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 put forth 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 $(\mathbb{x}, \mathbb{w}) \in \mathcal{L}$ along with a verifying proof π, and a valid target statement-witness pair $(\mathbb{x}', \mathbb{w}') \in \mathcal{L}$. 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 presenting the formal definition of dynamic zk-SNARKs, we provide two constructions. The first one is based on recursive SNARKs and has $O(\log n)$ update time. However it suffers from heuristic security—it must encode the random oracle in the SNARK circuit. The second one and our central contribution, Dynaverse, is based solely on KZG commitments and has $O(\sqrt{n} \log n)$ update time. Our preliminary evaluation shows, that, while worse asymptotically, Dynaverse outperforms the recursive-based approach by at least 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 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 without recomputation.

Defining dynamic zk-SNARKs. Our first contribution is to define dynamic zk-SNARKs—see Definition 2. Naturally, a dynamic zk-SNARK for a language

Table 1. Circuit-specific Dynaverse performance w/o and with IPA [2]. k is the number of updates between source and target statements and with n the number of multiplication and addition gates. For universal Dynaverse, all complexities remain the same except for $\mathcal{G}$ that runs in $n\sqrt{n}$ outputting public parameters pp of size $n\sqrt{n}$.

scheme	key generation $\mathcal{G}$	prove $\mathcal{P}$	update $\mathcal{U}$	verify $\mathcal{V}$	proof $\lvert\pi\rvert$	prover key $\lvert\mathsf{pk}\rvert$	verifier key $\lvert\mathsf{vk}\rvert$
Dynaverse	n	$n \log n$	$k\sqrt{n} \log n$	$\sqrt{n}$	$\sqrt{n}$	n	$\sqrt{n}$
Dynaverse (IPA)	n	$n \log n$	$k\sqrt{n} \log n$	$\log n$	$\log n$	n	$\log n$

$\mathcal{L}$ is a SNARK $(\mathcal{G}, \mathcal{P}, V)$ with an additional *update* algorithm $\mathcal{U}$: Algorithm $\mathcal{U}$, run by the prover, takes as input a valid source statement-witness pair $(\mathbb{x}, \mathbb{w}) \in \mathcal{L}$ along with a verifying proof π, and a valid target statement-witness pair $(\mathbb{x}', \mathbb{w}') \in \mathcal{L}$. 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 [12], Plonk [11], Bulletproofs [3], Orion [26]) 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 the use of 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 divident polynomial that encodes the wire assignments.

Existing dynamic proof systems. 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 in the literature. Those generally fall into two categories. Constructions in the first category crucially rely on *recursive* zk-SNARKs [1]. For example, Incremental Verifiable Computation (IVC) [24] uses recursive zk-SNARKs to support *dynamic chain computations* and Reckle trees [17] use recursive zk-SNARKs to support *dynamic batch proofs* in vector commitments. However, recursive zk-SNARKs are not known to be practical, and the ones that seem to be (e.g., Plonky2 [23]), encode a random oracle in the SNARK circuit, leading to only *heuristic* security proofs. Constructions in the second category do not use recursive zk-SNARKs (thus potentially more practical and provably secure) but have limited expressiveness. For example, authenticated data structures [22] and updatable vector commitments [6,21] 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., [4], as well as functional vector commitments supporting linear functions, e.g., [5].

Summary of our contributions. The main question we are asking in this work is whether we can build dynamic zk-SNARKs for *general purpose computation*. We answer this question in the affirmative and provide two constructions: The first one is a "folklore construction" that is using recursive SNARKs—see Section 3.1. It has excellent asymptotic complexities ($O(\log n)$ update time) but

inherits the practicality and heuristic security issues outlined before. Our second construction, Dynaverse, is not using recursive SNARKs and is based solely on KZG commitments [13]. It has $O(\sqrt{n}\log n)$ update time and is summarized in the following section. An initial unoptimized implementation (see Section 6) shows that Dynaverse is at least an order of magnitude faster, in terms of updates, than the recursive approach. The detailed complexities of our KZG-based construction are shown in Table 1.

1.1 Dynaverse: A dynamic zk-SNARK without recursion

We now summarize Dynaverse—the central contribution of this paper: From Plonkish arithmitization [11], 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., [11]) 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 [13] 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 2). 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. The most challenging part of this step is to deal with multiplication constraints: To prove multiplication constraints, we use a standard approach from Plonk [11], 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 $(i-1) + (j-1)m + 1$ 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 [2], 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 overall permutation σ. An illustration of how Dynaverse performs bucketization, uses quotient proofs and Dynamix proofs is shown in Figure 6.

