Dynamic zk-SNARKs

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 the notion of *dynamic* zk-SNARKs. A dynamic zk-SNARK is a zk-SNARK that has an additional *update* algorithm. The update algorithm 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 $(\mathbb{x}', \mathbb{w}')$ in *sublinear* time (for $(\mathbb{x}, \mathbb{w})$ and $(\mathbb{x}', \mathbb{w}')$ with small Hamming distance) potentially with the help of a data structure. To the best of our knowledge, none of the commonly-used zk-SNARKs are dynamic—a single update in $(\mathbb{x}, \mathbb{w})$ can be handled only by recomputing the proof, which requires at least linear time. After formally defining dynamic zk-SNARKs, we present three constructions.

The first one, Dynarec, is based on recursive zk-SNARKs, has $O(\log n)$ update time and is folklore, in the sense that it shares similarities (and limitations such as small number of compositions and heuristic security) with existing tree-based Incremental Verifiable Computation (IVC) schemes.

Our second contribution is Dynaverse, a dynamic zk-SNARK based on a new dynamic permutation argument that we propose—Dynamo. Dynaverse has $O(\sqrt{n} \log n)$ update time and proofs of $O(1)$ size. The security of Dynaverse rests upon the q-DLOG assumption in the Algebraic Group Model (AGM)—the same assumption used to prove security of the celebrated Groth16 zk-SNARK.

Our final construction, Dynalog, further advances Dynaverse via a data structure of exponentially-increasing levels and maintainable vector commitments to finally achieve polylogarithmic update time and proofs of $O(\log n)$ size, again under q-DLOG in the AGM (With a single level of SNARK composition, therefore not affecting security, Dynalog's proofs can be further compressed down to $O(1)$.)

As a central application of dynamic zk-SNARKs, we build a compiler from any dynamic zk-SNARK to *bounded-step* and *recursion-free* IVC, a model introduced by Tyagi et al. in 2022. Interestingly, when this transformation is instantiated with a slight modification of Dynaverse, we derive the first bounded-step and recursion-free IVC with sublinear space, removing the need for maintaining a linear-time server for retrieving proofs. We also detail additional applications of dynamic zk-SNARKs such as dynamic state proofs and keyless authentication.

Our preliminary evaluation shows that Dynaverse outperforms baseline PLONK proof recomputation by up to approximately 500× as well as heuristically-secure and asymptotically-superior Dynarec by up to one order of magnitude.

1 Introduction

Data structures are fundamental tools in computer science, enabling us to efficiently update the result of a computation whenever data inputs change. In this paper we put forth the problem of "data structures for zk-SNARKs" and accordingly introduce the notion of *dynamic zk-SNARKs*—SNARKs with efficiently-updatable proofs.

Consider for example the following "commit-and-prove" map-reduce application appearing in zk-coprocessors (e.g., [2]), 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 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 requiring proof recomputation—just as the Merkle commitment can be efficiently updated

Table 1. Asymptotic comparison of Groth16 [21] (static zk-SNARK), Dynarec, Dynaverse, Dynalog and Dynalog+. Dynalog+ further compresses the proof using a **single** level of Groth16 composition, therefore maintaining the same security assumptions. k is the number of updates between source and target statements and with n the number of multiplication and addition gates.

scheme	keygen $\mathcal{G}$	prove $\mathcal{P}$	update $\mathcal{U}$	verify $\mathcal{V}$	proof $\lvert\pi\rvert$	prover key $\lvert\mathsf{pk}\rvert$	verifier key $\lvert\mathsf{vk}\rvert$	security
Groth16 [21]	n	$n \log n$	$n \log n$	1	1	n	1	q-DLOG (AGM)
Dynarec	n	$n \log n$	$k \log n$	1	1	n	1	heuristic
Dynaverse	n	$n \log n$	$k\sqrt{n} \log n$	1	1	n	1	q-DLOG (AGM)
Dynalog	$n \log n$	$n \log^2 n$	$k \log^3 n$	$\log n$	$\log n$	$n \log n$	$\log n$	q-DLOG (AGM)
Dynalog+	$n \log n$	$n \log^2 n$	$k \log^3 n$	1	1	$n \log n$	1	q-DLOG (AGM)

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—in this work we aim to have *succinct* dynamic proofs (We detail more applications of dynamic zk-SNARKs in Sections 1.6 and 7.)

We begin this line of work by first formally defining dynamic zk-SNARKs—see Definition 2. Naturally, a dynamic zk-SNARK for a relation $\mathcal{R}$ is a SNARK $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ 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 $\mathcal{P}$ 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*. To the best of our knowledge, none of the commonly-used zk-SNARKs, such as Groth16 [21], PLONK [20], Bulletproofs [10] and Orion [40], 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 crucially depending on all circuit wires, or the use of polynomial division, which is sensitive to the changes on the dividend polynomial encoding wire values.

1.1 Dynarec: A dynamic zk-SNARK from recursion

While dynamic zk-SNARKs have not been formally defined before in their generality, there have been some constructions of dynamic proof systems for specific types of computation (but not for general-purpose computation), crucially relying on *recursive* zk-SNARKs [6]. For example, Incremental Verifiable Computation (IVC) [38] uses recursive zk-SNARKs to support *dynamic chain computations*, and Reckle trees [31] use recursive zk-SNARKs to support *dynamic batch proofs* in vector commitments.

Our first "warm-up" contribution is Dynarec, a folklore generalization of the aforementioned approaches to *general-purpose computation*—see Section 3.1. Dynarec is a dynamic zk-SNARK based on recursion. While Dynarec has excellent asymptotic complexities ($O(\log n)$ update time), it suffers from standard efficiency and security issues shared by all approaches performing recursive composition of SNARKs: In particular, recursive zk-SNARKs are not particularly practical, and the ones the seem to be (e.g., Plonky2 [36] or folding schemes that can implement recursive relations like Nova [24]) have verifiers that must call a random oracle which must then be encoded in the proven

relation, leading to only *heuristic* security proofs (unless unconventional random oracle models are to be considered [17]). In addition, in order to recurse $\log n$ times (which is needed for Dynarec), an assumption on the size of the underlying extractor is required (to avoid exponential blowup of the final extractor), in particular that the extractor size is at most a constant times the prover size [7] (In general, recursing beyond $\log n$ requires much stronger assumptions.) These security and efficiency limitations motivate our next constructions, Dynaverse and Dynalog, summarized below.

1.2 Dynaverse: A dynamic zk-SNARK from KZG

Our second and central contribution, Dynaverse, is a dynamic zk-SNARK that *is not using recursive SNARKs* and is based solely on KZG commitments [22]. It has $O(\sqrt{n}\log n)$ update time (see Table 1 for comparison). Dynaverse's technical highlights are summarized in the following.

From Plonkish arithmetization [20], recall that the wire assignment of a circuit $\mathcal{C}$ with n addition gates, n multiplication gates, n_0 public inputs and wire-consistency permutation σ (of size $N = 6n + n_0$) can be described with six n-sized vectors $\mathbf{z}_1$, $\mathbf{z}_2$, $\mathbf{z}_3$, $\mathbf{z}_4$, $\mathbf{z}_5$, $\mathbf{z}_6$ and one n_0-sized vector $\mathbf{z}_7$ such that: (i) $\mathbf{z}_1$ and $\mathbf{z}_4$ store the left inputs of addition and multiplication gates respectively; (ii) $\mathbf{z}_2$ and $\mathbf{z}_5$ store the right inputs of addition and multiplication gates respectively; (iii) $\mathbf{z}_3$ and $\mathbf{z}_6$ store the outputs of addition and multiplication gates respectively; (iv) $\mathbf{z}_7$ stores the public inputs. Also recall that $\mathbf{z} = [\mathbf{z}_1\ \mathbf{z}_2\ \mathbf{z}_3\ \mathbf{z}_4\ \mathbf{z}_5\ \mathbf{z}_6\ \mathbf{z}_7]$ is a satisfying assignment of $\mathcal{C}$ if and only if

- $\mathbf{z}[i] = \mathbf{z}[\sigma[i]]$, for all $i = 1, \ldots, 6n + n_0$ (*copy constraint*).
- $\mathbf{z}[i] + \mathbf{z}[n + i] = \mathbf{z}[2n + i]$, for all $i = 1, \ldots, n$ (add *gate constraint*).
- $\mathbf{z}[3n + i] \cdot \mathbf{z}[4n + i] = \mathbf{z}[5n + i]$, for all $i = 1, \ldots, n$ (mult *gate constraint*).

At a very high level, most known zk-SNARKs (e.g., [20]) commit to $\mathbf{z}$ and provide a proof π that $\mathbf{z}$ satisfies all three relations above. Our task is to find a way to do that so that π is efficiently updatable whenever some $\mathbf{z}$ wires change.

First step: Dynamo, a dynamic permutation SNARK. The most crucial piece of our construction is Dynamo, a dynamic permutation SNARK (or permutation argument, as is commonly known) that we build with $O(1)$ proof size and $O(1)$ update time—see Section 4. In particular, Dynamo allows a prover to commit to $\mathbf{z}$ using KZG [22] and provide a proof that the copy constraint is satisfied for a given σ. Dynamo's proof can be updated, whenever say, the $\mathbf{z}$ values of a k-size cycle in σ change, in $O(k)$ time. To the best of our knowledge, this is the first permutation SNARK with such an updatability property, and it could potentially have other applications. To construct Dynamo, the prover KZG-commits to $\mathbf{z}$ with a standard univariate Lagrange polynomial, i.e.,

$$[z(X)] = \left[\sum_i L_i(X) \cdot \mathbf{z}_i\right],$$

where $[z(X)]$ is the KZG commitment of polynomial $z(X)$. The permutation σ is KZG-committed to with another carefully-constructed bivariate polynomial

$$[\sigma(X, Y)] = \left[\sum_i L_i(X) \cdot (Y^i - Y^{\sigma^{-1}(i)})\right],$$

which is held by the verifier. Our main observation is that the polynomial

$$\sum_i \mathbf{z}_i \cdot (Y^i - Y^{\sigma^{-1}(i)})$$

is identically 0 if and only $\mathbf{z}$ satisfies the copy constraint. Dynamo provides a proof for exactly that, on input commitment $[z(X)]$ (from prover) and $[\sigma(X, Y)]$ (from verifier). Importantly, the Dynamo proof consists of 15 group elements (see Table 2), all of which can be expressed as linear combinations of $\mathbf{z}$ and other fixed polynomials (see Theorem 1). Therefore all group elements are efficiently updatable with a single group operation.

Second step: Dynamically enforcing gate constraints. To complete the construction of Dynaverse, what is left to do is provide a proof π that the commitment $\mathbf{z}$ also satisfies the gate constraints, in a way that π is also updatable—see Section 5. The most challenging part of this step is to deal with multiplication constraints: To prove multiplication constraints, we use a standard approach from PLONK [20], namely a zero test on $\{1, \ldots, n\}$ for the polynomial

$$\tau(X) = z(3n + X) \cdot z(4n + X) - z(5n + X)\,,$$

which is done by returning a commitment to the quotient polynomial $A(X) = \tau(X)/\prod_i (X - i)$. Unfortunately the commitment $[A(X)]$ is not efficiently updatable: A single change in $\mathbf{z}$ will completely change the quotient polynomial and therefore the update would take at least linear time (We note here that the same idea applied to addition constraints yields a quotient polynomial that is efficiently updatable, due to the linearity of addition!)

Addressing the expensive division problem: Bucketization. A natural way to address the expensive division problem is to "bucketize" the first $6 \cdot n$ entries of vector $\mathbf{z}$ into $6 \cdot m$ buckets of size m, where $m = \sqrt{n}$. Let $\mathbf{z}_{ij}$ be the m-sized bucket that starts at position $(i-1)n + (j-1)m + 1$ of $\mathbf{z}$ for $i = 1, \ldots, 6$ and $j = 1, \ldots, m$. Now the prover will commit to $6 \cdot m$ buckets outputting $6 \cdot m$ commitments $[z_{ij}(X)]$. Due to this bucketization, we can prove multiplication constraints by providing m commitments of smaller quotient polynomials $[A_i(X)]$ (instead of a single commitment of one large $A(X)$), with the effect that any update can be handled in $m = \sqrt{n}$ time since whenever a wire changes, we only need to update the quotient commitment $[A_i(X)]$ of the bucket that contains it. Therefore the update time of this approach becomes $O(\sqrt{n})$ and the proof size also becomes $O(\sqrt{n})$. Thankfully though, by using a proof system for pairing equations [9], we reduce the proof size and verification time to $O(\log n)$.

Wrapping up: Adjusting the permutation SNARK. We finally note that due to bucketization, we cannot apply Dynamo for copy constraints any more: Dynamo was meant to be applied to the *whole* vector $\mathbf{z}$. Instead, what we do is apply a slight generalization of Dynamo to each bucket $\mathbf{z}_{ij}$, called Dynamix—see Section 4.1. Dynamix provides proofs that the *local* values of $\mathbf{z}_{ij}$ are consistent with the permutation σ. Figure 17 in Appendix I illustrates how Dynaverse performs bucketization and utilizes quotient and Dynamix proofs.

1.3 Dynalog: A dynamic zk-SNARK with polylog update time

In Section 6, we build on the Dynaverse construction to further reduce the update time to polylogarithmic while having a proof size of $\log n$. The main idea is to use a data structure with exponentially-increasing levels to store the deltas of the updates and periodically build this data

structure. To achieve that we require the use of *maintainable* vector commitments, e.g., [34]. All details of the construction can be found in Section 6. We finally note that one can further decrease the proof size and verification time to $O(1)$, by using **one level of recursion**, therefore not compromising security further. We call this adjusted construction Dynalog+.

1.4 From dynamic zk-SNARKs to recursion-free IVC

As a central application of dynamic zk-SNARKs, we propose a compiler from dynamic zk-SNARKs to a version of IVC [6] that is *recursion-free* and has sublinear time per iteration, showing for the first time that non-trivial, fully-secure IVC is possible—see Section 7. Recall that in IVC, there is an initial input z_0 and a function F of size n, and the goal of the prover is to provide a proof for the public statement (i, z_0, z_i), 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 $w_{i-1}, \ldots, w_0$ such that

$$z_i = \mathsf{F}(z_{i-1}, w_{i-1}), z_{i-1} = \mathsf{F}(z_{i-2}, w_{i-2}), \ldots, z_1 = \mathsf{F}(z_0, w_0)\,,$$

where the w_i's denote non-deterministic inputs. An important feature in IVC is that if one is given a proof π_i for the statement (i, z_0, z_i), one should be able to use that proof to derive a proof for $(i+1, z_0, z_{i+1})$. Technically this is implemented by producing a proof π_{i+1} for a recursive SNARK circuit that verifies (i, z_0, z_i) through π_i and simply outputs $z_{i+1} = \mathsf{F}(z_i)$. While space-efficient (the SNARK circuit used is independent of the number of iterations), the security of the above IVC construction faces two challenges, as we discussed before: First, it cannot be proven beyond $\log \lambda$ iterations due to extractor blowup [7] (For example, the security definition of IVC in [23] is stated only for a constant number of iterations.) Second, when efficient recursive zk-SNARKs are used [36], one needs to encode the random oracle in the SNARK circuit, leading to an additional heuristic security argument. In conclusion, concretely-efficient IVC implemented with recursion faces severe security limitations.

Removing recursion at the expense of fixing the number N of IVC iterations: A naive $O(N^2)$ approach. Our proposal is to remove recursion from IVC by considering a different model, one which requires the number N of IVC iterations to be fixed ahead of time. While this seems to be a departure from the original IVC model, it is really not: The original IVC model restricts the number of iterations it can perform to constant or logarithmic due to the aforementioned extractor blowup problem [7], while our model allows this bound to be increased to any polynomial. In this setting, consider the following naive approach to implement IVC.

- There is an initial setup phase where N different SNARK circuits $C_1, \ldots, C_N$ are initialized, where circuit C_i contains i applications of F (hence it has size $O(i)$) starting with z_0 and proves the statement (i, z_0, z_i), without using recursion. Clearly setting up the public parameters for all N circuits would require time $1 + \ldots + N = O(N^2)$.
- Once these parameters have been setup, the prover can produce the proof π_i by running a proof for the SNARK circuit C_i. Crucially, in the proof π_i, it appends all witnesses $w_0, \ldots, w_i$, so that to ensure that π_{i+1} can be computed from π_i—a foundational requirement of IVC.

Clearly, the above approach has severe efficiency and privacy limitations. First, the amount of computation and communication required is $O(N^2)$ and the work per iteration is linear. Additionally, including the witnesses as part of π_i is not zero-knowledge. We observe that both limitations above can be addressed with dynamic zk-SNARKs.

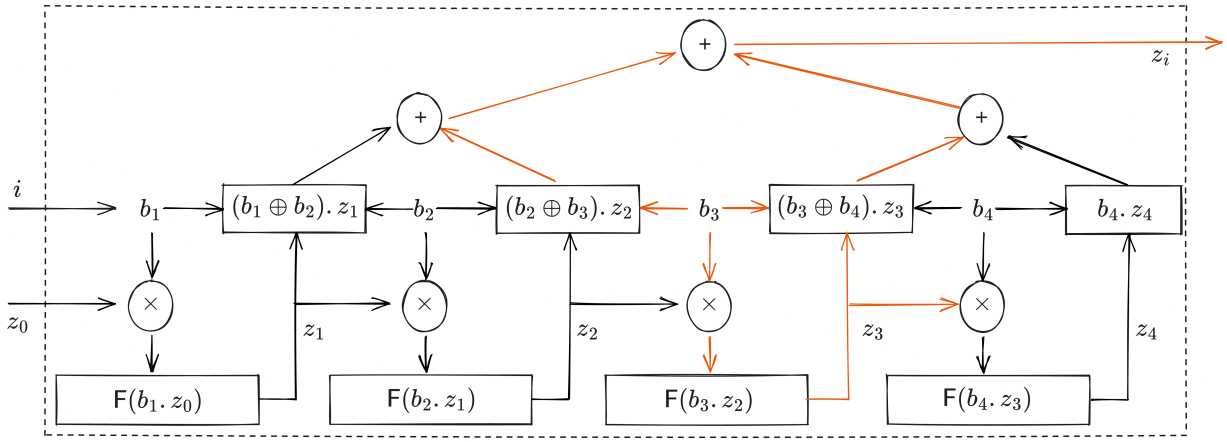

Fig. 1. Circuit F_N for recursion-free IVC using dynamic zk-SNARKs. We do not include non-deterministic input for simplicity. When proceeding from step 2 (1100) to step 3 (1110), only $\tilde{O}(1)$ wires change, indicated with orange above.

Our approach: Efficient recursion-free IVC from dynamic zk-SNARKs. Our idea to address the above inefficiency is to use a dynamic zk-SNARK (e.g., Dynaverse) on a special circuit F_N of size N that has the following properties.

- 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.
- 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 Figure 1.

The general description of the circuit can be found in Figure 9. Given the circuit F_N, it is now natural to build recursion-free IVC using a dynamic zk-SNARK $(\mathcal{G}, \mathcal{P}, \mathcal{U}, \mathcal{V})$ on F_N in the following way.

1. In the setup phase, one can compute the public parameters of F_N by running $\mathcal{G}$. Also we compute an initial proof π_0 for the public statement $(z_0, 0, z_0)$ using algorithm $\mathcal{P}$. Note that π_0 can be computed ahead of time and can be used as a starting point, irrespective of the future inputs z_i.
2. On input π_{i-1}, one can compute the proof π_i using $\mathcal{U}$ from the dynamic SNARK. Because neighboring statements differ by only $\tilde{O}(1)$ number of wires, computing π takes time almost equal to the SNARK update time.

If we use Dynaverse, this construction provides a recursion-free IVC with $\tilde{O}(\sqrt{N})$ step computation and communication time (as opposed to $O(N)$ of the naive approach), and $O(\log N)$ proof size. We also note here, that due to the way Dynaverse prover works, if a single proof every $\sqrt{N}$ iterations is to be produced (as opposed to every single iteration), then our a recursion-free IVC has $\tilde{O}(1)$ amortized step computation and communication time. Clearly, if a recursion-free dynamic zk-SNARK with better update time is constructed in the future, it could serve as drop-in replacement for the IVC construction, leading to better complexities. The detailed construction is described in Section 7.

1.5 Evaluation

In Section 8 we present an initial prototype implementation of Dynarec (using the Plonky2 recursive SNARK [36]) and Dynaverse. Our main findings are:

1. (Dynaverse update v. recomputing with PLONK [20]) Updating a Dynaverse proof is up to approximately 500× faster ($n = 2^{24}$) than recomputing a PLONK proof from scratch. This is to be expected, given the asymptotics. However, Dynaverse verification time and proof size are more larger but reasonable, up to 0.293s for verification and 768 KiB for proof size when $n = 2^{24}$.
2. (Dynaverse v. Dynarec.) Although Dynarec is superior asymptotically, we have found that its prover is 30× to 55× slower than Dynaverse, while also being heuristically-secure, as opposed to Dynaverse, whose security is based on KZG commitments. We expect using more practical implementations of recursion, such as folding schemes with Nova [24], to narrow this gap.

1.6 Other applications of dynamic zk-SNARKs

In this section, we detail other applications of dynamic zk-SNARKs mainly from the blockchain domain.

Updating block proofs. In Ethereum, consider computing a SNARK proof p_i for the following query on block i: What is the number of accounts in block i that have balances greater than 50? The answer changes for block $i + 1$. With dynamic SNARKs, the proof p_{i+1} can be computed 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). This is the motivation behind the Merkle example presented in the introduction. This technology is currently explored by several early-stage companies for verifiably querying on-chain data, including Axiom, Space and Time, Lagrange.

Digest translation. Ethereum data are stored with a Keccak-based authenticated data structure—Merkle Patricia Trie (MPT). Computing SNARK proofs directly on MPTs is notoriously slow, due to Keccak's SNARK unfriendliness. Therefore one can build another MPT using Poseidon, a SNARK-friendly hash, for this purpose. In this case, a proof of digest translation is required, to ensure that the two digests are computed on the same data, yet with different hash functions. This proof will have to be maintained from block to block, and just as before (low entropy from block to block), one can use dynamic SNARKs to do that efficiently. This technology is currently implemented, using Reckle trees [31], in Lagrange's prover network [1].

zkLogin systems. In zkLogin systems [4], users can log into web services without explicitly sending their credentials. This is achieved by providing a zero-knowledge proof for a specific statement/witness pair $(\mathbb{x}, \mathbb{w})$. In particular in the recent zkLogin system [4], the public statement is $\mathbb{x} = (pk_{op}, id, addr, T, vk_u)$ and the witness is $\mathbb{w} = (token, r_1, r_2)$. Whenever a client tries to reconnect to the web service, pk_{op} and id remain the same, and thus a dynamic zk-SNARK could alleviate the cost of recomputing the proof.

