z}_i, \mathbf{h}_i)) \to \pi\mathsf{perm}^{(i)}$.
    2. Output gate proof as $\mathtt{ComputeGateProof}(i) \to \pi\mathsf{gate}^{(i)}$.
  - Output proof π' to contain $\mathsf{com}(\mathbf{z}_i)$, $\mathsf{com}(\mathbf{h}_i)$, $\pi\mathsf{gate}^{(i)}$ and $\pi\mathsf{perm}^{(i)}$ for $i = 0, \ldots, k$. For $i > k$, proof π' has the same elements as π.
  - Update aux to aux' accordingly.

- $\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$:
  - Parse π as $\mathsf{com}(\mathbf{z}_i)$, $\mathsf{com}(\mathbf{h}_i)$, $\pi\mathsf{gate}^{(i)}$ and $\pi\mathsf{perm}^{(i)}$ for all $i = 0, \ldots, \ell$.
  - For each level $i = 0, \ldots, \ell$ do the following.
    1. Check $\mathcal{D}.\mathcal{V}(\mathsf{vk}_\mathcal{D}, (\mathsf{com}(\mathbf{z}_i), \mathsf{com}(\mathbf{h}_i), \pi\mathsf{perm}^{(i)}) \to 1$.
    2. Check the gate proof $\pi\mathsf{gate}^{(i)}$.
  - Parse $\mathbb{x}$ as $\mathbf{z}[3n + 1 \ldots 3n + n_0]$. Set $h_\mathbb{x}(Y) = \sum_{i=3n+1}^{3n+n_0} \mathbf{z}[i](Y^i - Y^{\sigma^{-1}(i)})$. Check whether $[h_\mathbb{x}(Y)] \cdot \prod_{j\in[0,l]} \mathsf{com}(\mathbf{h}_j) = 1_\mathbb{G}$.

Fig. 15. Dynalog using Dynamo $\mathcal{D}$, IPA ([5], see Section F), and AMT commitment [37].

- Public Inputs: $i = [b_1, \ldots, b_N], z_i, z_0$
- Witness: $w_0, z_1, w_1, \ldots, z_{N-1}, w_{N-1}, z_N, w\mathsf{F}$
- Computation:
  - Check $z_1 = \mathsf{F}(b_1 \cdot z_0, b_1 \cdot w_0)$;
  - Check $z_2 = \mathsf{F}(b_2 \cdot z_1, b_2 \cdot w_1)$;
  - $\ldots$
  - Check $z_N = \mathsf{F}(b_N \cdot z_{N-1}, b_N \cdot w_{N-1})$;
  - Check $z_i = \mathsf{SUMTREE}[(b_1 \oplus b_2) \cdot z_1, \ldots, (b_{N-1} \oplus b_N) \cdot z_{N-1}, b_N \cdot z_N]$;
  - Return true.

Fig. 16. Circuit F_N iterating F N times and then selecting the right output. Counter i is given in unary. The circuit might also take additional witnesses for F denoted $w\mathsf{F}$. $\mathsf{SUMTREE}(x_1, \ldots, x_N)$ adds $x_1, \ldots, x_N$ using a binary tree, i.e., in $\log N$ parallel time.

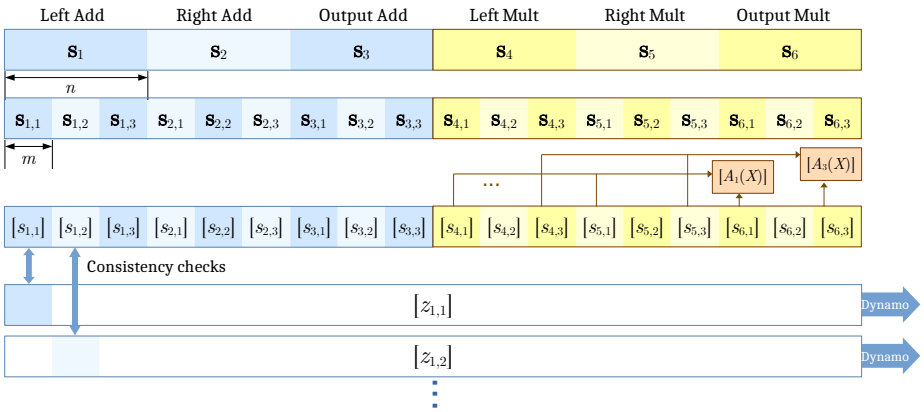

Fig. 17. The Dynavold dynamic SNARK. The initial $6n$-sized witness $[\mathbf{s}_1 \ldots \mathbf{s}_6]$ is split into subvectors $\mathbf{s}_{i,j}$ of size $m = \sqrt{n}$, which are KZG-committed to $[s_{i,j}]$. For every $[s_{i,j}]$ participating in multiplications ($i = 4, 5, 6$) we provide commitments to the quotient polynomials $A_i(X)$. For each subvector, we also compute $[z_{i,j}]$ and provide a Dynamo proof with respect to the permutation σ, and a consistency check between $[s_{i,j}]$ and $[z_{i,j}]$.

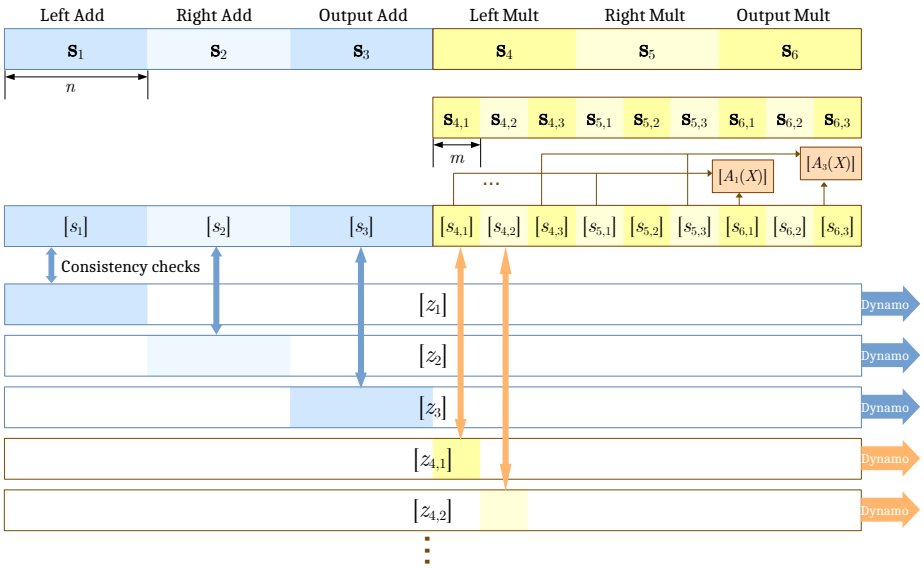

Fig. 18. The optimized Dynavold dynamic SNARK. The addition wires are committed with three commitments $[s_1]$, $[s_2]$ and $[s_3]$ of n-sized vectors. The multiplication wires are committed with 3ℓ commitments of ℓ-sized vectors, as before.