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 $\{n_1, n_1 + 1, \ldots, n_2 - 1, n_2\}$.

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 [13] of f is $g^{f(\alpha)}$, which we write as $[f(X)]$. The celebrated KZG commitment [13] 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 $\{[X^iY^j]\}_{i,j=0,\ldots,q}$. 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 $\mathtt{i}$ to denote an indexed relation, i.e., the description of the circuit checking a public statement $\mathtt{x}$ and a witness $\mathtt{w}$. We slightly abuse notation and write $(\mathtt{x}, \mathtt{w}) \in \mathtt{i}$ iff running $\mathtt{i}$ on $(\mathtt{x}, \mathtt{w})$ returns 1, so $\mathtt{i}$ is both the description of the computation and the set

of valid tuples in the language. For example, the indexed relation $\mathtt{i}_{\mathcal{P}} = [n, \sigma]$, where σ is a permutation of size n over domain $\mathbb{F}$, contains those $\mathtt{w}$ such that $\mathtt{w}[i] = \mathtt{w}[\sigma[i]]$ for all i (Note $\mathtt{x} = \emptyset$.)

Plonkish arithmetization. Per Plonkish arithmetization [7,11], an index $\mathtt{i}_{\mathcal{C}} = [n, n_0, \sigma]$ 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 $\mathtt{i}_{\mathcal{C}} = [n, n_0, \sigma]$ describing a circuit $\mathcal{C}$, an instance of public inputs $\mathtt{x} \in \mathbb{F}^{n_0}$, and a witness $\mathtt{w} \in \mathbb{F}^{6n}$, let $\mathbf{z} = [\mathtt{w}; \mathtt{x}] \in \mathbb{F}^{6n+n_0}$. We have $(\mathtt{x}, \mathtt{w}) \in \mathtt{i}_{\mathcal{C}}$ if and only if the following hold: (a) $(\emptyset, \mathbf{z}) \in \mathtt{i}_{\mathcal{P}} = [6n+n_0, \sigma]$. (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 [12] 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 [12]. For zk-SNARKs with universal setup, algorithm $\mathcal{G}$ below is separated into two algorithms, a universal generation $\mathcal{G}(1^\lambda) \to \mathsf{pp}$ and an indexer $\mathcal{I}(\mathsf{pp}, \mathtt{i}) \to (\mathsf{pk}, \mathsf{vk})$. To avoid complexity in our presentation, all our constructions are presented as circuit-specific, but we show how to turn them into universal. In both circuit-specific and universal zk-SNARKs, algorithm $\mathcal{G}$ must be trusted.

Definition 1 (zk-SNARKs). *A zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) for an indexed relation* $\mathtt{i}$ *is a tuple of PPT algorithms* $\mathsf{S} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ *with the following interface:*

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

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

– Completeness*: Let* $\mathcal{G}(1^\lambda, \mathtt{i}) \to (\mathsf{pk}, \mathsf{vk})$*. We say that* S *satisfies* completeness *if for any* $\mathtt{i}$*, for any* $(\mathbb{x}, \mathbb{w}) \in \mathtt{i}$*, 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* $\mathtt{i}$ *there exists a PPT extractor* $\mathcal{E}_\mathcal{A}$ *such that*

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

is negligible.
– Zero Knowledge*: Fix any* $\mathtt{i}$ *and* $(\mathbb{x}, \mathbb{w}) \in \mathtt{i}$*. Let* $\mathcal{D}$ *be the distribution of* π *as output by the experiment below.*
1. $\mathcal{G}(1^\lambda, \mathtt{i}) \to (\mathsf{pk}, \mathsf{vk})$*.*
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. $(t, \mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{S}(1^\lambda, \mathtt{i})$*.*
2. $\tilde{\pi} \leftarrow \mathcal{S}(t, \mathsf{pk}, \mathsf{vk}, \mathbb{x})$*.*
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 [10]. 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 [10,11], 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 C.

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

In this section we present the formal definition of dynamic zk-SNARKs and a first construction satisfying our definition (yet using a random oracle inside a SNARK circuit) based on recursive SNARKs. 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 is 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*

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

that

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

is negligible.