Verifiable inference of dynamic models. Consider a situation where an ML service provider has generated a proof π to prove that the inference result for input x is correct with respect to a committed model m. If the model is updated to m', the service provider can use dynamic zk-snarks to update π to prove the inference result on x under the updated model m'. We remark that updating the proofs is particularly relevant for models like decision trees [41], where only a portion of the model changes on retraining, as opposed to deep neural networks, where most weights undergo significant changes.

1.7 Other related work

Authenticated data structures and dynamic vector commitments. Dynamic proof systems have appeared before in the literature but with limited expressiveness. For example, authenticated data structures [35] and updatable vector commitments [13,34] 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., [11], as well as functional vector commitments supporting linear functions, e.g., [12].

Malleable proofs and homomorphic proofs. Another related line of work is that of *malleable proofs* [15]. 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* [3], 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 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 $\mathsf{pp}_{\mathsf{bl}} := (\mathbb{p}, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$ denote the pairing parameters. In particular $\mathbb{G}$ is a group of prime order $\mathbb{p}$, g is a generator of $\mathbb{G}$ and pairing function $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is such that $\forall u, w \in \mathbb{G}$ and $a, b \in \mathbb{Z}_{\mathbb{p}}$, it is $e(u^a, w^b) = e(u, w)^{ab}$. We note here that our actual implementation is using asymmetric pairings for efficiency, but we use symmetric pairings in our presentation for notational convenience.

KZG commitments. Let $(\mathbb{p}, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$ be bilinear pairing parameters and let secret $\alpha \in \mathbb{F}$ be chosen at random. A trusted party outputs the elements $g, g^\alpha, \ldots, g^{\alpha^q}$ for some polynomially-large q. For univariate polynomial $f(X)$ over variable X, the KZG commitment [22] of f is $g^{f(\alpha)}$, which we write as $[f(X)]$. The celebrated KZG commitment [22] allows a prover to commit to a polynomial $f(X)$ via $[f(X)]$ and 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 polynomial $q(X) = f(X)/\prod_{i \in [t]}(X - x_i)$ 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.

KZG variable check. We will be using KZG commitments on bivariate polynomials $f(X, Y)$ as well, in which case the trusted setup outputs $\{g^{\alpha^i \beta^j}\}_{i,j=0,\ldots,q}$, for random α and β, or $\{[X^i Y^j]\}_{i,j=0,\ldots,q}$. Variable check is a useful tool to ensure a polynomial does not contain a specific variable. In particular, when a prover commits to polynomial $f(X)$, we want to ensure that variable Y is not present. To do that, we ask the prover to provide a KZG commitment to $f(X) \cdot Y^q$ as well, and we use the pairing to check whether

$$e([f(X)], [Y^q]) = e([f(X) \cdot Y^q], g)\,.$$

Clearly, if Y was present in $f(X)$, the prover would not have been able to compute $[f(X) \cdot Y^q]$ since the commitment $[Y^{q+1}]$ is not output as part of the setup.

Indexed relations and permutation relation. We use $\mathbb{i}$ to denote an indexed relation, i.e., the description of the circuit checking a public statement $\mathbb{x}$ and a witness $\mathbb{w}$. We slightly abuse notation and write $(\mathbb{x}, \mathbb{w}) \in \mathbb{i}$ iff running $\mathbb{i}$ on $(\mathbb{x}, \mathbb{w})$ returns 1, so $\mathbb{i}$ is both the description of the computation and the set of valid tuples in the language. For example, the indexed relation $\mathbb{i}_{\mathcal{P}} = [n, \sigma]$, where σ is a permutation of size n over domain $\mathbb{F}$, contains those $\mathbb{w}$ such that $\mathbb{w}[i] = \mathbb{w}[\sigma[i]]$ for all i (Note $\mathbb{x} = \emptyset$.)

Plonkish arithmetization. Per Plonkish arithmetization [16,20], an index $\mathbb{i}_{\mathcal{C}} = [n, n_0, \sigma]$ is an indexed relation 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).
- $\sigma \in [6n+n_0]^{6n+n_0}$ is a permutation vector 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 $6n+n_0$. For example, if addition gate i's right input is connected to input wire j, then we may have $\sigma[6n+j] = n+i$.

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

Extractors. Following [21] 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 [21]. For zk-SNARKs with universal setup, algorithm $\mathcal{G}$ below is separated into two algorithms, a universal generation $\mathcal{G}(1^\lambda, |\mathbb{i}|) \to \mathsf{pp}$ and an indexer $\mathcal{I}(\mathsf{pp}, \mathbb{i}) \to (\mathsf{pk}, \mathsf{vk})$. 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 1 (zk-SNARKs). *A zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) for an indexed relation $\mathbb{i}$ is a tuple of PPT algorithms $\mathsf{S} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ with the following interface:*

- $\mathcal{G}(1^\lambda, \mathbb{i}) \to (\mathsf{pk}, \mathsf{vk})$: *Outputs prover key* pk *and verifier key* vk.
- $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{aux})$: *Given proving key* pk*, instance* $\mathbb{x}$*, and witness* $\mathbb{w}$*, outputs proof* π.

– $\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$: *Given verification key* vk*, instance* $\mathbb{x}$*, and a proof* π*, outputs accept or reject.*

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

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

$$\Pr\begin{bmatrix} \mathcal{G}(1^\lambda, \mathbb{i}) \to (\mathsf{pk}, \mathsf{vk}); ((\mathbb{x}, \pi); \mathbb{w}) \leftarrow (\mathcal{A}||\mathcal{E}_\mathcal{A})(\mathsf{pk}, \mathsf{vk}) \\ : \\ \mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 1 \wedge (\mathbb{x}, \mathbb{w}) \notin \mathbb{i} \end{bmatrix}$$

is negligible.
– Zero Knowledge*: Fix any* $\mathbb{i}$ *and* $(\mathbb{x}, \mathbb{w}) \in \mathbb{i}$*. Let* $\mathcal{D}$ *be the distribution of* π *as output by the experiment below.*
  1. $\mathcal{G}(1^\lambda, \mathbb{i}) \to (\mathsf{pk}, \mathsf{vk})$*.*
  2. $\pi \leftarrow \mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w})$*.*

  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.

Algebraic group model and q-DLOG assumption. For our security analysis, we will use the *algebraic group model* from [19]. 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 [19,20], our security also rests on the q-DLOG assumption, which we present in the following.

Assumption 1 (q-DLOG) *Fix integer* q*. 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* $\tau \xleftarrow{\$} \mathbb{F}$*, the probability of* $\mathcal{A}$ *outputting* τ *is* $\mathsf{negl}(\lambda)$*.*

We also present two standard lemmata with respect to the algebraic group model (polynomial check and variable check) in a more general form that will be useful for our proofs—see Appendix E.

3 Dynamic zk-SNARKs definition and a construction based on recursion

In this section we present the formal definition of dynamic zk-SNARKs and Dynarec—a heuristically-secure dynamic zk-SNARKs construction. Our dynamic zk-SNARKs definition (Definition 1) is an extension of the original zk-SNARKs definition 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 $(\mathbb{x}, w)$, 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 2 (Dynamic zk-SNARKs). *A dynamic zero-knowledge succinct non-interactive argument of knowledge (dynamic zk-SNARK) for an indexed relation* $\mathbb{i}$ *is a tuple of PPT algorithms* $\mathsf{DS} = (\mathcal{G}, \mathcal{P}, \mathcal{U}, \mathcal{V})$ *with the following interface:*

- $\mathcal{G}(1^\lambda, \mathbb{i}) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$: *Given* 1^λ *and an indexed relation* $\mathbb{i}$*, outputs a proving key* pk*, an update key* upk *and a verification key* vk*.*
- $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{aux})$: *Given proving key* pk*, instance* $\mathbb{x}$*, and witness* $\mathbb{w}$*, outputs a proof* π *and an extra 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 verification 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* $f(|\mathbb{x}|+|\mathbb{w}|) = o(|x|+|w|)$ *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* $(\mathbb{x}_0, \mathbb{w}_0) \in \mathbb{i}, \ldots, (\mathbb{x}_\ell, \mathbb{w}_\ell) \in \mathbb{i}$*, if* $(\pi_0, \mathsf{aux}_0) \leftarrow \mathcal{P}(\mathsf{pk}, \mathbb{x}_0, \mathbb{w}_0)$ *and,* $(\pi_{i+1}, \mathsf{aux}_{i+1}) \leftarrow \mathcal{U}(\mathsf{upk}, \mathbb{x}_{i+1}, \mathbb{w}_{i+1}, \mathbb{x}_i, \mathbb{w}_i, \pi_i, \mathsf{aux}_i)$*, for* $i = 0, \ldots, \ell - 1$*, then* $\mathcal{V}(\mathsf{vk}, \mathbb{x}_\ell, \pi_\ell) \to 1$*.*
- Knowledge Soundness*: We say that* DS *satisfies* knowledge soundness *if for any PPT adversary* $\mathcal{A}$ *and for any* $\mathbb{i}$ *there exists a PPT extractor* $\mathcal{E}_\mathcal{A}$ *such that*

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

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

[4] Note that for $k = o\left(T(\mathcal{P})/f(|\mathbb{x}| + |\mathbb{w}|)\right)$, 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.

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.

Remark. There could be an alternative definition for zero knowledge which requires that the output of $\mathcal{U}$ is indistinguishable from the output of $\mathcal{P}$. This is a stronger notion but our definition above is still meaningful, e.g., in some blockchain applications it is public knowledge that updates take place. In fact, our constructions also satisfy this indistinguishability definition.

3.1 Dynarec: Dynamic zk-SNARKs from recursive zk-SNARKs

We now provide a dynamic zk-SNARK scheme that satisfies our definition and uses a recursive SNARK as a black box. Our construction is similar to Mangrove [27], that builds a highly parallelizable SNARK from Proof-Carrying-Data (PCD), with necessary modifications to make it updatable (See end of section for a detailed comparison.) As we mentioned, Dynarec is a folklore construction—we give a high-level description here and the full construction is provided in Figure 13 in Appendix B.

At a high level, our protocol for $\mathbb{i} = [n, n_0, \sigma]$ works as follows. Consider a binary tree with n leaves, where each leaf i is a small circuit $\mathcal{C}_i$ verifying the integrity of one addition gate and one multiplication gate, and each internal node is a small circuit $\mathcal{D}$ verifying the proofs of its children (This is where a recursive SNARK is needed.) In particular, for every leaf i, its circuit checks if the gate constraints hold for the i-the addition and multiplication gates. In order to ensure that each leaf circuit i verifies the correct indices from $\mathbf{z}$, we hard-code the 6 indices $\sigma(jn+i)$ for $j \in [0, 5]$ in it, i.e., our construction has a circuit-specific setup phase.[5] Likewise, for internal nodes, we hard-code the specific $(\mathsf{vk}_L, \mathsf{vk}_R)$ keys corresponding to its children node circuits.

The above check guarantees that the input values in each circuit satisfy the gate constraints. We also need to ensure that collectively the values of $\mathbf{z}$ that the prover used satisfy the copy constraints, which may need to span checking across different circuits $\mathcal{C}_i$. For this, we use an incremental multiplicative hash function [18] to calculate the product

$$h_i = \prod_{j \in [0,5]} \frac{\mathsf{H}(m_j, jn + i)}{\mathsf{H}(m_j, \sigma(jn + i))} \tag{1}$$

for each circuit, where H is a random oracle. Subsequently, each internal circuit $\mathcal{D}$ first recursively verifies the two proofs of its children nodes, and then it calculates the product of their hash values. It is easy to see that at the root of the recursion tree the produced hash h_{root} satisfies

$$h_{\mathsf{root}} \cdot \prod_{i \in [n_0]} \frac{\mathsf{H}(\mathbb{x}[i], 6n + i)}{\mathsf{H}(\mathbb{x}[i], \sigma[6n + i])} = 1 \tag{2}$$

if the input values satisfy the copy constraints. The leaf circuit $\mathcal{C}_i$ is shown in Figure 11 in Appendix B and the internal circuit $\mathcal{D}$ is shown in Figure 12 in Appendix B. Note that, to preserve zero-knowledge and avoid revealing the partial product term $h_l \cdot h_r$, the root node takes as public statement the product over the hash values of the public inputs and checks the multiplication result is 1.

[5] It is possible to avoid this by committing to the permutation cycles and corresponding indices in a separate step and then providing them as input to a "generic" leaf circuit, together with their proofs of opening, as in [27]. Our current design choice allows us to simplify the presentation.

Updates of values are handled in a straightforward manner. Each change to an input circuit value corresponds to changes to a set of wire values. For each wire j that is updated, all proofs for the respective leaf circuits that are affected and their ancestor nodes must be recomputed, together with their h_i values, leading to $O(\log n)$ time. Our final protocol is in Figure 13 in Appendix B.

Similarities of Dynarec with Mangrove [27]. A similar approach was used in [27] to build scalable and parallelizable zk-SNARKs. As in our protocol, they also use an incremental hash function to parallelize checking the copy constraints. For efficiency purposes, they instantiate their H as a universal hash function with its parameters being chosen after the witness has been committed, e.g., via a Merkle tree. (This step can then be made non-interactive via the Fiat-Shamir heuristic.) Subsequently, all circuits $\mathcal{C}_i$ need to verify the provided $\mathbb{w}$ entries with respect to the Merkle root. Considering our goal of building dynamic zk-SNARKs, committing to the witness vector $\mathbb{w}$ introduces an important issue to updatability. Each change to $\mathbb{w}$ changes the Merkle tree root, hence, all n circuits $\mathcal{C}_i$ have to be re-computed, making updates as costly as running the original prover. Instead, our adopted approach avoids this, at the cost of embedding costly hash computations in each leaf circuit.

4 Dynamo: A new dynamic permutation SNARK

Our first step towards building a general dynamic zk-SNARK (i.e., for $\mathbb{i}_{\mathcal{C}}$) without using recursive zk-SNARKs is to build a recursion-free dynamic permutation argument for $\mathbb{i}_{\mathcal{P}} = [m, \sigma]$. Indeed, in this section we present Dynamo, a new zero-knowledge dynamic permutation argument for $\mathbb{i}_{\mathcal{P}} = [m, \sigma]$. Dynamo has optimal asymptotic complexities: Its proof size is $O(1)$ and its update complexity is $O(k)$, where k is the Hamming distance between two valid neighboring witness vectors $\mathbf{z}$ and $\mathbf{z}'$, i.e., two vectors $\mathbf{z}$ and $\mathbf{z}'$ satisfying $\mathbf{z}[i] = \mathbf{z}[\sigma[i]]$ and $\mathbf{z}'[i] = \mathbf{z}'[\sigma[i]]$ for $i = 1, \ldots, m$, for fixed σ.

To the best of our knowledge, Dynamo is the first permutation argument that is dynamic—all other permutation arguments (e.g., the one used in PLONK [20]) require at least linear time to handle a small update in the witness. The reason for that is mostly due to Fiat-Shamir, where the randomness used depends on all the entries of the witness, yielding proofs that cannot be efficiently updated.

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

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

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

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

where, for ease of notation, we write

$$\mathbf{y}[i] = \left(Y^i - Y^{\sigma^{-1}[i]}\right)\,.$$

It is easy to see that $(\emptyset, \mathbf{z}) \in \mathbb{i}_{\mathcal{P}} = [m, \sigma]$ if and only if $s(Y) = 0$ for all Y. We build our permutation argument on this idea: In particular, we have a prover commit to vector $\mathbf{z}$ and provide a proof that, for given σ, $s(Y)$ is the zero polynomial.

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

$$z(X) = \sum_{i \in [m]} L_i(X) \cdot \mathbf{z}[i] \tag{3}$$

and $v(X, Y)$ encodes $\mathbf{z}[i] \cdot \mathbf{y}[i]$ ($v(\omega^i, Y) = \mathbf{z}[i]\mathbf{y}[i]$ for all $i = 1, \ldots, m$), i.e.,

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

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.,

$$u(X, Y) = \sum_{i \in [m]} L_i(X) \cdot \mathbf{y}[i] \,. \tag{5}$$

Now to prove that the commitments to the polynomials $z(X)$, $v(X, Y)$ and $u(X, Y)$ satisfy $s(Y) = 0$ the prover must provide two additional proofs (in addition to the commitment of z 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 (as in PLONK [20]) for the polynomial $v(X, Y) - u(X, Y) \cdot z(X)$ on the set (Ω, Y), where $\Omega = \{\omega, \ldots, \omega^m\}$. The proof for that is a KZG commitment to the quotient polynomial

$$\alpha(X, Y) = \frac{v(X, Y) - u(X, Y) \cdot z(X)}{X^m - 1} \,. \tag{6}$$

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

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

Here we cannot use existing techniques from PLONK [20] since, as we mentioned, we must avoid Fiat-Shamir. The main idea is to have the prover commit to a "prefix polynomial" $p(X, Y)$ such that its evaluation at (ω^i, Y) equals the sum of the first i terms of the above sum, i.e.,

$$p(X, Y) = \sum_{i \in [m]} L_i(X) \sum_{j=1}^{i} v(\omega^j, Y) \,. \tag{7}$$

Now it is enough to have the prover prove that (i) $p(\omega, Y) = v(\omega, Y)$ (first term equality); (ii) $p(\omega^m, Y) = 0$ (sum-check correctness); (iii) and the following "prefix recursion"

$$p(\omega^i, Y) = p(\omega^{i-1}, Y) + v(\omega^i, Y) \text{ for all } i = 2, \ldots, m \,. \tag{8}$$

The first two relations are straightforward to prove using a standard KZG evaluation proof: For (i), the proof is a KZG commitment to the quotient polynomial

$$\beta(X,Y) = \frac{p(X,Y) - v(X,Y)}{X - \omega} \tag{9}$$

and for (ii), the proof is a KZG commitment to the quotient polynomial

$$\gamma(X,Y) = \frac{p(X,Y)}{X - 1}\,. \tag{10}$$

Computing a proof for prefix recursion is more involved, and we describe it next.

A proof system for prefix recursion. For Eq. (8) prefix recursion, the prover provides commitments to polynomials $p(X,Y)$ and

$$t(X,Y) = p(X \cdot \omega^{-1}, Y) \tag{11}$$

as well as $v(X,Y)$ and must also provide a proof that these commitments satisfy Equation 8. Note that as long as $t(X,Y) = p(X \cdot \omega^{-1}, Y)$, prefix recursion is reduced to a simple zero-check of $p(X,Y) - t(X,Y) - v(X,Y)$ on the set $\{(\omega^2, Y), \ldots, (\omega^m, Y)\}$. However it is easy to see that $p(X,Y) - t(X,Y) - v(X,Y)$ is identically 0, and therefore there is no need to provide a quotient polynomial—but the verifier will still need to check that this is the case.

So the fundamental problem remaining to solve is to design a proof system for "polynomial displacement": Given two commitments to polynomials $p(X,Y)$ and $t(X,Y)$ how can we prove that $t(X,Y) = p(X \cdot \omega^{-1}, Y)$?

A proof system for polynomial displacement. Using ideas from PLONK [20] we can solve polynomial displacement with Fiat-Shamir: For random r, KZG-evaluate $p(X,Y)$ at $(\alpha \cdot r, Y)$ and $t(X,Y)$ at (r,Y), and check whether the evaluations are the same. Since we cannot use Fiat-Shamir, we follow a different approach. We will use an additional variable W. The prover, along with commitments to $p(X,Y)$ and $t(X,Y)$, it provides a commitment to another polynomial

$$g(W,Y) = p(W,Y)\,. \tag{12}$$

To check that the two commitments $p(X,Y)$ and $g(W,Y)$ refer to the same polynomial, the prover provides a commitment to the quotient polynomial

$$\delta(X,W,Y) = \frac{p(X,Y) - g(W,Y)}{X - W}\,. \tag{13}$$

The final check is to ensure that the evaluation of $g(W,Y)$ at point $X \cdot \omega^{-1}$ is equal to $t(X,Y)$. This is easy to do by providing a commitment to the following

$$\varepsilon(X,W,Y) = \frac{g(W,Y) - t(X,Y)}{W - X \cdot \omega^{-1}}\,. \tag{14}$$

Final variable check. The final thing that the prover must do is variable checks for the polynomials $z(X)$, $v(X,Y)$, $p(X,Y)$, $t(X,Y)$ and $g(W,Y)$ as described in Appendix E. Let $Z(X,W,Y)$,

Table 2. The 15 polynomial commitments contained in the Dynamo proof.

Commitments	$z(X)$ Eq. (3)	$v(X,Y)$ Eq. (4)	$p(X,Y)$ Eq. (7)	$t(X,Y)$ Eq. (11)	$g(W,Y)$ Eq. (12)
Variable Checks	$Z(X,W,Y)$	$V(X,W,Y)$	$P(X,W,Y)$	$T(X,W,Y)$	$G(X,W,Y)$
Quotients	$\alpha(X,Y)$ Eq. (6)	$\beta(X,Y)$ Eq. (9)	$\gamma(X,Y)$ Eq. (10)	$\delta(X,W,Y)$ Eq. (13)	$\varepsilon(X,W,Y)$ Eq. (14)

$V(X,W,Y)$, $P(X,W,Y)$, $T(X,W,Y)$ and $G(X,W,Y)$ be the polynomials committed for that purpose. For example, $Z(X,W,Y) = z(X) \cdot Y^m \cdot W^m$. The other commitments are computed similarly.

Summary. The final Dynamo proof for $\mathbb{i}_{\mathcal{P}} = [m, \sigma]$ consists of 15 group elements that are KZG commitments of 5 committed polynomials, 5 variable checks and 5 committed quotient polynomials—see Table 2.

Computing and updating the proof. There are closed formulas for all polynomials of Table 2. In particular, we have the following theorem, whose proof can be found in Appendix F.

Theorem 1. *Let $\mathcal{F} = \{z, v, p, t, g, Z, V, P, T, G, \alpha, \beta, \gamma, \delta, \varepsilon\}$ be the set of polynomials from Table 2. Every polynomial $f \in \mathcal{F}$ from can be expressed as*

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

where $\{f_1, \ldots, f_m\}$ is a fixed set of m polynomials.