– Zero Knowledge*: Fix any* $\mathtt{i}$ *and* $(\mathbb{x}_0,\mathbb{w}_0) \in \mathtt{i},\ldots,(\mathbb{x}_\ell,\mathbb{w}_\ell) \in \mathtt{i}$ *for some polynomially bounded* ℓ*. Let* $\mathcal{D}$ *be the distribution of* $(\pi_0,\ldots,\pi_\ell)$ *as output by the experiment below.*
  1. $(\mathsf{pk},\mathsf{upk},\mathsf{vk}) \leftarrow \mathcal{G}(1^\lambda,\mathtt{i})$.
  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. $(t,\mathsf{pk},\mathsf{upk},\mathsf{vk}) \leftarrow \mathcal{S}(1^\lambda,\mathtt{i})$.
  2. $(\tilde{\pi}_0,\ldots,\tilde{\pi}_\ell) \leftarrow \mathcal{S}(t,\mathsf{pk},\mathsf{upk},\mathsf{vk},\mathbb{x}_0,\ldots,\mathbb{x}_\ell)$.

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

3.1 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 the recent scheme of Nguyen et al. [16] that builds a highly parallelizable SNARK from Proof-Carrying-Data (PCD), with necessary modifications to make it updatable (See below for an explanation of differences.) Suppose now S is a zk-SNARK protocol that we will use as a black box. Recall that S has the following interface:

– $\mathsf{S}.\mathcal{G}(1^\lambda,\mathtt{i}) \rightarrow (\mathsf{pk},\mathsf{vk})$. The circuit-specific setup algorithm.
– $\mathsf{S}.\mathcal{P}(\mathsf{pk},\mathbb{x},\mathbb{w}) \rightarrow \pi$. The prover algorithm.
– $\mathsf{S}.\mathcal{V}(\mathsf{vk},\mathbb{x},\pi) \rightarrow 0/1$. The verifier algorithm.

At a high level, our protocol for $\mathtt{i} = [n, n_0, \sigma]$ 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.

[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 [16]. Our current design choice allows us to simplify the presentation.

- 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. 1. The leaf circuit $\mathcal{C}_i$.

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 [9] 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 1 and the internal circuit $\mathcal{D}$ is shown in Figure 2. 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 result of the multiplication is 1.

Updates. 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. We present our final protocol in Figure 3.

Theorem 1. *The protocol of Figure 3 achieves the following complexities.*

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

- 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. 2. The internal circuit $\mathcal{D}$ with hard-coded $\mathsf{vk}_L, \mathsf{vk}_R$.

– $\mathcal{G}(1^\lambda, [n, n_0, \sigma]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - 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. 3. Our dynamic zk-SNARKs from recursive zk-SNARKs.

Proof. Clearly, the performance of our dynamic zk-SNARK depends on the choice of the underlying zk-SNARK. Assuming that, for a circuit C the zk-SNARK S has $O(|C|)$ setup time, $O(|C|)$ prover time, and verification time and proof size poly-logarithmic in the size of C, we can state the following. The runtime of $\mathcal{G}$ and $\mathcal{P}$ is $O(n)$ as each of the circuits in our recursion is of constant size. The verification time and proof size after recursion are $O(n_0)$ and $O(1)$, respec-

tively. Finally, for each affected recursion tree leaf after an update, computing the new proofs along the recursion tree path takes time $O(\log n)$, including the time to compute the new h_i values. If k is the Hamming distance between $\mathbb{w};\mathbb{x}$ and $\mathbb{w}';\mathbb{x}'$, then this takes time $O(k \log n)$ for all affected leafs. Note that, if necessary, the new h_{pub} can be computed via $O(k)$ field operations. □

Heuristic security of our construction. Since our protocol requires instantiating the hash function H that models a random oracle inside the evaluated circuits, we can only heuristically argue about its security as follows. (We note that this a common issue of several prior works on efficient recursive proof composition, e.g. [8,14,16]). Knowledge soundness follows from the same property of the underlying zk-SNARK. Given π_{root}, we can recursively extract accepting witnesses for each of the $\mathcal{D}$ nodes and eventually the $\mathcal{C}_i$ leaf nodes. This ensures the extracted witnesses form a $[6n + n_0]$ size array that satisfies the gate constraints. Modeling H as a random oracle, the probability the extracted witnesses contain differing pre-images for the same index i, including the $[n_0]$ input indices evaluated by the verifier, is negligible. Otherwise, the extracted witnesses array also satisfies the copy constraints, hence $(\mathbb{x}, \mathbb{w}) \in \mathbb{i}_{\mathcal{C}}$. Zero-knowledge is derived by the zero-knowledge of the underlying zk-SNARK, since the only information received by the verifier is π_{root} which can be independently simulated after each update.