We note here that Theorem 1 allows us not only to easily compute the Dynamo proof (e.g., without any division) but also to update the proof in constant time whenever a value $\mathbf{z}[i]$ changes.

Final protocol. Our complete circuit-specific Dynamo protocol is shown in Figure 2. We summarize our protocol in the following theorem.

Theorem 2 (Dynamo)**.** *The protocol of Figure 2 is a dynamic SNARK (per Definition 2) for $\mathbb{i}_{\mathcal{P}} = [m, \sigma]$ assuming q-DLOG (see Assumption 1) in the AGM model. Its complexities are as follows.*

1. *$\mathcal{G}$ runs in $O(m)$ time, outputs pk and upk 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;*
3. *$\mathcal{U}$ runs in $O(k)$ time, where k is the Hamming distance of $\mathbb{w}$ and $\mathbb{w}'$;*
4. *$\mathcal{V}$ runs in $O(1)$ time.*

Proof. Completeness and updatability follow naturally from the construction. For knowledge soundness, we define the following extractor $\mathcal{E}_{\mathcal{A}}(\mathsf{pk}, \mathsf{vk})$:

1. Run the algebraic adversary $(\mathbb{x}, \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk})$.
2. Parse $[z]$ from π. Note that since $\mathcal{A}$ is algebraic, it should also outputs vectors to show how $[z]$ can be computed from pk. Thus $\mathcal{E}_{\mathcal{A}}$ can reconstruct $\widetilde{z}(X)$ such that $[z] = [\widetilde{z}(X)]$. Abort if $\mathsf{deg}_Y \widetilde{z} > 0$ or $\mathsf{deg}_W \widetilde{z} > 0$.
3. Output $\mathbb{w} \in \mathbb{F}^m$ where $\mathbb{w}[i] = \widetilde{z}(\omega^i), \ \forall i \in [m]$.

– $\mathcal{G}(1^\lambda, [m, \sigma]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - $\mathsf{pp}_{\mathsf{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$;
  - Let $\mathcal{F} = \{z, v, p, t, g, Z, V, P, T, G, \alpha, \beta, \gamma, \delta, \varepsilon\}$ be the set of polynomials from Theorem 1.
  - Pick random a, b, c from $\mathbb{F}$ for variables X, Y and W respectively.
  - Set $\mathsf{pk} = \mathsf{upk}$ to contain the following KZG commitments, defined in Theorem 1, and computed using a, b and c directly

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

  - Set $\mathsf{vk} = \{[u(X, Y)], [X], [W], [X^m], [Y^m W^m], [W^m]\}$ (u is from Eq. (5)).

– $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{aux})$:
  - Parse $\mathbb{x}$ as $\emptyset$ and $\mathbb{w}$ as $\mathbf{z}[1], \ldots, \mathbf{z}[m]$.
  - Following Theorem 1, output $|\mathcal{F}| = 15$ KZG commitments as π and aux, i.e., for all $f \in \mathcal{F}$ output

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

– $\mathcal{U}(\mathsf{upk}, \mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}, \pi, \mathsf{aux}) \to (\pi', \mathsf{aux}')$:
  - Parse $\mathbb{w}$ as $\mathbf{z}$ and $\mathbb{w}'$ as a new valid witness $\mathbf{z}'$. Parse π and aux as $\{[f]\}_{f \in \mathcal{F}}$.
  - Let J be the set of locations that $\mathbf{z}$ and $\mathbf{z}'$ differ and let $\{\delta_j\}_{j \in J}$ be the corresponding deltas. Output as π' and aux' the new KZG commitments $\{[f']\}_{f \in \mathcal{F}}$ where

$$[f'] = [f] \cdot \prod_{j \in J} [f_j]^{\delta_j} .$$

– $\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$:
  - Parse vk and π as output by $\mathcal{G}$ and $\mathcal{P}$ respectively.
  - Output 1 if and only if all the following relations hold:

$$\begin{aligned}
&e([v], g) \cdot e([-u], [z]) = e([\alpha], [X^m - 1]).\\
&e([p], g) \cdot e([-v], g) = e([\beta], [X - \omega]).\\
&e([p], g) = e([\gamma], [X - 1]).\\
&[p] \cdot [-t] \cdot [-v] = 1_{\mathbb{G}}.\\
&e([p] \cdot [-g], g) = e([\delta], [X - W]).\\
&e([g] \cdot [-t], g) = e([\varepsilon], [W - X \cdot \omega^{-1}]).\\
&e([z], [Y^m W^m]]) = e([Z], g).\\
&e([v], [W^m]]) = e([V], g).\\
&e([p], [W^m]]) = e([P], g).\\
&e([t], [W^m]]) = e([T], g).\\
&e([g], [X^m]]) = e([G], g).
\end{aligned}$$

Fig. 2. The Dynamo SNARK.

Since verification accepts, all checks in the $\mathcal{V}$ algorithm of Figure 2 pass. Parse $\pi = \{[f]\}_{f \in \mathcal{F}}$. Since $\mathcal{A}$ is algebraic, it should also output vectors to show how $[f]$ can be computed from pk, and

then we can reconstruct $\{\widetilde{f}(X,W,Y)\}_{f\in\mathcal{F}}$ such that $[f]=[\widetilde{f}(X,W,Y)]$. From the zero-checks and variable checks, we have the following equations.

$$\widetilde{v}(X,Y)-\widetilde{u}(X,Y)\widetilde{z}(X)=\widetilde{\alpha}(X,Y,W)(X^m-1) \tag{15a}$$
$$\widetilde{p}(X,Y)-\widetilde{v}(X,Y)=\widetilde{\beta}(X,Y,W)(X-\omega) \tag{15b}$$
$$\widetilde{p}(X,Y)=\widetilde{\gamma}(X,Y,W)(X-1) \tag{15c}$$
$$\widetilde{p}(X,Y)-\widetilde{t}(X,Y)-\widetilde{v}(X,Y)=0 \tag{15d}$$
$$\widetilde{p}(X,Y)-\widetilde{g}(W,Y)=\widetilde{\delta}(X,W,Y)(X-W) \tag{15e}$$
$$\widetilde{g}(W,Y)-\widetilde{t}(X,Y)=\widetilde{\varepsilon}(X,W,Y)(W-X\cdot\omega^{-1}) \tag{15f}$$

From Eqs. (15e) and (15f), we have

$$\begin{aligned}\widetilde{p}(\omega^i,Y)&=\widetilde{g}(\omega^i,Y)\\ \widetilde{g}(\omega^{i-1},Y)&=\widetilde{t}(\omega^i,Y)\end{aligned} \tag{16}$$

By combining Eq. (16) with Eqs. (15b) to (15d), we have

$$\begin{aligned}&\widetilde{p}(\omega,Y)=\widetilde{v}(\omega,Y)\\ &\widetilde{p}(\omega^i,Y)=\widetilde{p}(\omega^{i-1},Y)+\widetilde{v}(\omega^i,Y),\ i\in[2,m]\\ &\widetilde{p}(1,Y)=0\end{aligned} \tag{17}$$

From Eq. (15a), we have $u(\omega^i,Y)\widetilde{z}(\omega^i)=\widetilde{v}(\omega^i,Y)$. Therefore,

$$\widetilde{p}(1,Y)=\sum_{i\in[m]}\widetilde{v}(\omega^i,Y)=\sum_{i\in[m]}u(\omega^i,Y)\widetilde{z}(\omega^i)=\sum_{i\in[m]}\mathbf{y}[i]\widetilde{z}(\omega^i)=0\,,$$

which means the output $\mathbb{w}$ of $\mathcal{E}_{\mathcal{A}}$ should be a valid witness. The complexities of $\mathcal{P},\mathcal{U},\mathcal{V}$ follow naturally from the protocol. For the runtime of $\mathcal{G}$, since this algorithm knows a, b and c, it can compute everything in linear time. □

Universal protocol. We note that our protocol can be turned into a universal protocol, introducing an $\mathcal{I}$ algorithm. Unfortunately, in the universal version of our protocol the time complexity of both $\mathcal{G}$ and $\mathcal{I}$ is $\widetilde{O}(m^2)$ (This is not an issue for our final protocol, since we apply the permutation argument in buckets.) The proof of the following lemma can be found in Appendix F.

Lemma 1 (Universal Dynamo). *There exists a universal version of* Dynamo *whose (i) $\mathcal{G}$ algorithm runs in $O(m^2)$ time and outputs public parameters of $O(m^2)$ size; (ii) $\mathcal{I}$ algorithm runs in $O(m^2)$ time and outputs* pk *of $O(m)$ size and* vk *of $O(1)$ size. All other complexities are the same.*

Finally note, that by following the ideas from [20,33], we can use random masks for the polynomials and add zero-knowledge to Dynamo. The proof of the lemma below can be found in Appendix F.

Lemma 2 (Zero-knowledge Dynamo). *There is a zero-knowledge version of* Dynamo *with the same complexities.*

4.1 Dynamix: A generalization of Dynamo

As we will see in the next section, our final dynamic zk-SNARK protocol will have to apply the permutation argument on $\sqrt{N}$ subvectors of $\mathbf{z}$ (each one containing $m = \sqrt{N}$ values of $\mathbf{z}$) and therefore we will be using a slight generalization of Dynamo which we call Dynamix (We need this generalization because the overall permutation condition holds for the whole vector $\mathbf{z}$ and not for subvectors of $\mathbf{z}$.) Recall that Dynamo provided a way for a prover to convince a verifier that, for a given σ it knows a vector $\mathbf{z}$ such that

$$\sum_{i\in[m]} \mathbf{z}[i] \cdot (Y^i - Y^{\sigma^{-1}[i]}) = 0\,.$$

With Dynamix, we make two changes to the above relation. First, we allow the exponents of Y to take arbitrary values $s_i \in [N]$ and $t_i \in [N]$ for some N that potentially is not equal to m. Second, we require that instead of 0, the sum equals another polynomial $h(Y)$, whose commitment the prover will provide. We therefore define the Dynamix relation as $\mathbb{i}_{\mathcal{D}} = [m, N, \mathbf{s}, \mathbf{t}]$ (where $\mathbf{s} = [s_i]_{i\in[m]}$, $\mathbf{t} = [t_i]_{i\in[m]}$ and $s_i, t_i \in [N]$) to contain $(\emptyset, (\mathbf{z}, h))$ such that

$$\sum_{i\in[m]} \mathbf{z}[i] \cdot (Y^{s_i} - Y^{t_i}) = h(Y)\,. \tag{18}$$

We can trivially extend Dynamo to Dynamix in the following fashion.

1. Instead of using $\mathbf{y}[i] = Y^i - Y^{\sigma^{-1}(i)}$ in KZG commitments $[v]$ (Eq. (4)) and $[u]$ (Eq. (5)), we use $\mathbf{y}[i] = Y^{s_i} - Y^{t_i}$.
2. Since the right-hand side of the sum-check equation changes from 0 to $h(Y)$, the prover must commit to an additional polynomial $h(Y)$ as defined in Equation 18. Note that along with $[h(Y)]$, the prover provides a commitment $[H(Y)]$ for variable check.
3. Since the right-hand side of the sum-check equation changes from 0 to $h(Y)$, the quotient polynomial $\gamma(X, Y)$ (Eq. (10)) is now computed as $\gamma(X, Y) = (p(X, Y) - h(Y))/(X - 1)$.
4. Note that the polynomial $p(X, Y) - t(X, Y) - v(X, Y)$ is not identically 0 anymore. We will therefore need a zero-check on $\{(\omega^2, Y), \ldots, (\omega^m, Y)\}$. In particular, the prover will have to commit to $(p(X, Y) - t(X, Y) - v(X, Y))/((X^m - 1)(X - \omega)^{-1})$, which after careful calculation, this is equal to $-\omega \cdot h(Y)/m$—therefore this part does not require a new commitment since the prover has already committed to $h(Y)$.
5. The third pairing equation in $\mathcal{V}$ becomes $e([p] \cdot [-h], g) = e([\gamma], [X - 1])$.

All in all, the Dynamix proof contains 17 group elements, 2 more than the Dynamo one. We provide the detailed Dynamix protocol in Figure 16 in the Appendix. Zero-knowledge and universality follow exactly in the same way as in Dynamo.

Theorem 3 (Dynamix)**.** *The protocol of Figure 16 is a dynamic SNARK (per Definition 2) for* $\mathbb{i}_{\mathcal{D}} = [m, N, \mathbf{s}, \mathbf{t}]$ *assuming q-DLOG (see Assumption 1) in the AGM model. Its complexities are as follows.*

1. $\mathcal{G}$ *runs in* $O(\min\{m \log N, N\})$ *time and outputs* pk *and* upk *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;*
3. $\mathcal{U}$ *runs in* $O(k)$ *time, where* k *is the Hamming distance of* $\mathbb{w}$ *and* $\mathbb{w}'$*;*
4. $\mathcal{V}$ *runs in* $O(1)$ *time.*

Proof. The proof is almost the same as the proof of Theorem 2. The main difference is the runtime of $\mathcal{G}$: In $\mathcal{G}$, we need to calculate $\{b^{s_i}, b^{t_i}\}_{i\in[m]}$ where s_i, t_i are bounded by N. We can either compute $\{b^j\}_{j\in[N]}$ in $O(N)$ time, or compute each b^{s_i}, b^{t_i} in $O(\log N)$ time through a fast exponentiation trick for every $i \in [m]$. □

Lemma 3 (Universal Dynamix). *There exists a universal version of* Dynamix *whose (i) $\mathcal{G}$ algorithm runs in $O(m \cdot \min\{m \log N, N\})$ time and outputs public parameters of $O(m^2)$ size; (ii) $\mathcal{I}$ algorithm runs in $O(m^2)$ time and outputs* pk *and* upk *of $O(m)$ size and* vk *of $O(1)$ size. All other complexities are the same.*

The proof of Lemma 3 is a based on the proofs of Lemma 1 and Theorem 3. Finally, by the same zk-masking techniques as in Dynamo, we have the following.

Lemma 4 (Zero-knowledge Dynamix). *There is a zero-knowledge version of* Dynamix *with the same complexities.*

5 Dynaverse: A dynamic zk-SNARK without recursion

In this section, we present Dynaverse, a general-purpose dynamic zk-SNARK (i.e., for $\mathbb{i} = [n, n_0, \sigma]$) without recursion. Dynaverse is using the Dynamix SNARK from Section 4.1. Dynaverse is a circuit-specific dynamic zk-SNARK that has $O(n)$ setup time, $O(k \cdot \sqrt{n} \log n)$ update time (where k is the Hamming distance between the statements) and $O(\log n)$ proof size. The universal version of Dynaverse has $O(n\sqrt{n})$ universal setup time and $O(n)$ circuit-setup time. We note here that Dynaverse's update algorithm is trivially parallelizable.

Background and problem with PLONK approach. Recall that for any fixed index $\mathbb{i}_{\mathcal{C}} = [n, n_0, \sigma]$ 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 set $\mathbf{z} = [\mathbb{w}; \mathbb{x}] \in \mathbb{F}^{6n+n_0}$. We have $(\mathbb{x}, \mathbb{w}) \in \mathbb{i}_{\mathcal{C}}$ if and only if the following hold:

- *Copy constraint:* $(\emptyset, \mathbf{z}) \in \mathbb{i}_{\mathcal{P}} = [6n + n_0, \sigma]$.
- *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 [20]. Recall that in PLONK, instead of committing to one large vector $\mathbf{z}$, the prover commits to six vectors of size n (Subvector $\mathbf{z}_7$ is of size n_0 and contains the public statement.) In particular one can write $\mathbf{z}$ as

$$[\mathbf{z}_1\ \mathbf{z}_2\ \mathbf{z}_3\ \mathbf{z}_4\ \mathbf{z}_5\ \mathbf{z}_6\ \mathbf{z}_7],$$

where $\mathbf{z}_1$ holds the left inputs of all addition gates, $\mathbf{z}_2$ holds the right inputs of all addition gates, $\mathbf{z}_3$ holds the outputs of all addition gates, $\mathbf{z}_4$ holds the left inputs of all multiplication gates, $\mathbf{z}_5$ holds the right inputs of all multiplication gates, and $\mathbf{z}_6$ holds the outputs of all multiplication gates. Let $z_t(X)$ (for $t = 1, \ldots, 6$) be the Lagrange polynomials that the prover uses to commit to those subvectors (We use φ to denote the n-th root of unity used in those Lagrange polynomials.)

Clearly, to prove that the committed polynomials $z_t(X)$ satisfy the gate constraints we need to prove that

$$z_1(X) + z_2(X) = z_3(X) \text{ for all } X = \varphi, \ldots, \varphi^n\,,$$

$$z_4(X) \cdot z_5(X) = z_6(X) \text{ for all } X = \varphi, \ldots, \varphi^n\,.$$

Note that the addition constraint is easy to check due to the fact that $[z_t(X)]$'s are additively homomorphic and therefore all the verifier has to do is to check whether $[z_3(X)] = [z_1(X)] \cdot [z_2(X)]$. Similarly, the prover can update the commitments in constant 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 $z_4(X) \cdot z_5(X) - z_6(X)$ on the set $\Phi = \{\varphi, \ldots, \varphi^n\}$ which can be done via a commitment to the quotient polynomial

$$A(X) = \frac{z_4(X) \cdot z_5(X) - z_6(X)}{X^n - 1}\,.$$

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

Our main technique: Enforcing gate constraints on subvectors. To address the linear update time of the multiplication gate constraints update, we follow a natural approach. We divide each vector $\mathbf{z}_t$ into the $m = \sqrt{n}$ succesive subvectors $\mathbf{z}_{t1}, \ldots, \mathbf{z}_{tm}$ of m values each. The prover will therefore provide m polynomial commitments for each vector $\mathbf{z}_t$ (note that for these commitments we are using m-th roots of unity), for a total of $6 \cdot m$ commitments, i.e., the commitments $[z_{t1}(X)], \ldots, [z_{tm}(X)]$ for $t = 1, \ldots, 6$. First, notice that the addition constraint is handled exactly as before.

For the multiplication constraint, the prover must now provide m commitments to the following quotient polynomials

$$A_i(X) = \frac{z_{4i}(X) \cdot z_{5i}(X) - z_{6i}(X)}{X^m - 1} \text{ for } i = 1, \ldots, m\,. \tag{19}$$

Wrapping up: Enforcing copy constraints across subvectors. Note now that the final thing that the prover must do is to convince the verifier that

$$[z_{t1}(X)], \ldots, [z_{tm}(X)] \text{ for } t = 1, \ldots, 6$$

are consistent with σ and $[z_7(X)]$—the commitment of the public input computed by the verifier. To do that, we will do $6 \cdot m$ invocations of the Dynamix protocol from Section 4.1, one for each one of the $6 \cdot m$ subvectors. In particular, for vector $\mathbf{z}_{ti}$ that covers the m-sized range $[x, y]$ from the original $\mathbf{z}$ vector (in particular $x = (t-1) \cdot n + (i-1)m + 1$ and $y = x + m - 1$), set

$$\mathbf{s}_{ti} = [x\ x+1 \ldots y] \text{ and } \mathbf{t}_{ti} = [\sigma^{-1}(x)\ \sigma^{-1}(x+1) \ldots \sigma^{-1}(y)]\,. \tag{20}$$

Then the prover will output the proof that is output by a Dynamix for $[m, N, \mathbf{s}_{ti}, \mathbf{t}_{ti}]$.

Final proof and verification. The final proof consists of $6 \cdot m$ Dynamix proofs and m commitments to the quotient polynomials from Equation 19. To verify the final proof the verifier first verifies the Dynamix proofs and the quotient polynomials A_i. Then the verifier computes a commitment to the h polynomial corresponding to the public statement, i.e.,

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

– $\mathcal{G}(1^\lambda, [n, n_0, \sigma]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - $\mathsf{pp_{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$.
  - Set $m = \sqrt{n}$ and $N = 6n + n_0$.
  - Pick random a, b, c from $\mathbb{F}$ for variables X, Y and W respectively.
  - For $t = 1, \ldots, 6$, for $i = 1, \ldots, m$, call

$$\mathcal{G}'_{a,b,c,\mathsf{pp_{bl}}}(1^\lambda, [m, N, \mathbf{s}_{ti}, \mathbf{t}_{ti}]) \to (\mathsf{pk}_{ti}, \mathsf{upk}_{ti}, \mathsf{vk}_{ti}) \,,$$

    where $\mathbf{s}_{ti}$ and $\mathbf{t}_{ti}$ are defined in Equation 20.
  - Set $\mathsf{pk} = \{\mathsf{pk}_{ti}\}_{t,i}$, $\mathsf{upk} = \{\mathsf{upk}_{ti}\}_{t,i}$ and $\mathsf{vk} = \{\mathsf{vk}_{ti}\}_{t,i}$.

– $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{aux})$:
  - Parse $\mathbb{x}$ as $\mathbf{z}_7$ and $\mathbb{w}$ as $\{\mathbf{z}_{t1}, \ldots, \mathbf{z}_{tm}\}_{t=1,\ldots,6}$.
  - For $t = 1, \ldots, 6$, for $i = 1, \ldots, m$ call $\mathcal{P}'(\mathsf{pk}_{ti}, \emptyset, \mathbf{z}_{ti}) \to (\pi_{ti}, \mathsf{aux}_{ti})$.
  - For $i = 1, \ldots, m$, compute the commitments $[A_i(X)]$ as in Equation 19.
  - Proof π contains $\{\pi_{ti}\}_{t,i}$ and $\{[A_i(X)]\}_i$, and aux contains $\{\mathsf{aux}_{ti}\}_{t,i}$.

– $\mathcal{U}(\mathsf{upk}, \mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}, \pi, \mathsf{aux}) \to (\pi', \mathsf{aux}')$:
  - 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 $\mathbf{z}_{ti}$.
  - For every $(t, i) \in C$ call

$$\mathcal{U}'(\mathsf{upk}_{ti}, \emptyset, \mathbf{z}'_{ti}, \emptyset, \mathbf{z}_{ti}, \pi_{ti}, \mathsf{aux}_{ti}) \to (\pi'_{ti}, \mathsf{aux}'_{ti}) \,.$$

  - Let $I = \{i : (t, i) \in C \wedge t \geq 4\}$.
  - For every $i \in I$ recompute $[A'_i(X)]$ as in Equation 19.
  - Output the updated proofs $\{\pi'_{ti}\}$ and the updated commitments $[A'_i(X)]$ as the updated proof π' and $\{\mathsf{aux}'_{ti}\}$ as aux'.