Similarities with Mangrove [16]. We note that a similar approach was used in [16] 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 choose to instantiate their H as a universal hash function but to make their protocol secure its parameters must be 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, the Merkle proofs for all n circuits $\mathcal{C}_i$ have to be recomputed, 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 [11]) require at least linear time to handle a small update in the witness. The reason for that is mostly due to the use of Fiat-Shamir, where the randomness used depends on all the entries of the witness vector, 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 $(\emptyset, \mathbf{z}) \in \mathtt{i}_{\mathcal{P}} = [m, \sigma]$ 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 a vector $\mathbf{z}$ and provide a proof that, for a 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 [11]) 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 [11] 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

$$p(X, Y) - t(X, Y) - v(X, Y)$$

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)

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 [11] 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 C. Let $Z(X,W,Y)$, $V(X,W,Y)$, $P(X,W,Y)$, $T(X,W,Y)$ 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 $\mathtt{i}_{\mathcal{P}} = [m, \sigma]$ 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 D.

– $\mathcal{G}(1^\lambda, [m, \sigma]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - $\mathsf{pp}_{\mathsf{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$;
  - 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 2.
  - 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 2, 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 2, 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. 4. The Dynamo SNARK.

Theorem 2. *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 2 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 4. We summarize our protocol in the following theorem.

Theorem 3 (Dynamo). *The protocol of Figure 4 is a dynamic SNARK (per Definition 2) for $\mathrm{i}_{\mathcal{P}} = [m, \sigma]$ 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 $\mathbb{w}[i] = \widetilde{z}(\omega^i),\ \forall i \in [m]$.

Since verification accepts, all checks in the $\mathcal{V}$ algorithm of Figure 4 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 exactly knows the secrets a, b and c, it can compute everything from scratch 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 going to be an issue for our final protocol, since as we will see we apply the permutation argument in buckets.) The proof of the following lemma can be found in Appendix D.

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 [11,20], 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 D.

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

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

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

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

   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 9 in the Appendix. Zero-knowledge and universality follow exactly in the same way as in Dynamo.

Theorem 4 (Dynamix). *The protocol of Figure 9 is a dynamic SNARK (per Definition 2) for* $\mathtt{i}_{\mathcal{D}} = [m, N, \mathbf{s}, \mathbf{t}]$ *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 3. The main difference is the runtime of $\mathcal{G}$. In particular, 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 combination of proof of Lemma 1 and proof of Theorem 4. Finally, by following 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 $\mathtt{i} = [n, n_0, \sigma]$) 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, $O(k \cdot \sqrt{n})$ 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 $\mathtt{i}_{\mathcal{C}} = [n, n_0, \sigma]$ 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 $(\mathbb{x}, \mathbb{w}) \in \mathtt{i}_{\mathcal{C}}$ if and only if the following hold:

- *Copy constraint:* $(\emptyset, \mathbf{z}) \in \mathtt{i}_{\mathcal{P}} = [6n + n_0, \sigma]$.
- *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 [11]. 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 contraint 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

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

– $\mathcal{G}(1^\lambda, [n, n_0, \sigma]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - $\mathsf{pp}_{\mathsf{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$.
  - 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