– $\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$:
  - Extract from π the commitments $[z_{ti}(X)]$ for $t = 1, \ldots, 6$ and $i = 1, \ldots, m$;
  - Check the addition constraints, i.e., that for all $i = 1, \ldots, m$ it is

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

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

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

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

$$\mathcal{V}'(\mathsf{vk}_{ti}, \emptyset, \pi_{ti}) \to 1 \,.$$

  - Parse $\mathbb{x}$ as $\mathbf{z}[6n + 1 \ldots 6n + n_0]$. Set $h_{\mathbb{x}}(Y) = \sum_{i=6n+1}^{6n+n_0} \mathbf{z}[i](Y^i + Y^{\sigma^{-1}(i)})$. Check whether $[h_{\mathbb{x}}(Y)] \cdot \prod_{t=1}^{6} \prod_{i=1}^{m} [h_{ti}(Y)] = 1_{\mathbb{G}}$, where $[h_{ti}(Y)]$ is extracted from the Dynamix proof π_{ti}.

Fig. 3. Dynaverse SNARK using Dynamix SNARK $(\mathcal{G}', \mathcal{P}', \mathcal{V}')$ as a black box.

and checks whether

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

where $[h_{ti}]$ is the commitment to the polynomial h corresponding to the Dynamix proof for the $\mathbf{z}_{ti}$ vector. Fig. 17 in Appendix I provides a pictorial description of Dynaverse. Our complete protocol is shown in Figure 3.

Theorem 4 (Dynaverse)**.** *The protocol of Figure 3 is a dynamic SNARK (per Definition 2) for* $\mathbb{i}_{\mathcal{C}} = [n, n_0, \sigma]$ *assuming 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 Dynamix to extract the witness which satisfies the copy constraints. The verification algorithm can also ensure that this extracted witness also satisfies the gate constraints. The complexities of $\mathcal{P}$,$\mathcal{U}$ and $\mathcal{V}$ follow naturally from the protocol. For the runtime of $\mathcal{G}$, although the runtime of $\mathcal{G}$ in Dynamix is $O(\min\{m \log N, N\})$, we can directly compute $\{b^j\}_{j \in [N]}$ and let each $\mathcal{G}'$ reuse these values. Hence, the runtime of $\mathcal{G}$ is $O(N + m^2) = O(n + n_0)$. □

Lemma 5 (Universal Dynaverse**).** *There is a universal version of* Dynaverse *whose (i)* $\mathcal{G}$ *algorithm runs in* $O(n\sqrt{n})$ *time and outputs public parameters of* $O(n\sqrt{n})$ *size; (ii)* $\mathcal{I}$ *algorithm runs in* $O(n\sqrt{n})$ *time and outputs* pk *of* $O(n)$ *size and* vk *of* $O(\sqrt{n} + n_0)$ *size. All other complexities are the same.*

Lemma 6 (Zero-knowledge Dynaverse**).** *There is a zero-knowledge version of* Dynaverse *with the same complexities.*

The proof for zero-knowledge is presented in Appendix F.

Concretely reducing the Dynaverse **proof size.** One of the main drawback of Dynaverse is that the constant in the $\widetilde{O}(\sqrt{n})$ update time/proof size is too large. More specifically, we have $6\sqrt{n}$ sub-vectors, and for copy constraints we need to run Dynamix for each sub-vector. One Dynamix proof contains 17 groups elements and for one update we need to update all of them. Overall, a Dynaverse proof contains $17 \times 6\sqrt{n} = 102\sqrt{n}$ groups elements. However, for gate constraints, we only need $\sqrt{n}$ groups elements—$[A_i]$. To concretely improve the proof size we run Dynamix on $O(1)$ length-$O(n)$ sub-vectors while still remaining $O(\sqrt{n} \log n)$ update time for gate constraints. For that, we face the following problems.

1. For the universal setup of Dynamix, the runtime of $\mathcal{G}$ and $\mathcal{I}$ and the size of pp output by $\mathcal{I}$ will go to $O(n^2)$.
2. We need to deal with the inconsistency between $[z]$ for Dynamix of length $O(n)$ and $[z]$ for gate constraints of length $O(\sqrt{n})$.

For the first problem, there is no simple way to avoid that so we decide to consider only circuit-specific setup for this optimization, and then the runtime of $\mathcal{G}$ for Dynamix is $O(n)$. For the second problem, we need to introduce a natural trick to show the consistency. See Figure 18 in Appendix I. First, we split $\mathbf{z}$ into $[\mathbf{z}_i]_{i\in[1,6]}$ of 6 n-sized vectors. For $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3$, we are dealing with add gates, i.e.,

$$\mathbf{z}_1[i] + \mathbf{z}_2[i] = \mathbf{z}_3[i], \ \forall i \in [n] \,.$$

Therefore for $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3$ we do not further partition to achieve sublinear time.

In order to achieve sublinear update time for $\mathbf{z}_4, \mathbf{z}_5, \mathbf{z}_6$ (multiplication gates), we follow the idea in the original Dynaverse to partition them into sub-vectors of length $\sqrt{n}$. However, we still want to apply the protocol for copy constraints on the whole vectors $\mathbf{z}_4, \mathbf{z}_5, \mathbf{z}_6$, so we use two-layer interpolations (same as the technique introduced in [39]): In particular, $\forall i \in \{4, 5, 6\}$, let $m = \sqrt{n}$,

$$z_i(X_1, X_2) = \sum_{j\in[m]} \sum_{k\in[m]} L_j(X_1) L_k(X_2) \mathbf{z}_i[(j-1)m + k] \,,$$

$$z_{i,j}(X_2) = \sum_{k\in[m]} L_k(X_2) \mathbf{z}_i[(j-1)m + k] \,,$$

then the consistency between them can be shown by the following check:

$$z_i(X_1, X_2) = q_{i,j}(X_1, X_2)(X_1 - \omega^j) + z_{i,j}(X_2) \,.$$

Here, we use two variables X_1, X_2 to avoid super-linear key size increment [39]. Now, we can finally apply a variant of Dynamix with bi-variate polynomials $z_i(X_1, X_2)$ (and we also need to modify all other polynomials to fit into this bi-variate setting) to $\mathbf{z}_4, \mathbf{z}_5, \mathbf{z}_6$ to achieve constant update time on copy constraints, while maintaining $O(\sqrt{n})$ update time on gate constraints with small constants. After this optimization, the new proof size will be $4\sqrt{n} + O(1)$ group elements, which is over $20\times$ smaller than original Dynaverse. Note this optimization will not improve the asymptotic complexities for Dynaverse (except that the size of vk will be improved from $O(\sqrt{n} + n_0)$ to $O(n_0)$) because the update algorithm and proof size are still lower-bounded by $\widetilde{O}(\sqrt{n})$ for gate constraints.

Asymptotically reducing the Dynaverse proof size. Bünz et al. [9] introduced IPA proofs for pairings, a way to delegate a pairing equation with n terms and have it proved with a proof of $\log n$ size—see Appendix D. By using this technique, the Dynaverse proof size and verification time can be reduced to $O(\log n)$. To do that, recall that a Dynaverse verifier must compute the following equation for every $i \in [6m]$: $e([p_i], g) = e([v_i], g) \cdot e([\beta_i], [X - \omega])$. If $\mathcal{V}$ picks $r \xleftarrow{\$} \mathbb{F}$, then it only needs to check the following combination:

$$\prod_{i\in[6m]} e([p_i], g^{r^{i-1}}) = \prod_{i\in[6m]} e([v_i], g^{r^{i-1}}) \cdot \prod_{i\in[6m]} e([\beta_i], [X - \omega]^{r^{i-1}}) \,, \tag{21}$$

Next, $\mathcal{P}$ can compute three products E_1, E_2, E_3 in Eq. (21) and provide proofs $\pi_{\mathsf{IPA},1}, \pi_{\mathsf{IPA},2}, \pi_{\mathsf{IPA},3}$ for each product so that $\mathcal{V}$ just needs to check the IPA proofs and whether $E_1 = E_2 \cdot E_3$. Such methods can be applied to all $O(\sqrt{n})$ similar equations that $\mathcal{V}$ needs to verify. Therefore, finally we can get a variant of Dynaverse with $O(\log n)$ proof size and verification time. Alternatively, we can consider using Groth16 [21] to directly compress the $O(\sqrt{n})$-size proof to constant size.

Lemma 7 (Succinct Dynaverse). *There exist variants of* Dynaverse *whose proof size and verification time are $O(\log n)$ using [9], or $O(1)$ using Groth16 [21]. All other complexities are the same.*

6 Dynalog: Dynamic zk-SNARK with polylog update time

In this section, we introduce a new dynamic zk-SNARK with polylog update time and proof size. To simplify the presentation, we still use the Plonkish arithmetization (Section 2) but with only multiplication gates (it is trivial to extend it to general circuits with addition gates). More specifically, for any $\mathbb{i}_{\mathcal{C}} = [n, n_0, \sigma]$ describing a circuit $\mathcal{C}$, public input $\mathbb{x} \in \mathbb{F}^{n_0}$, and witness $\mathbb{w} \in \mathbb{F}^{3n}$, let $\mathbf{z} = [\mathbb{w}; \mathbb{x}] = [\mathbf{z}_L; \mathbf{z}_R; \mathbf{z}_O; \mathbb{x}] \in \mathbb{F}^{3n+n_0}$. We have $(\mathbb{x}, \mathbb{w}) \in \mathbb{i}_{\mathcal{C}}$ if and only if $(\emptyset, \mathbf{z}) \in \mathbb{i}_{\mathcal{P}} = [3n+n_0, \sigma]$ and $\forall i \in [n]$, $\mathbf{z}_L[i] \cdot \mathbf{z}_R[i] = \mathbf{z}_O[i]$.

Recall that the main difficulty we face in previous sections to construct dynamic zk-SNARKs is gate constraints. Now we start to make use of a hierarchical data structure to enable efficient updates for multiplication gates. This data structure has also been adopted in various applications such as oblivious RAMs [29,28] and locally updatable codes [14].

Hierarchical data structure. We maintain $\log n+1$ buffers $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(\log n)}$ where each buffer $\mathsf{buf}^{(i)}$ has 2^i slots. The value for each slot is either empty or of form $(i, \delta_L, \delta_R, \delta_O)$, representing the *difference value* to update i-th gate. For any update to some value(s) in $\mathbf{z}$, we will regard it as updates in terms of some gate(s). 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 say the update to gate j is $(j, a, 0, b)$. The data structure works as follows.

- In the beginning, all buffers except the last one $\mathsf{buf}^{(\log n)}$ are full of empty slots, while $\mathsf{buf}^{(\log n)}$ is filled with the initial $\mathbf{z}$, i.e., $(i, \mathbf{z}_L[i], \mathbf{z}_R[i], \mathbf{z}_O[i])$, $\forall i \in [n]$.
- Whenever there is an update to some gate, start from $\mathsf{buf}^{(0)}$ and look for the first empty buffer, say it is $\mathsf{buf}^{(k)}$. Collect all the non-empty slots from $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(k-1)}$ together with the new update, put all of them into buf_k, and then empty $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(k-1)}$. 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 $\mathsf{buf}^{(k)}$. Denote $S^{(i)}$ as the set of all gate indices that appear in $\mathsf{buf}^{(i)}$.
- When all buffers are full, combine all the slots and get the latest $\mathbf{z}$. Empty $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(\log n-1)}$ and fill $\mathsf{buf}^{(\log n)}$ with values from the latest $\mathbf{z}$.

Based on this data structure, to show for gate constraints it is not sufficient to examine $\mathsf{buf}^{(i)}$ itself, since it contains only the *difference value* for each gate. What we should check is, for each gate in $\mathsf{buf}^{(i)}$, at the moment we apply the update, its "real" wires (inputs and output) satisfy the multiplication relation. To achieve this, we have the following key observation:

> For any two non-empty buffers $\mathsf{buf}^{(i)}, \mathsf{buf}^{(j)}$ where $i < j$, all the updates in $\mathsf{buf}^{(i)}$ come **after** those in $\mathsf{buf}^{(j)}$.

This observation allows us to check only $\mathsf{buf}^{(i)}, \ldots, \mathsf{buf}^{(\log n)}$ to obtain the "real" wires for gates in $\mathsf{buf}^{(i)}$.

Dynamic proving. Now we can explain in a high level how the proof system works based on the data structure above.

– *Copy constraints.* We directly use a variant of Dynamix, DynamixHyper, and maintain a log-size π_{perm}, which can be updated in $O(\log n)$ time per update.
– *Gate constraints.* Whenever there is a new update, we compute a non-empty buffer $\mathsf{buf}^{(i)}$ and empty $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(i-1)}$. Moreover, we attach a new proof $\pi^{(i)}_{\mathsf{gate}}$ showing that all the gates from $\mathsf{buf}^{(i)}$ still satisfy the multiplication relation.

The details of $\pi^{(i)}_{\mathsf{gate}}$ and π_{perm} are illustrated in the following subsections.

6.1 Relations for gate constraints

To show that all the gates from $\mathsf{buf}^{(i)}$ satisfy the multiplication relation, first we need a natural way to combine the buffers and get the "real" values on the wires. We choose homomorphic vector commitments based on KZG to achieve this because it is easy to tweak it to fit DynamixHyper later. More specifically, from each buffer $\mathsf{buf}^{(i)}$ we construct three size-n vectors $\boldsymbol{\delta}^{(i)}_L, \boldsymbol{\delta}^{(i)}_R, \boldsymbol{\delta}^{(i)}_O$ such that for every update $(j, a, b, c) \in \mathsf{buf}^{(i)}$, we have

$$\boldsymbol{\delta}^{(i)}_L[j] = a,\ \boldsymbol{\delta}^{(i)}_R[j] = b,\ \boldsymbol{\delta}^{(i)}_O[j] = c\,.$$

For all other locations in $\boldsymbol{\delta}^{(i)}_L, \boldsymbol{\delta}^{(i)}_R, \boldsymbol{\delta}^{(i)}_O$, we just put 0 there. Committing to these three vectors, we get commitments $C^{(i)}_L, C^{(i)}_R, C^{(i)}_O$. Due to homomorphism, the following three commitments are in fact the commitments to the latest $\mathbf{z}_L, \mathbf{z}_R, \mathbf{z}_O$ combining $\mathsf{buf}^{(i)}, \ldots, \mathsf{buf}^{(\log n)}$:

$$C_L = \prod_{j=i}^{\log n} C^{(j)}_L,\ C_R = \prod_{j=i}^{\log n} C^{(j)}_R,\ C_O = \prod_{j=i}^{\log n} C^{(j)}_O\,. \tag{22}$$

Based on $\{(C^{(j)}_L, C^{(j)}_R, C^{(j)}_O)\}_{j\in[i,\log n]}$, now we can show the gate consistency for $\mathsf{buf}^{(i)}$ in the following steps:

1. There is a set $S^{(i)}$ including the indices of all the updated gates from $\mathsf{buf}^{(i)}$. More formally, the vectors $C^{(i)}_L, C^{(i)}_R, C^{(i)}_O$ commit to are all 0 outside $S^{(i)}$.
2. Compute C_L, C_R, C_O as from Eq. (22). Open C_L, C_R, C_O at $S^{(i)}$ and all the gates satisfy multiplication relation.

Both steps require obtaining some opening proofs efficiently. In this case, tree-structure KZG-based vector commitments such as AMT [37] and Hyperproofs [34] are helpful. Here we pick Hyperproofs in our protocol (see the interface in Appendix G). From now on, we will use indices of form $\mathbf{i} = (i_1, \ldots, i_{\log n})$ as in Hyperproofs. Values in $\mathbf{z}_L, \mathbf{z}_R, \mathbf{z}_O$ can be accessed through subscriptions (e.g., $z_{L,\mathbf{i}}$) or evaluations (e.g., $z_R(\mathbf{i})$).

For Step 1 and each one of C_L, C_R, C_O, we build a tree based on $S^{(i)}$ and show that all the subtrees outside $S^{(i)}$ are empty. See Fig. 4 for an example. Consider the MLE polynomials we defined in Appendix G, what we need to do is to show that up to $O(|S^{(i)}|\log n)$ MLEs of the form $z_{i_1\ldots i_k}$ are identical to 0. Based on the final trick in Appendix G, for each $z_{i_1\ldots i_k}$ we show that Eq. (28) holds and in fact all required proofs are included in the proofs for the locations in $S^{(i)}$.

For Step 2, we enumerate all the gates in $S^{(i)}$ and open C_L, C_R, C_O at each gate to verify. Note that we maintain a MLT for each commitment in $\{(C^{(j)}_L, C^{(j)}_R, C^{(j)}_O)\}_{j\in[i,\log n]}$ and Hyperproofs are also homomorphic, so the proofs for C_L, C_R, C_O can be computed from the nodes in those MLTs.

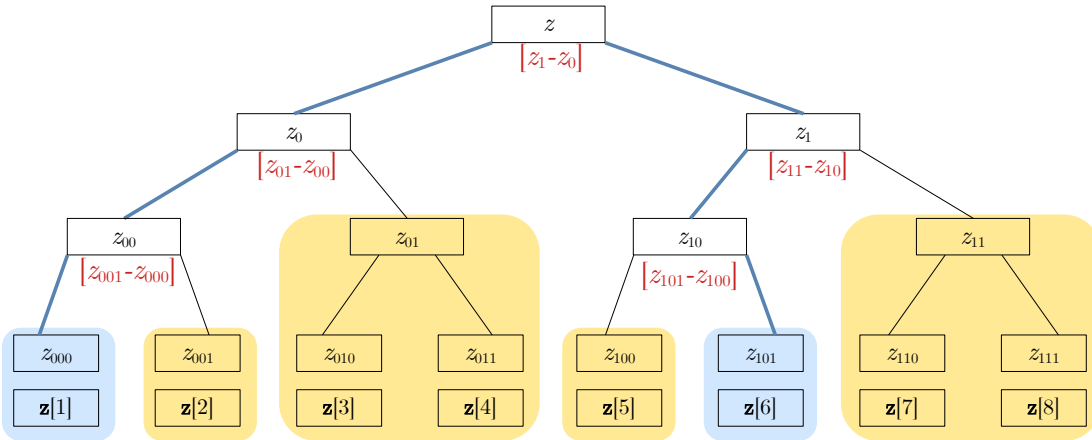

Fig. 4. A tree of MLEs with size 8 (red part is the proofs from MLT). When $S^{(i)} = \{1, 6\}$ (blue part), we only need to show that four sub-trees $z_{001}, z_{01}, z_{100}, z_{11}$ are all identical to 0 (yellow part). All the proofs for yellow part have already been included in the proofs for blue part, e.g., the proof for "$z_{01} = 0$" is $\{[z_1 - z_0], [z_{01} - z_{00}]\}$ from the proof for z_{000}.

We define a circuit $\mathcal{C}^{(i)}_{\mathsf{gate}}$ (see Fig. 5) to describe all the computation above. This circuit has size $O(|S^{(i)}| \log n)$ in terms of pairings/group operations. We will black-box use another proof system, such as Groth16 [21] to prove it and get the proof $\pi^{(i)}_{\mathsf{gate}}$.

6.2 DynamixHyper: A variant of Dynamix for copy constraints

Similar to our starting point in Section 4, given a permutation σ of size $m = 2^l$ and a vector $\mathbf{z}$ of size $m = 2^l$, to show $(\emptyset, \mathbf{z}) \in \mathbb{i}_{\mathcal{P}} = [m, \sigma]$, i.e., $\mathbf{z}[i] = \mathbf{z}[\sigma[i]]$ for all $i \in [m]$, we would like to show the following $s(Y)$ is the zero polynomial:

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

However, we cannot directly use Dynamo from Section 4 since we need to make sure our commitment to $\mathbf{z}$ is consistent with what we did in Section 6.1. With Hyperproofs, now the commitment to $\mathbf{z}$ is of form $[z]$ and z is a MLE to $\mathbf{z}$:

$$z(X_1, \ldots, X_l) = \sum_{\mathbf{i} \in \{0,1\}^l} z_{\mathbf{i}} \cdot \mathcal{S}_{\mathbf{i}}(X_1, \ldots, X_l) .$$

For $\mathbf{i} \in \{0,1\}^l$, define $\mathsf{num}(\mathbf{i}) = 1 + \sum_{j=1}^{l} i_j \cdot 2^{l-j}$ such that $\overline{i_1 \ldots i_l}$ is the binary form of $\mathsf{num}(\mathbf{i}) - 1$. Consider the new permutation polynomial

$$s(Y) = \sum_{\mathbf{i} \in \{0,1\}^l} z_{\mathbf{i}} \cdot \left(Y^{\mathsf{num}(\mathbf{i})} - Y^{\sigma^{-1}[\mathsf{num}(\mathbf{i})]} \right) = \sum_{\mathbf{i} \in \{0,1\}^l} z_{\mathbf{i}} \cdot \mathbf{y}_{\mathbf{i}} ,$$

- Public Input: $C_{\Delta}^{(i)}, C_{\Delta}^{(i+1)}, \ldots, C_{\Delta}^{(\log n)}$ for $\Delta \in \{L, R, O\}$
- Witness: $S^{(i)}$, $\{\pi_{\Delta,\mathbf{i}}^{(i)}\}_{\mathbf{i}\in S^{(i)}}$, $\{\pi_{\Delta,\mathbf{i}}\}_{\mathbf{i}\in S^{(i)}}$, $\{z_{\Delta,\mathbf{i}}\}_{\mathbf{i}\in S^{(i)}}$ for $\Delta \in \{L, R, O\}$
- Computation:
  1. For each $\Delta \in \{L, R, O\}$, check that opening $C_{\Delta}^{(i)}$ outside $S^{(i)}$ gets all 0s:
     For all $\mathbf{i} \in S^{(i)}$,
     (a) Parse $\pi_{\Delta,\mathbf{i}}^{(i)} = (\pi_{\Delta,\mathbf{i},1}^{(i)}, \ldots, \pi_{\Delta,\mathbf{i},\log n}^{(i)})$.
     (b) For all $k \in [\log n]$, if there is no $\mathbf{i}' \in S^{(i)}$ such that $\mathbf{i}'$ starts with $(i_1, \ldots, i_{k-1}, 1-i_k)$, then check
     $$e(C_{\Delta}^{(i)}, g) = e(\pi_{\Delta,\mathbf{i},k}^{(i)}, [X_k - (1-i_k)]) \prod_{j=1}^{k-1} e(\pi_{\Delta,\mathbf{i},j}^{(i)}, [X_j - i_j])$$
  2. For $\Delta \in \{L, R, O\}$, let $C_{\Delta} = \prod_{j=i}^{\log n} C_{\Delta}^{(j)}$, check opening C_{Δ} at $S^{(i)}$ gets $\{z_{\Delta,\mathbf{i}}\}_{\mathbf{i}\in S^{(i)}}$:
     For all $\mathbf{i} \in S^{(i)}$,
     (a) Parse $\pi_{\Delta,\mathbf{i}} = (\pi_{\Delta,\mathbf{i},1}, \ldots, \pi_{\Delta,\mathbf{i},\log n})$.
     (b) Check that
     $$e(C_{\Delta}/g^{z_{\Delta,\mathbf{i}}}, g) = \prod_{j=1}^{\log n} e(\pi_{\Delta,\mathbf{i},j}, [X_j - i_j])$$
  3. Check that
     $$z_{L,\mathbf{i}} \cdot z_{R,\mathbf{i}} = z_{O,\mathbf{i}}, \ \forall \mathbf{i} \in S^{(i)}.$$
  4. Return true.

Fig. 5. $\mathcal{C}_{\mathsf{gate}}^{(i)}$: circuit to check gate consistency for buffer $\mathsf{buf}^{(i)}$.

where, for ease of notation, we write

$$\mathbf{y}_{\mathbf{i}} = Y^{\mathsf{num}(\mathbf{i})} - Y^{\sigma^{-1}[\mathsf{num}(\mathbf{i})]}.$$

Now we can follow the idea as before: we have a prover commit to vector $\mathbf{z}$, i.e., $[z]$, and provide a proof that, for given σ, $s(Y)$ is the zero polynomial.

Computing the proof. The prover commits to $z(X_1, \ldots, X_l)$ and $v(X_1, \ldots, X_l, Y)$, where z encodes the vector $\mathbf{z}$ and v encodes $z_{\mathbf{i}} \cdot \mathbf{y}_{\mathbf{i}}$ ($v(\mathbf{i}, Y) = z_{\mathbf{i}} \cdot \mathbf{y}_{\mathbf{i}}$ for all $\mathbf{i} \in \{0,1\}^l$), i.e.,

$$v(X_1, \ldots, X_l, Y) = \sum_{\mathbf{i}\in\{0,1\}^l} z_{\mathbf{i}} \cdot \mathbf{y}_{\mathbf{i}} \cdot \mathcal{S}_{\mathbf{i}}(X_1, \ldots, X_l).$$

The verifier holds an honestly-computed commitment to $u(X_1, \ldots, X_l, Y)$ that encodes σ, i.e.,

$$u(X_1, \ldots, X_l, Y) = \sum_{\mathbf{i}\in\{0,1\}^l} \mathbf{y}_{\mathbf{i}} \cdot \mathcal{S}_{\mathbf{i}}(X_1, \ldots, X_l).$$

Now to prove that $s(Y) = 0$ the prover must also provide the following two additional proofs.

Zero-check. First the prover shows that for all $\mathbf{i} \in \{0,1\}^l$, it is $v(\mathbf{i}, Y) = u(\mathbf{i}, Y) \cdot z(\mathbf{i})$. This can also be done by computing $\alpha_i(X_1, \ldots, X_l, Y)$ for $i \in [l]$ to show that

$$v - u \cdot z = \sum_{i \in [l]} \alpha_i \cdot X_i(1 - X_i)\,.$$

Sum-check. Finally, the prover has to prove that

$$\sum_{\mathbf{i} \in \{0,1\}^l} v(\mathbf{i}, Y) = 0\,.$$

The idea from Section 4 using a "prefix polynomial" does not work directly here since we are evaluating v on the hypercube now. Inspired by the celebrated sum-check protocol [25], we define the following intermediate polynomials instead:

$$p_i(X_1, \ldots, X_i, Y) := \sum_{\mathbf{i}' \in \{0,1\}^i} \mathcal{S}_{\mathbf{i}'}(X_1, \ldots, X_i) \cdot \left(\sum_{\mathbf{i}'' \in \{0,1\}^{l-i}} v(\mathbf{i}', \mathbf{i}'', Y) \right), \text{ for all } i \in [l-1]\,. \tag{23}$$

Now it is enough to have the prover prove that

$$\begin{aligned}
&\text{(i) } p_{l-1}(X_1, \ldots, X_{l-1}, Y) = v(X_1, \ldots, X_{l-1}, 0, Y) + v(X_1, \ldots, X_{l-1}, 1, Y) \\
&\text{(ii) } p_{i-1}(X_1, \ldots, X_{i-1}, Y) = p_i(X_1, \ldots, X_{i-1}, 0, Y) + p_i(X_1, \ldots, X_{i-1}, 1, Y),\ \forall i \in [2, l-1] \\
&\text{(iii) } p_1(0, Y) + p_1(1, Y) = 0\,.
\end{aligned} \tag{24}$$

To show these relations, we additionally define the following polynomials for $j \in \{0,1\}$:

$$\begin{aligned}
v_j(X_1, \ldots, X_{l-1}, Y) &:= v(X_1, \ldots, X_{l-1}, j, Y) \\
p_{i,j}(X_1, \ldots, X_{i-1}, Y) &:= p_i(X_1, \ldots, X_{i-1}, j, Y),\ \forall i \in [l-1]
\end{aligned} \tag{25}$$

and the prover computes as proofs the commitments to the following quotient polynomials:

$$\begin{aligned}
\beta_j(X_1, \ldots, X_{l-1}, Y) &= \frac{v(X_1, \ldots, X_l, Y) - v_j(X_1, \ldots, X_{l-1}, Y)}{X_l - j} \\
\gamma_{i,j} &= \frac{p_i(X_1, \ldots, X_i, Y) - p_{i,j}(X_1, \ldots, X_{i-1}, Y)}{X_i - j},\ \forall i \in [l-1]\,.
\end{aligned} \tag{26}$$

Variable check. The prover must also do variable checks for all the polynomials above. We just ignore this part for simplicity in this section since the technique is exact the same as what we did in Section 4, but the variable check is still required in the final protocol and its security proofs.

Generalization to Dynamix(Hyper). Note that in the beginning of this section, we let $\mathbf{z} = [\mathbf{z}_L; \mathbf{z}_R; \mathbf{z}_O; \mathbb{x}] \in \mathbb{F}^{3n+n_0}$ where $\mathbf{z}_L, \mathbf{z}_R, \mathbf{z}_O \in \mathbb{F}^n$. Similar to the way we use Dynamix in Section 4, we need to tweak the proof system above to show that for each $\Delta \in \{L, R, O\}$, $(\emptyset, (\mathbf{z}_\Delta, h_\Delta)) \in \mathbb{i}_D = [n, N, \mathbf{s}, \mathbf{t}]$ (where $\mathbf{s}, \mathbf{t} \in [N]^n$), i.e., when letting $k = 0$ if $\Delta = L$, $k = 1$ if $\Delta = R$, $k = 2$ if $\Delta = O$,

$$\sum_{\mathbf{i} \in \{0,1\}^l} z_{\Delta,\mathbf{i}} \cdot (Y^{s_\mathbf{i}} - Y^{t_\mathbf{i}}) = h_\Delta(Y) \text{ where } s_\mathbf{i} = \mathsf{num}(\mathbf{i}) + kn,\ t_\mathbf{i} = \sigma^{-1}[s_\mathbf{i}]\,. \tag{27}$$

The only two extra things we need to do here is (i) The prover must also commit to h_Δ and the last relation in Eq. (24) should be changed to $p_1(0,Y)+p_1(1,Y)=h_\Delta(Y)$. (ii) $\mathbf{y_i}$ and u should be modified to be consistent with $Y^{s_\mathbf{i}}-Y^{t_\mathbf{i}}$.

Summary. The final proof for $(\emptyset,(\mathbf{z},h))\in \mathbb{i}_D=[n,N,\mathbf{s},\mathbf{t}]$ consists of $6l+2$ group elements that are KZG commitments of $\{h,z,\{\alpha_i\}_{i\in[l]},v,v_0,v_1,\beta_0,\beta_1,\{p_i,p_{i,0},p_{i,1},\gamma_{i,0},\gamma_{i,1}\}_{i\in[l-1]}\}$ plus necessary variable checks.

Computing and updating the proof. The following theorem is adapted from Theorem 1 to enable efficient proof computing and updating. Its proof uses the similar approach as for Theorem 1 so we omit it here.

Theorem 5. *Let $\mathcal{F}=\{h,z,\{\alpha_i\}_{i\in[l]},v,v_0,v_1,\beta_0,\beta_1,\{p_i,p_{i,0},p_{i,1},\gamma_{i,0},\gamma_{i,1}\}_{i\in[l-1]}\}$ be the set of polynomials defined as above. Every polynomial $f\in\mathcal{F}$ from can be expressed as*

$$f=\sum_{\mathbf{i}\in\{0,1\}^l} f_\mathbf{i}\cdot z_\mathbf{i}\,,$$

where $\{f_\mathbf{i}\}_{\mathbf{i}\in\{0,1\}^l}$ is a fixed set of 2^l polynomials.

We provide the detailed protocol of DynamixHyper in Fig. 6 (variable checks omitted) and summarize our protocol in the following theorem (its proof is similar to Theorems 2 and 3 and thus omitted).

Theorem 6 (DynamixHyper). *The protocol of Figure 16 is a dynamic SNARK (per Definition 2) for $\mathbb{i}_\mathcal{D}=[n,N,\mathbf{s},\mathbf{t}]$ where $N=O(n)$ assuming q-DLOG (see Assumption 1) in the AGM model. Its complexities are as follows.*

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

6.3 Put everything together

We now wrap up all the previous sub-sections and show our complete protocol in Figs. 7 and 8 (to simplify the presentation, we only consider the circuit with multiplication gates and ignore the variable checks in DynamixHyper).

Theorem 7 (Dynalog). *The protocol of Figure 8 is a dynamic SNARK (per Definition 2) for $\mathbb{i}_\mathcal{C}=[n,n_0,\sigma]$ assuming q-DLOG (see Assumption 1) in the AGM model if $\mathcal{S}$ is a secure SNARK protocol. Moreover, if we instantiate $\mathcal{S}$ with Groth16 [21], then the only assumption we need is q-DLOG in the AGM model. Its complexities are as follows (with Groth16).*

1. *$\mathcal{G}$ runs in $O(n\log n)$ time, outputs* pk *of $O(n\log n)$ size,* vk *of $O(\log n)$ size;*
2. *$\mathcal{P}$ runs in $O(n\log^2 n)$ time and outputs a proof π of $O(\log n)$ 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 $O(n_0+\log n)$ time.*

– $\mathcal{G}(1^\lambda, [n, N, \mathbf{s}, \mathbf{t}]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - Set $l = \log n$.
  - $\mathsf{pp_{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$.
  - Let $\mathcal{F} = \{h, z, \{\alpha_i\}_{i\in[l]}, v, v_0, v_1, \beta_0, \beta_1, \{p_i, p_{i,0}, p_{i,1}, \gamma_{i,0}, \gamma_{i,1}\}_{i\in[l-1]}\}$ be the set of polynomials from Theorem 5.
  - Pick random $a_1, \ldots, a_l$, b from $\mathbb{F}$ for variables $X_1, \ldots, X_l$ and Y respectively.
  - Set $\mathsf{pk} = \mathsf{upk}$ to contain the following KZG commitments, defined in Theorem 5, and computed using $a_1, \ldots, a_l$ and b directly

$$\{[f_{\mathbf{i}}]\}_{f\in\mathcal{F}, \mathbf{i}\in\{0,1\}^l} \cdot$$

  - Set

$$\mathsf{vk} = \{[u], \{[X_i], [X_i(1 - X_i)]\}_{i\in[l]}\},$$

  where u is $u(X_1, \ldots, X_l, Y) = \sum_{\mathbf{i}\in\{0,1\}^l} \mathcal{S}_{\mathbf{i}}(X_1, \ldots, X_l) \cdot (Y^{s_{\mathbf{i}}} - Y^{t_{\mathbf{i}}})$.

– $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{aux})$:
  - Parse $\mathbb{x}$ as $\emptyset$ and $\mathbb{w}$ as $\{z_{\mathbf{i}}\}_{\mathbf{i}\in\{0,1\}^l}$.
  - Output $|\mathcal{F}| = 6l + 2$ KZG commitments as π and aux, i.e., for all $f \in \mathcal{F}$ output

$$[f] = \prod_{\mathbf{i}\in\{0,1\}^l} [f_{\mathbf{i}}]^{z_{\mathbf{i}}}.$$

– $\mathcal{U}(\mathsf{upk}, \mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}, \pi, \mathsf{aux}) \to (\pi', \mathsf{aux}')$:
  - Parse $\mathbb{w}$ as $\mathbf{z}$ and $\mathbb{w}'$ as a new valid witness $\mathbf{z}'$. Parse π and aux as $\{[f]\}_{f\in\mathcal{F}}$.
  - Let J be the set of locations that $\mathbf{z}$ and $\mathbf{z}'$ differ and let $\{\delta_{\mathbf{j}}\}_{\mathbf{j}\in J}$ be the corresponding deltas. Output as π' and aux' the new KZG commitments $\{[f']\}_{f\in\mathcal{F}}$ where

$$[f'] = [f] \cdot \prod_{\mathbf{j}\in J} [f_{\mathbf{j}}]^{\delta_{\mathbf{j}}}.$$

– $\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$:
  - Parse vk and π as output by $\mathcal{G}$ and $\mathcal{P}$ respectively.
  - Output 1 if and only if all the following relations hold:

$e([v], g) \cdot e([-u], [z]) = \prod_{i\in[l]} e([\alpha_i], [X_i(1 - X_i)])$.
$[p_{l-1}] = [v_0] \cdot [v_1]$.
$[p_{i-1}] = [p_{i,0}] \cdot [p_{i,1}]$ for all $i \in [2, l - 1]$.
$[p_{1,0}] \cdot [p_{1,1}] = [h]$.
$e([v], g) \cdot e([-v_j], g) = e([\beta_j], [X_l - j])$ for $j \in \{0, 1\}$.
$e([p_i], g) \cdot e([-p_{i,j}], g) = e([\gamma_{i,j}], [X_i - j])$ for $i \in [l - 1], j \in \{0, 1\}$.

Fig. 6. DynamixHyper.

Sub-routine $\mathsf{NewBuffer}^{(i)}$

Input: $\mathsf{pk}^{(i)}_{\mathsf{gate}}, \mathsf{pp}_{\mathsf{VC}}, i, S^{(i)}, \boldsymbol{\delta}^{(i)}_L, \boldsymbol{\delta}^{(i)}_R, \boldsymbol{\delta}^{(i)}_O, (C^{(j)}_\Delta, \pi^{(j)}_\Delta, z^{(j)}_\Delta)_{j\in[i+1,l],\Delta\in\{L,R,O\}}$

Output: $\pi^{(i)}_{\mathsf{gate}}, (C^{(i)}_\Delta, \pi^{(i)}_\Delta)_{\Delta\in\{L,R,O\}}$

Computation:

1. For $\Delta \in \{L, R, O\}$, compute $C^{(i)}_\Delta \leftarrow \mathsf{VC.Com}(\mathsf{pp}_{\mathsf{VC}}, \boldsymbol{\delta}^{(i)}_\Delta)$, $\pi^{(i)}_\Delta \leftarrow \mathsf{VC.OpenAll}(\mathsf{pp}_{\mathsf{VC}}, \boldsymbol{\delta}^{(i)}_\Delta)$.
2. Let $\mathbb{x} = (C^{(j)}_\Delta)_{j\in[i,l],\Delta\in\{L,R,O\}}$ and $\mathbb{w} = \{S^{(i)}\}$.
3. For each $\mathbf{i} \in S^{(i)}$ and $\Delta \in \{L, R, O\}$, extract $\pi^{(i)}_{\Delta,\mathbf{i}}$ from $\pi^{(i)}_\Delta$ and put it into $\mathbb{w}$.
4. For each $\mathbf{i} \in S^{(i)}$ and $\Delta \in \{L, R, O\}$, compute the following and put them into $\mathbb{w}$:
   - $z_{\Delta,\mathbf{i}} = \sum_{j=i}^{l} z^{(j)}_{\Delta,\mathbf{i}}$;
   - $\pi_{\Delta,\mathbf{i}} = \prod_{j=i}^{l} \pi^{(j)}_{\Delta,\mathbf{i}}$ where $\pi^{(j)}_{\Delta,\mathbf{i}}$ is extracted from $\pi^{(j)}_\Delta$.
5. Compute $\pi^{(i)}_{\mathsf{gate}} \leftarrow \mathcal{S}.\mathcal{P}(\mathsf{pk}^{(i)}_{\mathsf{gate}}, \mathbb{x}, \mathbb{w})$.
6. Output $\pi^{(i)}_{\mathsf{gate}}, (C^{(i)}_\Delta, \pi^{(i)}_\Delta)_{\Delta\in\{L,R,O\}}$.

Fig. 7. Sub-routine for $\mathsf{NewBuffer}^{(i)}$

Proof. Completeness and updatability follow naturally from the construction. For the extractor for knowledge soundness, we directly use the extractor for DynamixHyper to extract $\mathbf{z}_L, \mathbf{z}_R, \mathbf{z}_O$ from $\pi_{\mathsf{perm},L}, \pi_{\mathsf{perm},R}, \pi_{\mathsf{perm},O}$ and output $\mathbb{w} = [\mathbf{z}_L; \mathbf{z}_R; \mathbf{z}_O]$. Let $\mathbf{z} = [\mathbb{w}; \mathbb{x}]$. It is natural to see that such $\mathbf{z}$ satisfy the copy constraints from the security proof of DynamixHyper. Now we argue that $\mathbb{w} = [\mathbf{z}_L; \mathbf{z}_R; \mathbf{z}_O]$ also satisfies the gate constraints.

Parse $\pi = (\pi_{\mathsf{perm},L}, \pi_{\mathsf{perm},R}, \pi_{\mathsf{perm},O}, (C^{(j)}_L, C^{(j)}_R, C^{(j)}_O, \pi^{(j)}_{\mathsf{gate}})_{j\in[0,l]})$. Since the adversary is algebraic, we can reconstruct $\{z^{(j)}_\Delta\}_{j\in[0,l],\Delta\in\{L,R,O\}}$ such that $[z^{(j)}_\Delta] = C^{(j)}_\Delta$. Based on line 2 of $\mathcal{V}$, the extracted witness $[\mathbf{z}_L; \mathbf{z}_R; \mathbf{z}_O]$ has the following MLEs:

$$z_L = \sum_{j\in[0,l]} z^{(j)}_L,\ z_R = \sum_{j\in[0,l]} z^{(j)}_R,\ z_O = \sum_{j\in[0,l]} z^{(j)}_O\,.$$

Now we consider all the multiplication gates. For gate with index $\mathbf{i} \in \{0,1\}^l$, suppose it first appears in $S^{(i)}$, i.e., for all $j \in [0, i-1]$, $\mathbf{i} \notin S^{(j)}$. Then for any $j \in [0, i-1]$, either we have $\mathsf{buf}^{(j)}$ is empty ($z^{(j)}_\Delta = 0$), or we have a proof $\pi^{(j)}_{\mathsf{gate}}$ showing that as an index outside $S^{(j)}$, $C^{(j)}_\Delta$ opens to 0 at $\mathbf{i}$, i.e., $z^{(j)}_\Delta(\mathbf{i}) = 0$ for every $\Delta \in \{L, R, O\}$. Therefore we have

$$z_\Delta(\mathbf{i}) = \sum_{j\in[0,l]} z^{(j)}_\Delta(\mathbf{i}) = \sum_{j\in[i,l]} z^{(j)}_\Delta(\mathbf{i}),\ \Delta \in \{L, R, O\}\,.$$

– $\mathcal{G}(1^\lambda, [n, n_0, \sigma]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - Set $l = \log n$ and $N = 3n + n_0$.
  - Include the following in pp:

$$\mathsf{pp}_{\mathsf{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda),\ \mathsf{pp}_{\mathsf{VC}} \leftarrow \mathsf{VC.Gen}_{\mathsf{pp}_{\mathsf{bl}}}(1^\lambda, n),\ (\mathsf{pk}^{(i)}_{\mathsf{gate}}, \mathsf{vk}^{(i)}_{\mathsf{gate}}) \leftarrow \mathcal{S}.\mathcal{G}(1^\lambda, \mathcal{C}^{(i)}_{\mathsf{gate}}),\ \forall i \in [0, l]\,.$$

  - Pick random $a_1, \ldots, a_l$, b from $\mathbb{F}$ for variables $X_1, \ldots, X_l$ and Y respectively.
  - For $\Delta \in \{L, R, O\}$, call

$$\mathcal{D}.\mathcal{G}_{a_1,\ldots,a_l,b,\mathsf{pp}_{\mathsf{bl}}}(1^\lambda, [n, N, \mathbf{s}_\Delta, \mathbf{t}_\Delta]) \to (\mathsf{pk}_\Delta, \mathsf{upk}_\Delta, \mathsf{vk}_\Delta)\,,$$

    where $\mathbf{s}_\Delta$ and $\mathbf{t}_\Delta$ are defined in Eq. (27).
  - Set $\mathsf{pk} = \{\mathsf{pp}, \mathsf{pk}_L, \mathsf{pk}_R, \mathsf{pk}_O\}$, $\mathsf{upk} = \{\mathsf{pp}, \mathsf{upk}_L, \mathsf{upk}_R, \mathsf{upk}_O\}$ and $\mathsf{vk} = \{\mathsf{pp}, \mathsf{vk}_L, \mathsf{vk}_R, \mathsf{vk}_O\}$.