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

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

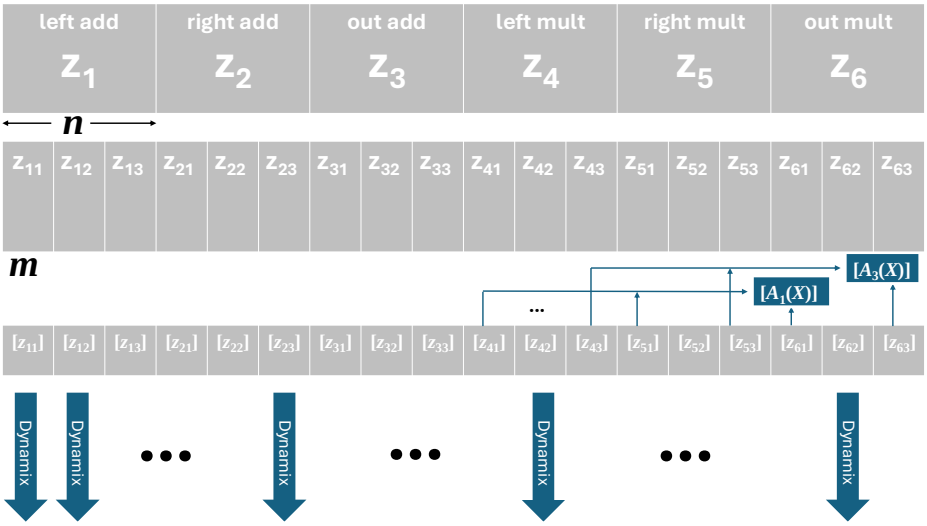

Fig. 6. The Dynaverse dynamic SNARK. The initial $6n$-sized witness $[\mathbf{z}_1 \dots \mathbf{z}_6]$ is split into subvectors $\mathbf{z}_{ij}$ of size $m = \sqrt{n}$, which are KZG-committed to $[z_{ij}]$. For each $[z_{ij}]$, we provide a Dynamix proof with respect to the permutation σ. For every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

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

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

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

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

Then the prover will output the proof that is output by a Dynamix argument 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.,

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

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. A pictorial description of Dynaverse is shown in Figure 6. Our complete protocol is shown in Figure 5.

Theorem 5 (Dynaverse). *The protocol of Figure 5 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 and outputs* pk *of $O(n)$ size and* 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 universal Dynaverse follows the same techniques as before. The proof for zero-knowledge is presented in Appendix D.

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 concetely improve the proof size we will run Dynamix on $O(1)$ number of length-$O(n)$ sub-vectors while still remaining $O(\sqrt{n} \log n)$ update time for gate constraints. For that, we need to address the following problems.

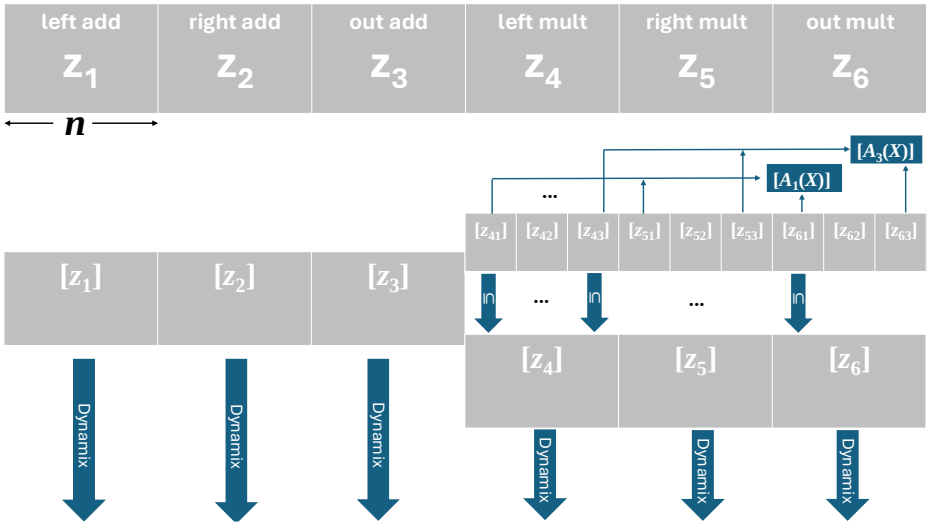

Fig. 7. The optimized Dynaverse dynamic SNARK. The addition wires are committed with three commitments $[z_1]$, $[z_2]$ and $[z_3]$ of n-sized vectors. The multiplication wires are committed in two ways, i.e., first with three commitments $[z_4]$, $[z_5]$ and $[z_6]$ of n-sized vectors and then with $3m$ 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 $i = 4, 5, 6$.

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 7. 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 addition 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 require further partition to achieve sublinear update 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 [25]): 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 [25]. 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 to $O(\log n)$. Bünz et al. [2] 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 B. 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.