– $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{aux})$:
  - Parse $\mathbb{w}$ as $[\mathbf{z}_L, \mathbf{z}_R, \mathbf{z}_O]$.
  - For each $\Delta \in \{L, R, O\}$, call $\mathcal{D}.\mathcal{P}(\mathsf{pk}_\Delta, \emptyset, \mathbf{z}_\Delta) \to (\pi_{\mathsf{perm},\Delta}, \mathsf{aux}_\Delta)$.
  - Call $\mathsf{NewBuffer}^{(l)}(\mathsf{pk}^{(l)}_{\mathsf{gate}}, \mathsf{pp}_{\mathsf{VC}}, l, [n], \mathbf{z}_L, \mathbf{z}_R, \mathbf{z}_O) \to (\pi^{(l)}_{\mathsf{gate}}, (C^{(l)}_\Delta, \pi^{(l)}_\Delta)_{\Delta\in\{L,R,O\}})$.
  - Output $\pi = (\pi_{\mathsf{perm},L}, \pi_{\mathsf{perm},R}, \pi_{\mathsf{perm},O}, C^{(l)}_L, C^{(l)}_R, C^{(l)}_O, \pi^{(l)}_{\mathsf{gate}})$ and put all other data in aux.

– $\mathcal{U}(\mathsf{upk}, \mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}, \pi, \mathsf{aux}) \to (\pi', \mathsf{aux}')$:
  - Parse $\mathbb{w}$ as $[\mathbf{z}_L, \mathbf{z}_R, \mathbf{z}_O]$ and $\mathbb{w}'$ as $[\mathbf{z}'_L, \mathbf{z}'_R, \mathbf{z}'_O]$.
  - For each $\Delta \in \{L, R, O\}$ call $\mathcal{D}.\mathcal{U}(\mathsf{upk}_\Delta, \emptyset, \mathbf{z}'_\Delta, \emptyset, \mathbf{z}_\Delta, \pi_{\mathsf{perm},\Delta}, \mathsf{aux}_\Delta) \to (\pi'_{\mathsf{perm},\Delta}, \mathsf{aux}'_\Delta)$.
  - For each update to a gate (j, a, b, c):
    If there is some empty buffer,
    * Start from $\mathsf{buf}^{(0)}$ and find the first empty buffer $\mathsf{buf}^{(i)}$.
    * Put (j, a, b, c) and all non-empty slots from $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(i-1)}$ into $\mathsf{buf}^{(i)}$, and combine those slots for the same gate. Clear $\mathsf{buf}^{(k)}$ and all its data structure, $\forall k \in [0, i-1]$.
    * Derive $S^{(i)}, \boldsymbol{\delta}^{(i)}_L, \boldsymbol{\delta}^{(i)}_R, \boldsymbol{\delta}^{(i)}_O$ from $\mathsf{buf}^{(i)}$. Collect all the necessary inputs from aux, call $\mathsf{NewBuffer}^{(i)}$, and gets $\pi^{(i)}_{\mathsf{gate}}, (C^{(i)}_\Delta, \pi^{(i)}_\Delta)_{\Delta\in\{L,R,O\}}$.

    Otherwise (all buffers are full), call $\mathsf{NewBuffer}^{(l)}(\mathsf{pk}^{(l)}_{\mathsf{gate}}, \mathsf{pp}_{\mathsf{VC}}, l, [n], \mathbf{z}'_L, \mathbf{z}'_R, \mathbf{z}'_O) \to (\pi^{(l)}_{\mathsf{gate}}, (C^{(l)}_\Delta, \pi^{(l)}_\Delta)_{\Delta\in\{L,R,O\}})$, and clear $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(l-1)}$.
  - Output $\pi' = (\pi'_{\mathsf{perm},L}, \pi'_{\mathsf{perm},R}, \pi'_{\mathsf{perm},O}, (C^{(j)}_L, C^{(j)}_R, C^{(j)}_O, \pi^{(j)}_{\mathsf{gate}})_{j\in[0,l]})$ and update all other data in aux'.

– $\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$:
  - For each $i \in [0, l]$, pick $\mathbb{x}^{(i)}$ from $\{C^{(j)}_\Delta\}$ and check $\mathcal{S}.\mathcal{V}(\mathsf{vk}^{(i)}_{\mathsf{gate}}, \mathbb{x}^{(i)}, \pi^{(i)}_{\mathsf{gate}}) \to 1$.
  - For each $\Delta \in \{L, R, O\}$ extract $[z_\Delta]$ from $\pi_{\mathsf{perm},\Delta}$ and check that $[z_\Delta] = \prod_{j\in[l]} C^{(j)}_\Delta$.
  - For each $\Delta \in \{L, R, O\}$ check that $\mathcal{D}.\mathcal{V}(\mathsf{vk}_\Delta, \emptyset, \pi_{\mathsf{perm},\Delta}) \to 1$.
  - Parse $\mathbb{x}$ as $\mathbf{z}[3n+1 \ldots 3n+n_0]$. Set $h_{\mathbb{x}}(Y) = \sum_{i=6n+1}^{6n+n_0} \mathbf{z}[i](Y^i + Y^{\sigma^{-1}(i)})$. Check whether $[h_{\mathbb{x}}(Y)] \cdot [h_L(Y)] \cdot [h_R(Y)] \cdot [h_O(Y)] = 1_{\mathbb{G}}$, where $[h_\Delta(Y)]$ is extracted from $\pi_{\mathsf{perm},\Delta}$.

Fig. 8. Dynalog using DynamixHyper $\mathcal{D}$ (Fig. 6), a secure SNARK $\mathcal{S}$, and a vector commitment scheme VC (instantiated with Hyperproofs, see Appendix G and [34]).

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

$$\left(\sum_{j\in[i,l]} z^{(j)}_L(\mathbf{j})\right) \cdot \left(\sum_{j\in[i,l]} z^{(j)}_R(\mathbf{j})\right) = \sum_{j\in[i,l]} z^{(j)}_O(\mathbf{j}),$$

which is also true for $\mathbf{i} \in S^{(i)}$. Finally, we have

$$z_L(\mathbf{i}) \cdot z_R(\mathbf{i}) = z_O(\mathbf{i}),$$

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

Our security proof is based on the security proofs of DynamixHyper (q-DLOG (see Assumption 1) in the AGM), Hyperproofs (q-SDH, Assumption 2) and the SNARK used for $\mathcal{S}$. Since q-DLOG is harder than q-SDH (any algorithm that solves q-DLOG can easily solve q-SDH), and Fuchsbauer et al. [19] show that Groth16 [21] is secure under q-DLOG in the AGM, we only need q-DLOG in the AGM as the assumption for our protocol if $\mathcal{S}$ is instantiated with Groth16.

Each $\mathcal{C}^{(i)}_{\mathsf{gate}}$ has $O(2^i \cdot \log n)$ circuit size. When using Groth16, the prover time to compute $\pi^{(i)}_{\mathsf{gate}}$ is $O(2^i \cdot \log^2 n)$ and the proof size is $O(1)$. Therefore, in our final protocol, the proof size is $O(\log n)$ (combined with Dynamix proofs) and the amortized update time is $O(\log^3 n)$ (each $\mathsf{buf}^{(i)}$ will be recomputed after every $O(2^i)$ updates).

□

De-amortization for $O(\log^3 n)$ update time in the worse case. Finally, we would like to remark that the update algorithm in Fig. 8 can be de-amortized to achieve $O(\log^3 n)$ update time in the worse case. At a high level, this can be done through decomposition of the prove algorithm for $\mathcal{C}^{(i)}_{\mathsf{gate}}$ into 2^i steps. The data structures for buffers $\mathsf{buf}^{(0)}, \ldots, \mathsf{buf}^{(i-1)}$ will be kept until the computation for $\pi^{(i)}_{\mathsf{gate}}$ finishes.

Dynalog+: $O(1)$ proof size and verification time. We finally note that one can further decrease the proof size and verification time to $O(1)$, by using **one level of recursion** with Groth16 [21], therefore not compromising security further. We call this adjusted construction Dynalog+.

7 Recursion-free Incremental Verifiable Computation

We now present our recursion-free IVC scheme. Recall the main difference from other IVC schemes is that our construction must know the number of iterations N it can support. The definition of IVC adjusted to include N from [23] is given in the Appendix (see Definition 3). Our main idea is the following: We represent the IVC computation with an N-sized circuit F_N (see Figure 9) whose public input is (i, z_0, z_i)—the same with the public input of an 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 a logarithmic number of wires change. We will use a dynamic zk-SNARK DS, as in Definition 2, to build an IVC scheme I as in Definition 3. 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 the final IVC scheme are provided in Figure 10.

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., Muhash [5]). The hash then will have to be computed using a binary tree inside F_N.

Ensuring the transferred state is sublinear and zero-knowledge. Our IVC scheme appends to the dynamic SNARK proof the state aux that is required for the dynamic SNARK to perform the next update. We note that this state, 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.) In particular, recall that Dynaverse, for proving multiplication constraints, bucketizes the witness in $\sqrt{n}$-sized buckets. Because updates in IVC are monotonically increasing, only the bucket needs to be appended as state, which keeps the transferred state sublinear. In Appendix H, we also show how to keep the state zero-knowledge. We now have the following Theorem (See Appendix F for proofs.)

Theorem 8 (Recursion-free IVC from dynamic zk-SNARKs). *The protocol of Figure 10 is a recursion-free IVC scheme (per Definition 3) for N iterations of function F, assuming a dynamic zk-SNARK DS with algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{U}, \mathcal{V})$ (per Definition 2). 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 NP relation $\mathbb{i}$ defined by the $N \cdot |F|$-sized circuit F_N from Figure 1. It outputs pk of size $|\mathsf{DS.pk}| + |\mathsf{DS.upk}|$ and vk of size and $|\mathsf{DS.vk}|$;*
2. *$\mathcal{P}$ runs in $|\mathsf{DS}.\mathcal{U}|$ time and outputs a proof π of $|\mathsf{DS}.\pi| + |\mathsf{DS.aux}|$ size;*
3. *$\mathcal{V}$ runs in $|\mathsf{DS}.\mathcal{V}|$ time.*

We note for the above compiler to output a zero-knowledge IVC, the used dynamic zk-SNARK must be zero-knowledge even when it outputs its state aux as part of the proof, which is the case for Dynaverse. We now have the following.

Lemma 8 (Recursion-free IVC from Dynaverse). *There exists a recursion-free IVC scheme (per Definition 3) 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 $O(M \log M)$ and outputs $O(M)$-size pk and $O(\sqrt{M})$-size vk;*
2. *$\mathcal{P}$ runs in $O(\sqrt{M} \log M)$ time and outputs a proof π of $O(\sqrt{M})$ size;*
3. *$\mathcal{V}$ runs in $O(\sqrt{M})$ time.*

We note here that the IVC proof above can further be compressed to $O(\log M)$ if needed, using GIPA, as in Dynaverse.

8 Evaluation

In this section, we evaluate the performance of our constructions. Specifically, we compare the performance of Dynarec and Dynaverse against a PLONK-based [20] implementation that needs to recompute the proof from scratch whenever there is an update.

- 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 $z_2 = \mathsf{F}(b_2 \cdot z_1, b_2 \cdot w_1)$;
  - ...
  - 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. 9. 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.

- $\mathcal{G}(1^\lambda, \mathsf{F}, N) \to (\mathsf{pk}, \mathsf{vk})$.
  1. Let F_N be the N-sized circuit wrt to F from Figure 9;
  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. 10. Constructing an IVC scheme from a dynamic zk-SNARK.

Table 3. Comparison of one random update between Dynaverse, Dynarec, and PLONK-based baseline.

$L = \log_2 n$	PLONK (baseline)			Dynarec			Dynaverse		
	Update (s)	Verify (ms)	Proof (KiB)	Update (s)	Verify (ms)	Proof (KiB)	Update (s)	Verify (ms)	Proof (KiB)
18	1.24	≤ 5	0.64	3.82	≤ 6	129.8	0.07	95	96
20	4.92			4.53			0.08	125	192
22	19.38			4.63			0.12	179	384
24	80.12			5.09			0.16	293	768

The implementation details are as follows: (1) Our Dynaverse implementation[6] is written in Rust using the BLS12-318 elliptic curve. (2) Our Dynarec implementation is written in Rust using Plonky2 [36]. Plonky2 [36] is a recursive proof system based on the Goldilocks field, which is faster than the BLS12-318 elliptic curve used in Dynaverse. However, this difference in fields negatively presents Dynaverse's performance. Despite this disadvantage, we demonstrate that Dynaverse outperforms Dynarec and the baseline in this section. (3) Our baseline implementation for measuring the cost of recomputing proofs from scratch whenever the witness is updated is written in Golang and uses the PLONK proof system. This baseline implementation uses the state-of-the-art `gnark` library [8] and the BLS12-318 elliptic curve with KZG commitments [22].

Recall that Dynarec requires incremental multiset hashing. However, incremental multiset hashing requires the hash function output to be several thousand bits long [26]. To overcome this limitation, we adopt the elliptic curve-based incremental multiset hashing approach proposed by Maitin-Shepard et al. [26]. Specifically, we use the EcGFp5 curve [32], based on the $\mathrm{GF}(p^5)$ extension of the Goldilocks field, and apply the Poseidon hash within our circuits.

Hardware. We executed our experiments on an AWS EC2 c7i.48xlarge instance with Intel Xeon Scalable CPU with 3.2 GHz, 192 cores and 384 GB of RAM. All the experiments are parallelized and use as many threads as allowed by the multi-threading library.

Comparison with a baseline. In our experiments, we generate a random circuit of size n (see Plonkish arithmetization in Section 2), run $\mathcal{G}$ and $\mathcal{P}$ to produce the initial proof, and then modify a randomly selected witness location to update the proof. In our PLONK baseline, we ensure the number of gates matches that of our construction.

We present the prover and verifier time, and proof size in Table 3. We observe that the update time of Dynaverse is $17 - 488\times$ faster than the PLONK [20] baseline for circuit sizes between 2^{18} to 2^{24}. This is because asymptotically update time of Dynaverse is at least $\sqrt{n}$ times faster than recomputing from scratch using PLONK. However, Dynaverse verification time and proof size are more expensive but still reasonable—up to 0.293 seconds for verification and 768 KiB for proof size when $n = 2^{24}$.

Although, asymptotically, Dynarec is better than Dynaverse, concretely, the update time of Dynarec is $30\times$ to $55\times$ smaller. This is because of small constants and the inefficiency of SNARK recursion. However, Dynarec has comparable or better verification time and proof size when compared to Dynaverse. Regardless of its concrete performance, Dynarec is only heuristically secure.

[6] https://anonymous.4open.science/r/dsnark-CC7E

References

1. https://www.lagrange.dev/blog/lagrange-deploys-first-production-ready-zk-prover-network-powered-by-coinbase-kraken-and-okx
2. Lagrange prover network, https://app.lagrange.dev/zk-coprocessor/explore/dashboard
3. 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)
4. Baldimtsi, F., Chalkias, K.K., Ji, Y., Lindstrøm, J., Maram, D., Riva, B., Roy, A., Sedaghat, M., Wang, J.: zklogin: Privacy-preserving blockchain authentication with existing credentials. CCS (2024)
5. 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)
6. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. Algorithmica **79**(4), 1102–1160 (2017)
7. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for snarks and proof-carrying data. p. 111–120. New York, NY, USA (2013)
8. Botrel, G., Piellard, T., Housni, Y.E., Kubjas, I., Tabaie, A.: Consensys/gnark: v0.12.0 (Jan 2025). https://doi.org/10.5281/zenodo.5819104
9. 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)
10. 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)
11. 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)
12. 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)
13. 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)
14. 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)
15. 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)
16. 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)
17. 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)
18. Clarke, D.E., Devadas, S., van Dijk, M., Gassend, B., Suh, G.E.: Incremental multiset hash functions and their application to memory integrity checking. In: Laih, C. (ed.) Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings. vol. 2894, pp. 188–207 (2003)
19. 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)
20. 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)
21. 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)
22. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-Size Commitments to Polynomials and Their Applications. In: ASIACRYPT'10 (2010)

23. Kothapalli, A., Setty, S.T.V.: Hypernova: Recursive arguments for customizable constraint systems. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X. vol. 14929, pp. 345–379 (2024)
24. Kothapalli, A., Setty, S.T.V., Tzialla, I.: Nova: Recursive zero-knowledge arguments from folding schemes. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV. vol. 13510, pp. 359–388 (2022)
25. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM **39**(4), 859–868 (oct 1992). https://doi.org/10.1145/146585.146605, https://doi.org/10.1145/146585.146605
26. Maitin-Shepard, J., Tibouchi, M., Aranha, D.F.: Elliptic curve multiset hash. CoRR **abs/1601.06502** (2016), http://arxiv.org/abs/1601.06502
27. 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)
28. 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). https://doi.org/10.1145/100216.100289, https://doi.org/10.1145/100216.100289
29. 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). https://doi.org/10.1007/0-387-34805-0_55
30. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of Correct Computation. In: TCC'13 (2013)
31. 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)
32. Pornin, T.: EcGFp5: a specialized elliptic curve. Cryptology ePrint Archive, Paper 2022/274 (2022)
33. Sefranek, M.: How (not) to simulate PLONK. Cryptology ePrint Archive, Paper 2024/848 (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. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008 (2008)
39. Wang, W., Ulichney, A., Papamanthou, C.: BalanceProofs: Maintainable vector commitments with fast aggregation. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 4409–4426. Anaheim, CA (Aug 2023)
40. 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)
41. Zhang, J., Fang, Z., Zhang, Y., Song, D.: Zero knowledge proofs for decision tree predictions and accuracy. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020. pp. 2039–2053 (2020)
42. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vsql: Verifying arbitrary sql queries over dynamic outsourced databases. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 863–880 (2017)

A IVC Definition

Definition 3 (IVC). *An incremental verifiable computation scheme (IVC) scheme is a tuple of PPT algorithms* $\mathsf{I} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ *with the following interface:*

- $\mathcal{G}(1^\lambda, \mathsf{F}, N) \to (\mathsf{pk}, \mathsf{vk})$. *On input security parameter, the function* F *and the number of iterations* N*, it outputs a prover key* pk *and verification key* vk.

– $\mathcal{P}(\mathsf{pk}, i, z_0, \pi_{i-1}, z_{i-1}, w_{i-1}, z_i) \to \pi_i$. *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.*
– $\mathcal{V}(\mathsf{vk}, (i, z_i, z_0), \pi_i) \to 0/1$: *On input verification key* vk*, counter i, output z_i, initial input z_0 and proof π_i, outputs accept or reject.*

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

– Completeness*: Let N be polynomially-bounded and let $\mathcal{G}(1^\lambda, \mathsf{F}, N) \to (\mathsf{pk}, \mathsf{vk})$ 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 $\mathcal{P}(\mathsf{pk}, j, z_0, \pi_{j-1}, z_{j-1}, w_{j-1}, z_j) \to \pi_j$, then $\mathcal{V}(\mathsf{vk}, (i, z_j, z_0), \pi_j) \to 1$.
– 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*

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

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. $\mathcal{G}(1^\lambda, \mathsf{F}, N) \to (\mathsf{pk}, \mathsf{vk})$;
2. For $j = 1$ to i output $\mathcal{P}(j, z_0, \pi_{j-1}, z_{j-1}, w_{j-1}, z_j, \mathsf{pk}) \to \pi_j$.
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 IVC 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 Details of Dynarec

C Auxiliary lemmas

Lemma 9. *Suppose $f(X_1, \ldots, X_s) \in \mathbb{F}_{\leq d}[X_1, \ldots, X_s]$ 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:

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

- Public Input: h_i
- Witness: $(m_j)_{j\in[0,5]}$
- Computation:
  1. Check $m_0 + m_1 = m_2,\ m_3 \cdot m_4 = m_5$.
  2. Check $h_i = \prod_{j\in[0,5]} \frac{\mathsf{H}(m_j, jn+i)}{\mathsf{H}(m_j, \sigma(jn+i))}$. Note that the values of $\sigma(jn+i)$ are hard-coded into $\mathcal{C}_i$.
  3. Return true.

Fig. 11. Dynarec's leaf circuit $\mathcal{C}_i$.

- Public Input: h
- Witness: π_L, π_R, h_L, h_R
- Computation:
  1. Check $\mathsf{S}.\mathcal{V}(\mathsf{vk}_L, h_L, \pi_L)$ and $\mathsf{S}.\mathcal{V}(\mathsf{vk}_R, h_R, \pi_R)$. Note that specific vk_L and vk_R keys are hard-coded for each $\mathcal{D}$.
  2. if $\mathcal{D}$ is the root node, check $h \cdot h_L \cdot h_R = 1$. Else, check $h_L \cdot h_R = h$.
  3. Return true.

Fig. 12. Dynarec's internal circuit $\mathcal{D}$ with hard-coded $\mathsf{vk}_L, \mathsf{vk}_R$.

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

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

Finally, we have

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

– $\mathcal{G}(1^\lambda, [n, n_0, \sigma]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - For each leaf node $i = 1, \ldots, n$, run

$$(\mathsf{pk}_{\mathcal{C}_i}, \mathsf{vk}_{\mathcal{C}_i}) \leftarrow \mathsf{S}.\mathcal{G}(1^\lambda, \mathcal{C}_i)\,.$$

  - For each internal node $\mathcal{D}$, run

$$(\mathsf{pk}_{\mathcal{D}}, \mathsf{vk}_{\mathcal{D}}) \leftarrow \mathsf{S}.\mathcal{G}(1^\lambda, \mathcal{D})\,.$$

  - Set $\mathsf{pk} = \mathsf{upk} = (\{(\mathsf{pk}_{\mathcal{C}_i}, \mathsf{vk}_{\mathcal{C}_i})\}_{i\in[n]},\ \{(\mathsf{pk}_{\mathcal{D}}, \mathsf{vk}_{\mathcal{D}})\}_{\mathcal{D}})$.
  - Set $\mathsf{vk} = (\mathsf{vk}_{\mathsf{root}}, \mathsf{vk}_\sigma = \{[\sigma(6n+i)]\}_{i\in[n_0]})$.

– $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{aux})$:
  - For $i = 1, \ldots, n$, compute h_i as in Equation 1.
  - Compute h_{pub} as in Equation 2.
  - For all leaves $i = 1, \ldots, n$, $\pi_{\mathcal{C}_i} \leftarrow \mathsf{S}.\mathcal{P}(\mathsf{pk}_{\mathcal{C}_i}, h_i, (\mathbb{w}[jn+i])_{j\in[0,5]})$.
  - For all internal nodes $\mathcal{D}$, $\pi_{\mathcal{D}} \leftarrow \mathsf{S}.\mathcal{P}(\mathsf{pk}_{\mathcal{D}}, h_L \cdot h_R, (\pi_L, \pi_R, h_L, h_R))$.
  - For the root, $\pi_{\mathsf{root}} \leftarrow \mathsf{S}.\mathcal{P}(\mathsf{pk}_{\mathcal{D}}, h_{\mathsf{pub}}, (\pi_L, \pi_R, h_L, h_R))$.
  - Output $\pi = \pi_{\mathsf{root}}$. Include all proofs and h_{pub} in aux.