Lemma 7 (Dynaverse with IPA). *There exists a variant of* Dynaverse *whose proof size and verification time is $O(\log n)$. All other complexities are the same.*

6 Evaluation

In this section, we measure the performance of the optimized Dynaverse. We do not consider the reduction of proof size to $O(\log n)$ using IPA because it is not concretely efficient. Our baseline is the recursive approach from Section 3.1. *Again, recall that the baseline approach cannot be proven formally secure, due to the fact it requires a random oracle be encoded in the SNARK circuit.*

Implementation details. We implement both constructions in Rust (it is not trivial to adapt the implementation of [16] into a dynamic SNARK in practice) and use Rayon [19] for parallelism. For our optimized Dynaverse, we use the BLS12-381 implementation [27] for bilinear pairings. We implement the baseline using the Plonky2 proof system [23], which is a Goldilocks field based recursive proof system. However, Dynaverse is based on BLS12-318 elliptic curve which is slower than Goldilocks field based operations due to the large size of the prime modulus. This difference in fields negatively affects the performance of Dynaverse. Previous works have noted that implementing incremental multiset hashing requires the hash function output to be several thousand bits long [15]. To overcome this limitation, we adopt the elliptic curve-based incremental multiset hashing approach proposed by Maitin-Shepard et al. [15]. Specifically, we use the EcGFp5 curve [18], based on the GF(p^5) extension of the Goldilocks field, and apply the Poseidon hash within our circuits.

Hardware. Experiments are executed 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 possible.

Experimental setting. Based on our Plonkish arithmetization with index $\mathtt{i} = [n, n_0, \sigma]$ (see Section 2), we run experiments for random circuits with size parameter n from 2^{18} to 2^{24}. More specifically, we first run $\mathcal{G}$ and $\mathcal{P}$ to get the initial proof, and then pick a random location of the witness to modify and update the proof. Although for many valid updates in practice, the Hamming distance between $\mathbf{z}, \mathbf{z}' \in \mathbb{F}^{6n+n_0}$ is usually some $k > 1$, we find that both protocols are "almost" fully parallelizable: For a set of k updates, the recursive approach can start at most k threads at the same time to update the recursive tree on k paths from leaves to root, while ours can also use k threads, each for one update. Therefore, to simplify the experimental settings, we only pick one random location to update and compare the metrics.

Results and comparison. We list the experimental results in Table 3. Our update time is 10× to 20× faster than the baseline while still keeping the verification time and proof size relatively small.

- Comparing to the baseline with $O(\log n)$ update time, our protocol with $O(\sqrt{n} \log n)$ update time is concretely faster because of the small constants (optimization in Fig. 7) and the inefficiency of SNARK recursion.
- The baseline has much faster verification time because it only needs to go through the public input and verify a constant-size SNARK proof.

Table 3. Comparison of one random update between baseline and Dynaverse.

	Baseline (Section 3.1)			Dynaverse		
$L = \log_2 n$	Update (s)	Verify (s)	Proof (KiB)	Update (s)	Verify (s)	Proof (KiB)
18	3.82	≤0.006	129.8	0.15	0.78	96
20	4.53	≤0.006	129.8	0.18	1.03	192
22	4.63	≤0.006	129.8	0.24	1.58	384
24	5.09	≤0.006	129.8	0.38	2.40	768

References

1. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. Algorithmica **79**(4), 1102–1160 (2017)
2. 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. Springer International Publishing, Cham (2021)
3. 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). https://doi.org/10.1109/SP.2018.00020
4. 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)
5. 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. Lecture Notes in Computer Science (2022)
6. 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)
7. 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. Springer Nature Switzerland, Cham (2023)
8. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12105, pp. 769–793. Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_27, https://doi.org/10.1007/978-3-030-45721-1_27
9. 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. Lecture Notes in Computer Science, vol. 2894, pp. 188–207. Springer (2003). https://doi.org/10.1007/978-3-540-40061-5_12

10. 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. Springer International Publishing, Cham (2018)
11. 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), https://eprint.iacr.org/2019/953, https://eprint.iacr.org/2019/953
12. 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)
13. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-Size Commitments to Polynomials and Their Applications. In: ASIACRYPT'10 (2010)
14. 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. Lecture Notes in Computer Science, vol. 13510, pp. 359–388. Springer (2022). https://doi.org/10.1007/978-3-031-15985-5_13, https://doi.org/10.1007/978-3-031-15985-5_13
15. Maitin-Shepard, J., Tibouchi, M., Aranha, D.F.: Elliptic curve multiset hash. CoRR **abs/1601.06502** (2016), http://arxiv.org/abs/1601.06502
16. 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. Springer Nature Switzerland, Cham (2024)
17. 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. CCS '24 (2024)
18. Pornin, T.: EcGFp5: a specialized elliptic curve. Cryptology ePrint Archive, Paper 2022/274 (2022), https://eprint.iacr.org/2022/274
19. Rayon-rs: Rayon, https://docs.rs/rayon/latest/rayon
20. Sefranek, M.: How (not) to simulate PLONK. Cryptology ePrint Archive, Paper 2024/848 (2024), https://eprint.iacr.org/2024/848
21. 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). USENIX Association, Boston, MA (Aug 2022)
22. Tamassia, R.: Authenticated data structures. In: Di Battista, G., Zwick, U. (eds.) Algorithms - ESA 2003. pp. 2–5. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
23. 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

24. 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. Lecture Notes in Computer Science (2008)
25. Wang, W., Ulichney, A., Papamanthou, C.: BalanceProofs: Maintainable vector commitments with fast aggregation. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 4409–4426. USENIX Association, Anaheim, CA (Aug 2023)
26. 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)
27. zkcrypto: bls12-381 implementation, https://github.com/zkcrypto/bls12_381

A Auxiliary lemmas

Lemma 8. *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:

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

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

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

□

The following lemma is Lemma 1 from [20] that is helpful to prove the zero-knowledge property. We refer to [20] to see its formal proof.

Lemma 9 ([20]). *Let $S \subset \mathbb{F}$ and $Z_S(X) := \prod_{a\in S}(X-a)$. Fix a polynomial $f \in \mathbb{F}[X]$ and any distinct values $x_1,\ldots,x_k \in \mathbb{F}\setminus S$. Then the following distribution is uniform in $\mathbb{F}^k$:*

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

B Inner Product Arguments

Bünz et al. [2] 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:

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

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 [2]), and the proof size is $|\pi| = O(\log m)$.

C Algebraic Group Model

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

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

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

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

so that the verifier checks if

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

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

Lemma 10. *For any PPT algebraic adversary $\mathcal{A}$, given $\mathsf{pp}_{\mathsf{bl}}\leftarrow\mathcal{G}_{\mathsf{bl}}(1^\lambda)$ and $\mathbf{L}=\{g^{h_i(\alpha_1,\ldots,\alpha_s)}\}_h$ as the initial list ($h_i(X_1,\ldots,X_s)$ are some pre-defined public polynomials), the following probability is negligible under q-DLOG assumption:*

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

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

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

1. Pick $r_1,\ldots,r_s\xleftarrow{\$}\mathbb{F}$. Let $\alpha_1:=r_1\tau,\ \ldots,\ \alpha_s:=r_s\tau$. Compute
$$\mathbf{L}=\left\{g^{h_i(\alpha_1,\ldots,\alpha_s)}=g^{h_i(r_1\tau,\ldots,r_s\tau)}\right\}_h$$
and sends $\mathsf{pp}_{\mathsf{bl}}$ and $\mathbf{L}$ to $\mathcal{A}$.
2. Receive $\{C_{l,i}\}_{i\in[t],l\in\{1,2\}}$ from $\mathcal{A}$. Note that since $\mathcal{A}$ is algebraic, $\mathcal{A}$ should also outputs vectors to show how each group element in $(C_{1,i},C_{2,i})_{i\in[t]}$ can be computed from $\mathbf{L}$. Thus $\mathcal{A}^*$ can reconstruct $f_{l,i}(X_1,\ldots,X_s)$ such that
$$C_{l,i}=[f_{l,i}(X_1,\ldots,X_s)],\quad \forall i\in[t],l\in\{1,2\}.$$

3. If the following holds:

$$\sum_{i\in[t]} f_{1,i}(X_1,\ldots,X_s)\cdot f_{2,i}(X_1,\ldots,X_s) \not\equiv 0 \wedge$$
$$\prod_{i\in[t]} e([f_{1,i}(X_1,\ldots,X_s)],[f_{2,i}(X_1,\ldots,X_s)]) = 1\,,$$

then $\mathcal{A}^*$ knows that $\sum_{i\in[t]} f_{1,i}(X_1,\ldots,X_s)\cdot f_{2,i}(X_1,\ldots,X_s)$ is a non-zero polynomial which evaluates 0 on $(\alpha_1,\ldots,\alpha_s)$. According to Lemma 8, $g(X) := \sum_{i\in[t]} f_{1,i}(r_1X,\ldots,r_sX)\cdot f_{2,i}(r_1X,\ldots,r_sX)$ 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 11. *For any PPT algebraic adversary $\mathcal{A}$, given $\mathsf{pp}_{\mathsf{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$ and $\mathbf{L} = \{g_l^{h_i(\alpha_1,\ldots,\alpha_s)}\}_h$ as the initial list ($h_i(X_1,\ldots,X_s)$ are pre-defined public polynomials) where $d_1,\ldots,d_s$ are the maximum degree of $X_1,\ldots,X_s$ among $h_i(X_1,\ldots,X_s)$, the following probability is negligible under q-DLOG assumption:*