– $\mathcal{U}(\mathsf{upk}, \mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}, \pi, \mathsf{aux}) \to (\pi', \mathsf{aux}')$:
  - Determine the set of affected values due to the update from $\mathbb{w}||\mathbb{x}$ to $\mathbb{w}'||\mathbb{x}'$.
  - Update corresponding h_i, and $\pi_{\mathcal{C}_i}, \pi_{\mathcal{D}}$ from leaf to root.
  - Update related terms in π and aux to π' and aux'.

– $\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$:
  - Compute h_{pub} as in Equation 2.
  - Check $\mathsf{S}.\mathcal{V}(\mathsf{vk}_{\mathsf{root}}, h_{\mathsf{pub}}, \pi_{\mathsf{root}})$.

Fig. 13. Dynarec from a recursive zk-SNARK $(\mathcal{G}, \mathcal{P}, \mathcal{V})$.

□

The following lemma is Lemma 1 from [33] that is helpful to prove the zero-knowledge property. We refer to [33] to see its formal proof.

Lemma 10 ([33]). *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 $\rho \leftarrow \mathbb{F}^{(\le k-1)}[X]$ of degree $k-1$ and define*

$$\tilde{f}(X) := f(X) + Z_S(X)\rho(X)\,.$$

2. *Output $(\tilde{f}(x_1), \ldots, \tilde{f}(x_k)) \in \mathbb{F}^k$.*

D Inner Product Arguments

Bünz et al. [9] 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:

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

We give an abstraction for their non-interactive argument for $\mathcal{R}^m_{\mathsf{IPA}}$:

- $\mathcal{G}_{\mathsf{IPA}}(1^\lambda, m) \to (\mathsf{pk}, \mathsf{vk})$: Outputs $\mathsf{pk} = ([g^{\alpha^i}]_{i\in[0,2m-2]}, [h^{\beta^i}]_{i\in[0,2m-2]})$ and $\mathsf{vk} = (g^\beta, h^\alpha)$.
- $\mathcal{P}_{\mathsf{IPA}}(\mathsf{pk}, \mathbb{x}_{\mathsf{IPA}}, \mathbb{w}_{\mathsf{IPA}}) \to \pi$: Outputs a proof π that $(\mathbb{x}_{\mathsf{IPA}}, \mathbb{w}_{\mathsf{IPA}}) \in \mathcal{R}^m_{\mathsf{IPA}}$.
- $\mathcal{V}_{\mathsf{IPA}}(\mathsf{vk}, \mathbb{x}_{\mathsf{IPA}}, \pi) \to 0/1$: Verifies the proof π that $(\mathbb{x}_{\mathsf{IPA}}, \mathbb{w}_{\mathsf{IPA}}) \in \mathcal{R}^m_{\mathsf{IPA}}$.

$\mathcal{P}_{\mathsf{IPA}}$ takes $O(m)$ time, $\mathcal{V}_{\mathsf{IPA}}$ takes $O(\log m)$ time (using the optimization in Section 5 of [9]), and the proof size is $|\pi| = O(\log m)$.

E 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 11. *For any PPT algebraic adversary $\mathcal{A}$, given $\mathsf{pp_{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$ 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:

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

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

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

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 $\mathsf{pp_{bl}}$ 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 9, $g(X) := \sum_{i\in[t]} f_{1,i}(r_1X,\ldots,r_sX)\cdot f_{2,i}(r_1X,\ldots,r_sX)$ 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. □

Variable 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

$$e([f(X_1,\ldots,X_s)],[X_1^{d_1}]) = e([f(X_1,\ldots,X_s)X_1^{d_1}],g) \qquad \text{(Variable check)}$$

Similarly, we can check the following to ensure f has no variable X_1, X_3:

$$e([f(X_1,\ldots,X_s)],[X_1^{d_1}X_3^{d_3}]) = e([f(X_1,\ldots,X_s)X_1^{d_1}X_3^{d_3}],g)$$

The following lemma states that it suffices to use variable checks to ensure some f is not a polynomial over some variable(s).

Lemma 12. *For any PPT algebraic adversary $\mathcal{A}$, given $\mathsf{pp_{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$ 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:*

$$\Pr\left[\begin{array}{ll} C=[f(X_1,\ldots,X_s)], C'=[f'(X_1,\ldots,X_s)] & \\ \wedge \mathsf{deg}_{X_1} f(X_1,\ldots,X_s) > 0 & : (C,C') \leftarrow \mathcal{A}(\mathsf{pp_{bl}},\mathbf{L}) \\ \wedge e([C,[X_1^{d_1}]_2) = e(C',g_2) & \end{array}\right]$$

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:

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

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 $\mathsf{pp_{bl}}$ 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:

$$\begin{aligned} &\mathsf{deg}_{X_1} f(X_1,\ldots,X_s) > 0 \\ \wedge\ &e([f(X_1,\ldots,X_s)],[X_1^{d_1}]) = e([f'(X_1,\ldots,X_s)],g)\,, \end{aligned}$$

   then $\mathcal{A}^*$ knows that $f(X_1,\ldots,X_s)\cdot X_1^{d_1} - f'(X_1,\ldots,X_s)$ is a non-zero polynomial which evaluates 0 on $(\alpha_1,\ldots,\alpha_s)$. According to Lemma 9, the polynomial $g(X) := f(r_1X,\ldots,r_sX)\cdot(r_1X)^{d_1} - f'(r_1X,\ldots,r_sX)$ is a zero polynomial with probability at most $d/|\mathbb{F}|$ (d the maximum degree of any variable in $f(X_1,\ldots,X_s)\cdot X_1^{d_1} - f'(X_1,\ldots,X_s)$), 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. □

F Other proofs

F.1 Proof of Theorem 1.

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

- $z_i(X) = L_i(X)$;
- $v_i(X,Y) = \mathbf{y}[i]L_i(X)$;
- $p_i(X,Y) = \mathbf{y}[i]\sum_{j=i}^{m} L_j(X)$;
- $t_i(X,Y) = \mathbf{y}[i]\sum_{j=i}^{m} L_j(X\cdot\omega^{-1})$;
- $g_i(W,Y) = \mathbf{y}[i]\sum_{j=i}^{m} L_j(W)$;
- $Z_i(X,W,Y) = Y^m \cdot W^m \cdot L_i(X)$;
- $V_i(X,W,Y) = W^m \cdot \mathbf{y}[i]L_i(X)$;
- $P_i(X,W,Y) = W^m \cdot \mathbf{y}[i]\sum_{j=i}^{m} L_j(X)$;
- $T_i(X,W,Y) = W^m \cdot \mathbf{y}[i]\sum_{j=i}^{m} L_j(X\cdot\omega^{-1})$;
- $G_i(X,W,Y) = X^m \cdot \mathbf{y}[i]\sum_{j=i}^{m} L_j(W)$;
- $\alpha_i(X,Y) = \frac{L_i(X)(\mathbf{y}[i]-u(X,Y))}{X^m-1}$;
- $\beta_i(X,Y) = \mathbf{y}[i]\cdot\frac{\sum_{j=i+1}^{m} L_j(X)}{X-\omega}$;
- $\gamma_i(X,Y) = \mathbf{y}[i]\cdot\frac{\sum_{j=i}^{m-1} L_j(X)}{X-1}$;
- $\delta_i(X,W,Y) = \mathbf{y}[i]\cdot\sum_{j=i}^{m}\frac{L_j(X)-L_j(W)}{X-W}$;
- $\varepsilon_i(X,W,Y) = \mathbf{y}[i]\cdot\sum_{j=i}^{m}\frac{L_j(W)-L_j(X\cdot\omega^{-1})}{W-X\cdot\omega^{-1}}$.

□

F.2 Proof of Lemma 1

We present the universal protocol in Fig. 14. We only need to show the complexity of $\mathcal{G}$ and $\mathcal{I}$ here (all other parts are the same as the proof of the circuit-specific version). For the runtime of $\mathcal{G}$, since this algorithm exactly knows the secrets a, b, c, it can computes everything from scratch in $O(m^2)$ time. For the runtime of $\mathcal{I}$, let us take a look at every $f \in \mathcal{F}$:

- $[z_i], [v_i], [p_i], [g_i]$. Directly extracted/computed from pp in $O(m^2)$ time.
- $[t_i(X,Y)] = [\mathbf{y}[i]\sum_{j=i}^{m} L_j(X\cdot\omega^{-1})]$. Similar to $[p_i]$. Note that

$$L_j(X\cdot\omega^{-1}) = \frac{\omega^j((X\cdot\omega^{-1})^m-1)}{m(X\cdot\omega^{-1}-\omega^j)} = \frac{\omega^{j+1}(X^m-1)}{m(X-\omega^{j+1})} = L_{j+1}(X)\,.$$

- $[Z_i], [V_i], [P_i], [T_i], [G_i]$ are similar to $[z_i], [v_i], [p_i], [t_i], [g_i]$.
- $[\alpha_i(X,Y)] = [\frac{L_i(X)(\mathbf{y}[i]-u(X,Y))}{X^m-1}]$. Expand this polynomial,

$$\begin{aligned}&\frac{L_i(X)(\mathbf{y}[i]-u(X,Y))}{X^m-1}\\ =&\sum_{j\in[m]\setminus i}\frac{\mathbf{y}[j](\omega^j L_i(X)-\omega^i L_j(X))}{m(\omega^j-\omega^i)} - \frac{\omega^i(L_i(X)-1)\mathbf{y}[i]}{m(X-\omega^i)}\,,\end{aligned}$$

– $\mathcal{G}(1^\lambda, m) \to (\mathsf{pp})$:
  - $\mathsf{pp}_{\mathsf{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$;
  - Pick random a, b, c from $\mathbb{F}$ for variables X, Y and W respectively. Let $c' = c \cdot \omega^{-1}$.
  - Set $\mathsf{pp} = \mathsf{pp}_{\mathsf{bl}}$ and all the following:

$$\{[a^i],\ [a^i b^m c^m]\}_{i\in[m]} \quad \{[a^i b^j],\ [a^i b^j c^m],\ [b^j c^i],\ [a^m b^j c^i]\}_{i,j\in[m]}$$
$$\{[L_i(a) b^j],\ [L_i(a) b^j c^m],\ [b^j L_i(c)],\ [a^m b^j L_i(c)]\}_{i\in[m],j\in[0,m]}$$
$$\left\{\left[b^j \frac{L_i(a)-1}{a-\omega^i}\right],\ \left[b^j \frac{L_i(a)-L_i(c)}{a-c}\right],\ \left[b^j \frac{L_i(a)-L_i(c')}{a-c'}\right]\right\}_{i,j\in[m]}$$

– $\mathcal{I}(\mathsf{pp}, [m, \sigma]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - Let $\mathcal{F} = \{z, v, p, t, g, Z, V, P, T, G, \alpha, \beta, \gamma, \delta, \varepsilon\}$ be the set of polynomials from Theorem 1.
  - Set $\mathsf{pk} = \mathsf{upk}$ to contain the following, computed using pp

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

  - Set $\mathsf{vk} = \{[u(X,Y)], [X], [W], [X^m], [Y^m W^m], [W^m]\}$ (u is from Eq. (5)).

Fig. 14. The universal Dynamo SNARK. $\mathcal{P}$, $\mathcal{U}$, $\mathcal{V}$ are the same as Fig. 2.

which can be computed from $[Y^j \frac{L_i(X)-1}{X-\omega^i}]$ and $[Y^j L_i(X)]$ in $O(m^2)$ time.

– $\gamma_i(X,Y) = \mathbf{y}[i] \cdot \frac{\sum_{j=i}^{m-1} L_j(X)}{X-1}$. Note that when $i \in [m-1]$,

$$\frac{L_i(X)}{X-1} = \frac{\omega^i(X^m-1)}{m(X-\omega^i)(X-1)} = \frac{L_i(X)-\omega^i L_1(X)}{\omega^i-1}\,,$$

then $[\gamma_i]$ can be computed from $[Y^j L_i(X)]$ in $O(m^2)$ time.

– $\beta_i(X,Y) = \mathbf{y}[i] \cdot \frac{\sum_{j=i+1}^{m} L_j(X)}{X-\omega}$. Similar to $[\gamma_i]$.

– $\delta_i(X,W,Y) = \mathbf{y}[i] \cdot \sum_{j=i}^{m} \frac{L_j(X)-L_j(W)}{X-W}$. Compute from $[Y^j \frac{L_i(X)-L_i(W)}{X-W}]$ in $O(m^2)$ time.

– $\varepsilon_i(X,W,Y) = \mathbf{y}[i] \cdot \sum_{j=i}^{m} \frac{L_j(W)-L_j(X\cdot\omega^{-1})}{W-X\cdot\omega^{-1}}$. Similar to $[\delta_i]$.

F.3 Proof of Lemma 2

We introduce how to add zero-knowledge for Dynamo here. Following the idea of [20,33], we can use random mask polynomials. More specifically, we introduce masks for z, v, p, t, g:

$$z^{\mathsf{zk}}(X) = z(X) + \rho_z \cdot (X^m - 1)\,,$$
$$v^{\mathsf{zk}}(X,Y) = v(X,Y) + \rho_v \cdot (X^m - 1)\,,$$
$$p^{\mathsf{zk}}(X,Y) = p(X,Y) + (\rho_p^{(2)} X^2 + \rho_p^{(1)} X + \rho_p^{(0)}) \cdot (X^m - 1)\,,$$

$$t^{\mathsf{zk}}(X,Y) = t(X,Y) + (\rho_p^{(2)}(X\omega^{-1})^2 + \rho_p^{(1)}(X\omega^{-1}) + \rho_p^{(0)}) \cdot (X^m - 1)\,,$$
$$g^{\mathsf{zk}}(W,Y) = g(W,Y) + (\rho_p^{(2)}W^2 + \rho_p^{(1)}W + \rho_p^{(0)}) \cdot (W^m - 1)\,,$$

where the randomness $\rho_z, \rho_v, \rho_p^{(2)}, \rho_p^{(1)}, \rho_p^{(0)} \xleftarrow{\$} \mathbb{F}$ should be picked at the beginning of $\mathcal{P}^{\mathsf{zk}}$ and $\mathcal{U}^{\mathsf{zk}}$.

Based on $z^{\mathsf{zk}}, v^{\mathsf{zk}}, p^{\mathsf{zk}}, t^{\mathsf{zk}}, g^{\mathsf{zk}}$, we can naturally derive corresponding Z^{zk}, V^{zk}, P^{zk}, T^{zk}, G^{zk} for variable checks. However, for new quotient polynomials $\alpha^{\mathsf{zk}}, \beta^{\mathsf{zk}}, \gamma^{\mathsf{zk}}, \delta^{\mathsf{zk}}, \varepsilon^{\mathsf{zk}}$, we need to carefully calculate their forms. Replace z, v, p, t, g in Eqs. (6), (9), (10), (13) and (14) with $z^{\mathsf{zk}}, v^{\mathsf{zk}}, p^{\mathsf{zk}}, t^{\mathsf{zk}}, g^{\mathsf{zk}}$, we have

- $\alpha^{\mathsf{zk}}(X,Y) = \alpha(X,Y) + \rho_v - \rho_z u(X,Y)\,;$
- $\beta^{\mathsf{zk}}(X,Y) = \beta(X,Y) + ((\rho_p^{(2)}X^2 + \rho_p^{(1)}X + \rho_p^{(0)}) - \rho_v) \cdot \frac{X^m-1}{X-\omega}\,;$
- $\gamma^{\mathsf{zk}}(X,Y) = \gamma(X,Y) + (\rho_p^{(2)}X^2 + \rho_p^{(1)}X + \rho_p^{(0)}) \cdot \frac{X^m-1}{X-1}\,;$
- $\delta^{\mathsf{zk}}(X,W,Y) = \delta(X,W,Y) + \rho_p^{(2)}\frac{(X^{m+2}-W^{m+2})-(X^2-W^2)}{X-W} + \rho_p^{(1)}(\frac{X^{m+1}-W^{m+1}}{X-W} - 1) + \rho_p^{(0)}\frac{X^m-W^m}{X-W}\,;$
- $\varepsilon^{\mathsf{zk}}(X,W,Y) = \varepsilon(X,W,Y) + \rho_p^{(2)}\frac{(W^{m+2}-(X\omega^{-1})^{m+2})-(W^2-(X\omega^{-1})^2)}{W-X\omega^{-1}} + \rho_p^{(1)}(\frac{W^{m+1}-(X\omega^{-1})^{m+1}}{W-X\omega^{-1}} - 1) + \rho_p^{(0)}\frac{W^m-(X\omega^{-1})^m}{W-X\omega^{-1}}\,.$

Now we can use

$$\mathcal{F}^{\mathsf{zk}} = \{z^{\mathsf{zk}}, v^{\mathsf{zk}}, p^{\mathsf{zk}}, t^{\mathsf{zk}}, g^{\mathsf{zk}}, Z^{\mathsf{zk}}, V^{\mathsf{zk}}, P^{\mathsf{zk}}, T^{\mathsf{zk}}, G^{\mathsf{zk}}, \alpha^{\mathsf{zk}}, \beta^{\mathsf{zk}}, \gamma^{\mathsf{zk}}, \delta^{\mathsf{zk}}, \varepsilon^{\mathsf{zk}}\}$$

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:

$\mathcal{S}(1^\lambda, \mathbb{i}) \to (t, \mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:

Follow every step of $\mathcal{G}(1^\lambda, [m, \sigma])$, and output $(t = (a, b, c), \mathsf{pk}, \mathsf{upk}, \mathsf{vk})$.

$\mathcal{S}(t, \mathsf{pk}, \mathsf{upk}, \mathsf{vk}, \mathbb{x}_0, \ldots, \mathbb{x}_l) \to (\tilde{\pi}_0, \ldots, \tilde{\pi}_l)$:

For every $i \in [0, l]$,

(a) For every $f \in \{z, v, p, t, g\}$, pick $\tau_f \xleftarrow{\$} \mathbb{F}$ and let $[\widetilde{f^{\mathsf{zk}}}] = [\tau_f]$. Also compute the variable-check polynomials for f from (a, b, c).

(b) For every $f \in \{\alpha, \beta, \gamma, \delta, \varepsilon\}$, compute $[\widetilde{f^{\mathsf{zk}}}]$ as following:

$$[\widetilde{\alpha^{\mathsf{zk}}}] = \left[\frac{\tau_v - u(a,b)\tau_z}{a^m - 1}\right] \quad [\widetilde{\beta^{\mathsf{zk}}}] = \left[\frac{\tau_p - \tau_v}{a - \omega}\right] \quad [\widetilde{\gamma^{\mathsf{zk}}}] = \left[\frac{\tau_p}{a - 1}\right]$$
$$[\widetilde{\delta^{\mathsf{zk}}}] = \left[\frac{\tau_p - \tau_g}{a - c}\right] \quad [\widetilde{\varepsilon^{\mathsf{zk}}}] = \left[\frac{\tau_g - \tau_t}{c - a}\right]$$

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

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

For fixed polynomial $z(X)$ and a value a, if $a \notin \Omega$ and $\rho_z \xleftarrow{\$} \mathbb{F}$, then according to Lemma 10, $z^{\mathsf{zk}}(a) = z(a) + \rho_z(a^m - 1)$ is also uniform in $\mathbb{F}$.

For fixed polynomial $v(X,b)$ and a value a, if $a \notin \Omega$ and $\rho_v \xleftarrow{\$} \mathbb{F}$, then according to Lemma 10, $v^{\mathsf{zk}}(a,b) = v(a,b) + \rho_v(a^m - 1)$ is also uniform in $\mathbb{F}$.

For fixed polynomial $p(X,b)$ and value $a, c, \omega^{-1}a$, if $a, c, \omega^{-1}a \notin \Omega$ and $\rho_p^{(2)}, \rho_p^{(1)}, \rho_p^{(0)} \xleftarrow{\$} \mathbb{F}$, then according to Lemma 10 and the following calculation,

$$\begin{aligned}
p^{\mathsf{zk}}(a,b) &= p(a,b) + (\rho_p^{(2)}a^2 + \rho_p^{(1)}a + \rho_p^{(0)})(a^m - 1) \\
g^{\mathsf{zk}}(c,b) &= p(c,b) + (\rho_p^{(2)}c^2 + \rho_p^{(1)}c + \rho_p^{(0)})(c^m - 1) \\
t^{\mathsf{zk}}(a,b) &= p(\omega^{-1}a,b) + (\rho_p^{(2)}(\omega^{-1}a)^2 + \rho_p^{(1)}(\omega^{-1}a) + \rho_p^{(0)})((\omega^{-1}a)^m - 1)
\end{aligned}$$

$(p^{\mathsf{zk}}(a,b), g^{\mathsf{zk}}(c,b), t^{\mathsf{zk}}(a,b))$ is also uniform.

Above all, as long as $a, c, \omega^{-1}a \notin \Omega$ (which is of overwhelming probability),

$$(z^{\mathsf{zk}}(a), v^{\mathsf{zk}}(a,b), p^{\mathsf{zk}}(a,b), g^{\mathsf{zk}}(c,b), t^{\mathsf{zk}}(a,b))$$

is uniform in $\mathbb{F}^5$ and thus $\mathcal{S}$ can perfectly simulate $[f^{\mathsf{zk}}]_{f\in\{z,v,p,g,t\}}$. Based on the codes of the prover and the simulator, $[f^{\mathsf{zk}}]_{f\in\{Z,V,P,G,T,\alpha,\beta,\gamma,\delta,\varepsilon\}}$ are exactly determined by $[f^{\mathsf{zk}}]_{f\in\{z,v,p,g,t\}}$. Therefore, $\mathcal{S}$ can successfully simulate a prover.

F.4 Proof of Lemma 6

We briefly introduce how to add zero knowledge to Dynaverse here. The intuition is also to add mask polynomials, following the next two steps:

1. Similar to the way we add zero knowledge to Dynamo, we add mask polynomials to $\mathcal{F}$ in Dynamix. However, we cannot put $[h]$ in π_{ti} because it could leak some information about $\mathbb{w}$. We should remove all $[h], [H]$ in the proof and fix the equations with h with the following trick. Note that we can compute $\gamma_{ti}^{\mathsf{zk}}$ as

$$\gamma_{ti}^{\mathsf{zk}}(X,Y) = \frac{p_{ti}^{\mathsf{zk}}(X,Y) - h_{ti}(Y)}{X-1},$$

then we have

$$\sum_{t\in[1,6],i\in[m]} h_{ti}(Y) = \sum_{t\in[1,6],i\in[m]} p_{ti}^{\mathsf{zk}}(X,Y) - (X-1)\sum_{t\in[1,6],i\in[m]} \gamma_{ti}^{\mathsf{zk}}(X,Y).$$

Replace all the $[\gamma_{ti}]$ in the proof with one group element $[\gamma^{\mathsf{zk}}]$ where

$$\gamma^{\mathsf{zk}}(X,Y) = \sum_{t\in[1,6],i\in[m]} \gamma_{ti}^{\mathsf{zk}}(X,Y).$$

Then for $[h_{\mathbb{x}}(Y)] \cdot \prod_{t=1}^{6}\prod_{i=1}^{m}[h_{ti}(Y)] \stackrel{?}{=} 1_{\mathbb{G}}$, $\mathcal{V}$ can verify the following instead

$$e([h_{\mathbb{x}}], g) \cdot \left(\prod_{i\in[m],t\in[1,6]} e([p_{ti}^{\mathsf{zk}}], g)\right) \cdot e([-\gamma^{\mathsf{zk}}], [X-1]) \stackrel{?}{=} 1_{\mathbb{G}_T}.$$

$[\gamma^{\mathsf{zk}}]$ can be simulated from $[h_{\mathbb{x}}]$ and $[p_{ti}^{\mathsf{zk}}]$.

And for $e([p]\cdot[-t]\cdot[-v],g) \stackrel{?}{=} e([\frac{-\omega h}{m}],[\frac{(X^m-1)}{(X-\omega)}])$, a new quotient polynomial should be computed as follows (and can be easily simulated):

$$\frac{p^{\mathsf{zk}}(X,Y) - t^{\mathsf{zk}}(X,Y) - v^{\mathsf{zk}}(X,Y)}{(X^m-1)(X-\omega)^{-1}}.$$

2. We also need to add masks for $z_{t,i}$ for $t \in \{4,5,6\}$ and A_i should also be modified due to changes in Eq. (19).

F.5 Proof of Theorem 8

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

F.6 Proof of Lemma 8

The proof is similar to the proof of Theorem 8. We want to mention that although there are $O(\log M)$ wires changed for the $\mathsf{SUMTREE}$, they are basically all from addition gates. The wires changed from multiplication gates are all from the computation F.

G Vector Commitments and Hyperproofs [34]

Srinivasan et al. [34] present Hyperproofs, a vector commitment that is efficiently maintainable and aggregatable built on PST [30]. First we present some definitions for vector commitments.

Definition 4 (VC scheme). *A Vector Commitment (VC) scheme is a set of the following PPT algorithms.*

- $\mathsf{Gen}(1^\lambda, n) \to \mathsf{pp}$: *Given security parameter λ and vector size n, it outputs public parameters* pp.
- $\mathsf{Com}(\mathsf{pp}, \mathbf{z}) \to (C, \mathsf{aux})$: *Given vector $\mathbf{z}$, it outputs commitment C along with auxiliary information* aux.
- $\mathsf{Open}(\mathsf{pp}, i, \mathbf{z}, \mathsf{aux}) \to \pi_i$: *Given index i, vector $\mathbf{z}$ and auxiliary information* aux, *it outputs a proof π_i.*
- $\mathsf{OpenAll}(\mathsf{pp}, \mathbf{z}) \to (\pi_1, \ldots, \pi_n)$: *Given vector $\mathbf{z}$, it outputs all proofs $\pi_1, \ldots, \pi_n$.*
- $\mathsf{Agg}(\mathsf{pp}, I, (\pi_i, z_i)_{i \in I}) \to \pi_I$: *Given proof-value pairs $\{(\pi_i, z_i)\}_{i \in I}$, for $I \subseteq [n]$, it outputs an aggregate proof π_I.*
- $\mathsf{Verify}(\mathsf{pp}, C, I, \{z_i\}_{i \in I}, \pi_I) \to 0/1$: *Given commitment C, values $\{z_i\}_{i \in I}$, for $I \subseteq [n]$, and (aggregate) proof π_I, it outputs either 0 or 1.*
- $\mathsf{UpdCom}(\mathsf{pp}, i, \delta, C) \to C'$: *Given index i, value δ, commitment C, it outputs C' reflecting position i changing by δ.*
- $\mathsf{UpdAll}(\mathsf{pp}, i, \delta, \{\pi_j\}, \mathsf{aux}) \to (\{\pi'_j\}, \mathsf{aux}')$: *Given index i, difference δ, proofs $\{\pi_j\}_{j \in [n]}$ and auxiliary information* aux, *it outputs updated proofs $\{\pi'_j\}_{j \in [n]}$ and updated auxiliary information* aux', *to reflect position i changing by δ.*

Definition 5 (VC correctness). *A VC scheme is correct if for all $\lambda \in \mathbb{N}$ and $n = \mathsf{poly}(\lambda)$, for all $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, n)$, for all vectors $\mathbf{z}$, for all $i \in [n]$, if $(C, \mathsf{aux}) = \mathsf{Com}(\mathsf{pp}, \mathbf{z})$ and $\pi_i = \mathsf{Open}(\mathsf{pp}, i, \mathbf{z}, \mathsf{aux})$ (or π_i is derived from* $\mathsf{OpenAll}$*), then, for any polynomial number of updates (j, δ) resulting in a new vector $\mathbf{z}'$, if C' and π'_i are obtained via calls to* UpdCom *and* UpdAll *(with* aux *replaced by* aux'*) respectively, then*

1. $\Pr[1 \leftarrow \mathsf{Verify}(\mathsf{pp}, C', i, z'_i, \pi'_i)] = 1$;
2. *For all $I \subseteq [n]$ it is*

$$\Pr[1 \leftarrow \mathsf{Verify}(\mathsf{pp}, C', I, (z'_i)_{i \in I}, \mathsf{Agg}(\mathsf{pp}, I, (\pi'_i, z'_i)_{i \in I}))] = 1.$$

Definition 6 (VC soundness). *For all PPT adversaries $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, n), \\ (C, I, J, (z_i)_{i \in I}, (z'_j)_{j \in J}, \pi_I, \pi_J) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}) : \\ 1 \leftarrow \mathsf{Verify}_{\mathsf{pp}}(C, I, (z_i)_{i \in I}, \pi_I) \wedge \\ 1 \leftarrow \mathsf{Verify}_{\mathsf{pp}}(C, J, (z'_j)_{j \in J}, \pi'_J) \wedge \\ \exists k \in I \cap J \text{ such that } z_k \neq z'_k \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Hyperproofs. We refer to [34] for the details of Hyperproofs. Here we only briefly present the necessary part (without aggregation) for this paper.

Multilinear extension (MLE) of a vector. For a vector $\mathbf{z}$ of length $n = 2^l$, we define the following multilinear extension polynomials [34,42] for itself and its sub-vectors:

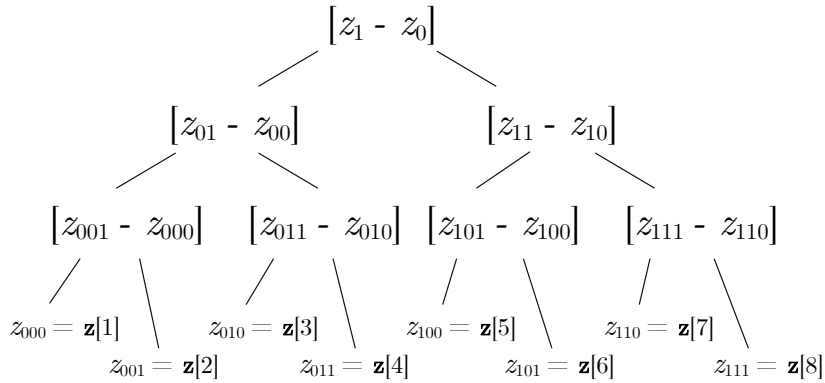

Fig. 15. A multilinear tree (MLT) of size 8. The proof for $z_{\mathbf{i}}$ consists all nodes along $z_{\mathbf{i}}$'s path to the root (e.g., for z_{100}, $([z_1 - z_0], [z_{11} - z_{10}], [z_{101} - z_{100}])$).

– For any $i \in [n]$, suppose $\overline{i_1 \dots i_l}$ is the binary form of $i - 1$, i.e., $i = 1 + \sum_{j=1}^{l} i_j \cdot 2^{l-j}$. Define

$$z_{i_1 \dots i_l} := \mathbf{z}[i] \,.$$

– For any $1 \le k < l$ and $i_1, \dots, i_k \in \{0, 1\}$, define

$$z_{i_1 \dots i_k}(X_{k+1}, \dots, X_l) := (1 - X_{k+1}) \cdot z_{i_1 \dots i_k 0}(X_{k+2}, \dots, X_l) + X_{k+1} \cdot z_{i_1 \dots i_k 1}(X_{k+2}, \dots, X_l) \,.$$

– Define

$$z(X_1, \dots, X_l) := (1 - X_1) \cdot z_0(X_2, \dots, X_l) + X_1 \cdot z_1(X_2, \dots, X_l) \,.$$

Expand $z(X_1, \dots, X_l)$ in details, we have

$$z(X_1, \dots, X_l) = \sum_{i_1, \dots, i_l \in \{0,1\}} z_{i_1 \dots i_l} \cdot \mathcal{S}_{i_1 \dots i_l}(X_1, \dots, X_l) \,,$$

where

$$\mathcal{S}_{i_1 \dots i_l}(X_1, \dots, X_l) = \prod_{j=1}^{l} \left(i_j X_j + (1 - i_j)(1 - X_j) \right) \,.$$

In fact $z_{i_1 \dots i_k}$ is a MLE of the following sub-vector:

$$\mathbf{z}\left[s : s + 2^{l-k} - 1 \right], \text{ where } s = 1 + \sum_{j=1}^{k} i_j \cdot 2^{l-j} \,.$$

For instance, for $n = 8$, z_0 is the MLE for $\mathbf{z}[1:4]$ and z_{10} is for $\mathbf{z}[5:6]$.

To simplify notations, in Hyperproofs we may use indices of form $\mathbf{i} = (i_1, \dots, i_l) \in \{0, 1\}^l$ and access the values of the vector by subscriptions $z_{\mathbf{i}}$ or evaluating $z(\mathbf{i})$.

Multilinear trees (MLT). Hyperproofs use MLTs to maintain the proofs for all positions. See Fig. 15 for an example of a MLT with $n = 8$ and $l = 3$. The commitment to the whole vector is $C = [z]$. The proof for $z_{\mathbf{i}}$ at position $\mathbf{i}$ consists of all the nodes in MLT from the root to $z_{\mathbf{i}}$. Denote the proof

for $z_{\mathbf{i}}$ as $\pi_{\mathbf{i}} = (\pi_{\mathbf{i},1}, \ldots, \pi_{\mathbf{i},h})$, where

$$\pi_{\mathbf{i},j} = [z_{i_1 \ldots i_{j-1}1} - z_{i_1 \ldots i_{j-1}0}] \,.$$

To verify the proof for $z_{\mathbf{i}}$, the verifier checks the following:

$$e(C/g^{z_{\mathbf{i}}}, g) = \prod_{j=1}^{l} e(\pi_{\mathbf{i},j}, [X_j - i_j]) \,.$$

We omit the detailed full protocol for Hyperproofs and only show their main theorem here.

Assumption 2 (q-Strong Diffie-Hellman (q-SDH)) *Let $\tau \in_R \mathbb{Z}_P^*$. Given as input a $(q+1)$-tuple $(g, g^{\tau}, \ldots, g^{\tau^q}) \in \mathbb{G}^{q+1}$, for any adversary $\mathcal{A}_{q\text{-}SDH}$, we have the following for any $a \in \mathbb{Z}_p \backslash \{-\tau\}$:*

$$\Pr[\mathcal{A}_{q\text{-}SDH}(g, g^{\tau}, \ldots, g^{\tau^q}) = (a, g^{\frac{1}{\tau+a}})] \leq \mathsf{negl}(\lambda)$$

Theorem 9 (Hyperproofs [34]). *The individual $\log n$-sized proofs are correct and sound as per Assumption 2.*

Proof for an empty sub-tree. Now we consider a simple trick based on the original Hyperproofs to prove that some $z_{i_1 \ldots i_k}$ is identical to 0, which is equivalent to show the following

$$\forall i'_{k+1}, \ldots, i'_l \in \{0,1\}, z_{i_1 \ldots i_k i'_{k+1} \ldots i'_l} = 0 \,.$$

Consider the MLT in such case. Since for any $i'_{k+1}, \ldots, i'_l \in \{0,1\}$ we have $z_{i_1 \ldots i_k i'_{k+1} \ldots i'_l} = 0$, the sub-tree with root $[z_{i_1 \ldots i_k 1} - z_{i_1 \ldots i_k 0}]$ is full of empty nodes. Furthermore, all the leaf nodes $z_{i_1 \ldots i_k i'_{k+1} \ldots i'_l}$ share the same path from the root to the node $[z_{i_1 \ldots i_k 1} - z_{i_1 \ldots i_k 0}]$, so they also share exact the same verification equation:

$$e(C/g^0, g) = \prod_{j=1}^{k} e([z_{i_1 \ldots i_{j-1}1} - z_{i_1 \ldots i_{j-1}0}], [X_j - i_j]) \,. \tag{28}$$

This equation is all we need to prove that $z_{i_1 \ldots i_k}$ is identical to 0.

H Ensuring the IVC transferred state is zero-knowledge

Recall that in the update algorithm of Dynaverse (cf. Fig. 3), the only place we need the whole witnesses for one bucket is where we compute quotient polynomials $A'_i(X)$ from Eq. (19). 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):

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

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

$$l_{ij}(X) = \frac{L_i(X) \cdot L_j(X)}{X^m - 1} \,.$$

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

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

then we have

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

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 $[l_{ij}(X)]$. 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 $O(m)$ size since updates in IVC are monotonically increasing.

I Details for Dynamix and Dynaverse

– $\mathcal{G}(1^\lambda, [m, N, \mathbf{s}, \mathbf{t}]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - $\mathsf{pp_{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$.
  - Let $\mathcal{F} = \{z, v, p, t, g, h, Z, V, P, T, G, H, \alpha, \beta, \gamma, \delta, \varepsilon\}$ be the set of polynomials from Theorem 1, including h and H from Equation 18.
  - Pick random a, b, c from $\mathbb{F}$ for variables X, Y and W respectively.
  - Set $\mathsf{pk} = \mathsf{upk}$ to contain the following KZG commitments, defined in Theorem 1, and computed using a, b and c directly

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

  - Set

$$\mathsf{vk} = \{[u], [X], [W], [X^m], [(X^m - 1)(X - \omega)^{-1}], [Y^N W^m], [W^m], [X^m W^m]\} ,$$

  where u is $u(X, Y) = \sum_{i \in [m]} L_i(X) \cdot (Y^{s_i} - Y^{t_i})$.

– $\mathcal{P}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{aux})$:
  - Parse $\mathbb{x}$ as $\emptyset$ and $\mathbb{w}$ as $\mathbf{z}[1], \ldots, \mathbf{z}[m]$.
  - Output $|\mathcal{F}| = 17$ KZG commitments as π and aux, i.e., for all $f \in \mathcal{F}$ output

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

– $\mathcal{U}(\mathsf{upk}, \mathbb{x}', \mathbb{w}', \mathbb{x}, \mathbb{w}, \pi, \mathsf{aux}) \to (\pi', \mathsf{aux}')$:
  - Parse $\mathbb{w}$ as $\mathbf{z}$ and $\mathbb{w}'$ as a new valid witness $\mathbf{z}'$. Parse π and aux as $\{[f]\}_{f \in \mathcal{F}}$.
  - Let J be the set of locations that $\mathbf{z}$ and $\mathbf{z}'$ differ and let $\{\delta_j\}_{j \in J}$ be the corresponding deltas. Output as π' and aux' the new KZG commitments $\{[f']\}_{f \in \mathcal{F}}$ where

$$[f'] = [f] \cdot \prod_{j \in J} [f_j]^{\delta_j} .$$

– $\mathcal{V}(\mathsf{vk}, \mathbb{x}, \pi) \to 0/1$:
  - Parse vk and π as output by $\mathcal{G}$ and $\mathcal{P}$ respectively.
  - Output 1 if and only if all the following relations hold:

$$e([v], g) \cdot e([-u], [z]) = e([\alpha], [X^m - 1]).$$
$$e([p], g) \cdot e([-v], g) = e([\beta], [X - \omega]).$$
$$e([p] \cdot [-h], g) = e([\gamma], [X - 1]).$$
$$e([p] \cdot [-t] \cdot [-v], g) = e([-\omega h/m], [(X^m - 1) \cdot (X - \omega)^{-1}]).$$
$$e([p] \cdot [-g], g) = e([\delta], [X - W]).$$
$$e([g] \cdot [-t], g) = e([\varepsilon], [W - X \cdot \omega^{-1}]).$$
$$e([z], [Y^N W^m]]) = e([Z], g).$$
$$e([v], [W^m]]) = e([V], g).$$
$$e([p], [W^m]]) = e([P], g).$$
$$e([t], [W^m]]) = e([T], g).$$
$$e([g], [X^m]]) = e([G], g).$$
$$e([h], [X^m W^m]]) = e([H], g).$$

Fig. 16. The Dynamix SNARK.

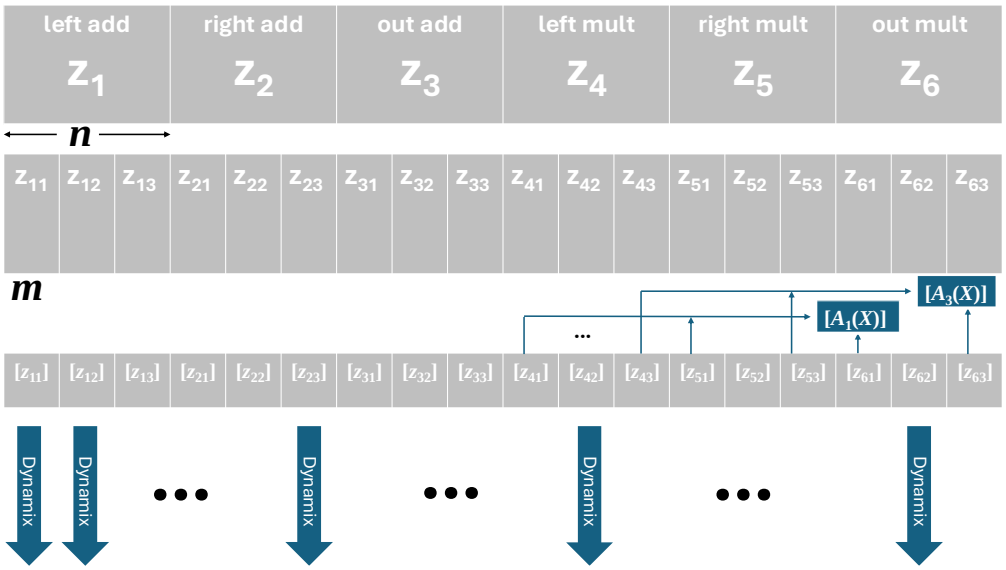

Fig. 17. The Dynaverse dynamic SNARK. The initial $6n$-sized witness $[\mathbf{z}_1 \dots \mathbf{z}_6]$ is split into subvectors $\mathbf{z}_{ij}$ of size $m = \sqrt{n}$, which are KZG-committed to $[z_{ij}]$. For each $[z_{ij}]$, we provide a Dynamix proof with respect to the permutation σ. For every $[z_{ij}]$ participating in multiplications ($i = 4, 5, 6$) we provide commitments to the quotient polynomials $A_i(X)$.

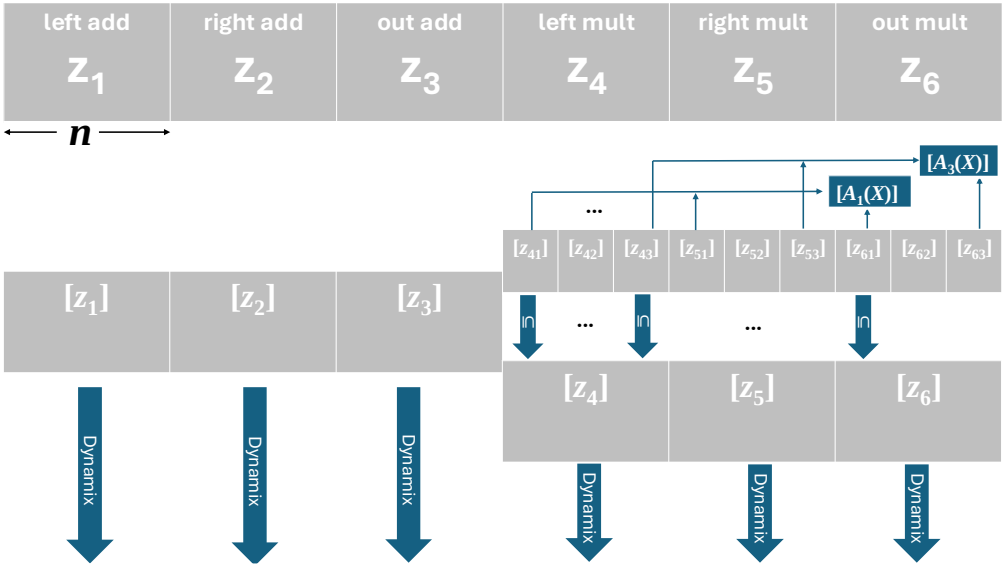

Fig. 18. The optimized Dynaverse dynamic SNARK. The addition wires are committed with three commitments $[z_1]$, $[z_2]$ and $[z_3]$ of n-sized vectors. The multiplication wires are committed in two ways, i.e., first with three commitments $[z_4]$, $[z_5]$ and $[z_6]$ of n-sized vectors and then with $3m$ commitments of m-sized vectors, as before. Overall we reduce the number of Dynamix proofs to six, we maintain the quotients polynomials, and we provide additional subvector proofs, denoted with "$\subseteq$", to ensure consistency between $[z_i]$ and $[z_{ij}]$ for $i = 4, 5, 6$.