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

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:

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

1. Pick $r_1, \ldots, r_s \xleftarrow{\$} \mathbb{F}$. Let $\alpha_1 := r_1\tau, \ \ldots, \ \alpha_s := r_s\tau$. Compute

$$\mathbf{L} = \left\{ g_l^{h_{l,i}(\alpha_1,\ldots,\alpha_s)} = g_l^{h_{l,i}(r_1\tau,\ldots,r_s\tau)} \right\}_{l\in\{1,2\},h}$$

   and sends $\mathsf{pp}_{\mathsf{bl}}$ and $\mathbf{L}$ to $\mathcal{A}$.
2. Receive (C, C') from $\mathcal{A}$. Note that since $\mathcal{A}$ is algebraic, $\mathcal{A}$ should also outputs vectors to show how (C, C') can be computed from $\mathbf{L}$. Thus $\mathcal{A}^*$ can reconstruct $f(X_1, \ldots, X_s), f'(X_1, \ldots, X_s)$ such that

$$C = [f(X_1, \ldots, X_s)], C' = [f'(X_1, \ldots, X_s)].$$

   Note that $\mathsf{deg}_{X_1} f(X_1, \ldots, X_s) \le d_1, \ \mathsf{deg}_{X_1} f'(X_1, \ldots, X_s) \le d_1$.
3. If the following holds:

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

   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 8, 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. □

D Other proofs

D.1 Proof of Theorem 2.

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

- $z_i(X) = L_i(X)$;
- $v_i(X, Y) = \mathbf{y}[i]L_i(X)$;
- $p_i(X, Y) = \mathbf{y}[i] \sum_{j=i}^{m} L_j(X)$;
- $t_i(X, Y) = \mathbf{y}[i] \sum_{j=i}^{m} L_j(X \cdot \omega^{-1})$;
- $g_i(W, Y) = \mathbf{y}[i] \sum_{j=i}^{m} L_j(W)$;
- $Z_i(X, W, Y) = Y^m \cdot W^m \cdot L_i(X)$;
- $V_i(X, W, Y) = W^m \cdot \mathbf{y}[i]L_i(X)$;

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

□

D.2 Proof of Lemma 1

We present the universal protocol in Fig. 8. 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,

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

  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 $[\delta_i]$.

– $\mathcal{G}(1^\lambda, m) \to (\mathsf{pp})$:
  - $\mathsf{pp}_{\mathsf{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$;
  - 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 2.
  - 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. 8. The universal Dynamo SNARK. $\mathcal{P}$, $\mathcal{U}$, $\mathcal{V}$ are the same as Fig. 4.

D.3 Proof of Lemma 2

We introduce how to add zero-knowledge for Dynamo here. Following the idea of [11,20], 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:

$\mathcal{S}(1^\lambda, \mathrm{i}) \to (t, \mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
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 9, $z^{\mathsf{zk}}(a) = z(a) + \rho_z(a^m - 1)$ 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 9, $v^{\mathsf{zk}}(a, b) = v(a, b) + \rho_v(a^m - 1)$ 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 9 and the following calculation,

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

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

D.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).

E The Dynamix detailed protocol

– $\mathcal{G}(1^\lambda, [m, N, \mathbf{s}, \mathbf{t}]) \to (\mathsf{pk}, \mathsf{upk}, \mathsf{vk})$:
  - $\mathsf{pp}_{\mathsf{bl}} \leftarrow \mathcal{G}_{\mathsf{bl}}(1^\lambda)$.
  - Let $\mathcal{F} = \{z, v, p, t, g, h, Z, V, P, T, G, H, \alpha, \beta, \gamma, \delta, \varepsilon\}$ be the set of polynomials from Theorem 2, 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 2, 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. 9. The Dynamix SNARK.