

Dynasaurs: Efficient Universal Dynamic zkSNARKs from Sparse Linear Arguments

Martí Batista¹, Álvaro Montes², Nikitas Paslis², and Carla Ràfols²

¹ Universitat de Barcelona,
martibatista03@gmail.com

² Universitat Pompeu Fabra,
{alvaro.montes, nikitas.paslis, carla.rafols}@upf.edu

Abstract. Dynamic zkSNARKs were recently introduced by Wang et al. [Eurocrypt, 2026]. This primitive extends standard zkSNARKs with an *update* algorithm that adapts a proof to a new statement in time sublinear in the circuit size, provided the witness changes in few positions. However, existing constructions either need a circuit-specific setup or, in the universal case, send over 130 group elements and require over 180 pairings.

As is the case for universal zkSNARKs, dynamic ones can be built from dynamic arguments for Hadamard products and linear relations. Wang et al. handle the latter in the particular case of a permutation matrix, via a *sparse* argument—a protocol whose prover runs in time proportional to the Hamming weight of the witness. Nevertheless, their techniques do not directly extend to the arbitrary matrices arising in constraint systems such as R1CS or CCS. Furthermore, the standard approach for proving general linear relations is unsuitable for the sparse setting because of a *witness-independent* step: the prover commits to an auxiliary polynomial determined by the matrices alone, and is hence dense regardless of how sparse the witness might be.

Our first contribution is a *sparse zkSNARK for linear relations*, which we build from a fully witness-dependent argument together with what we call a *rational encoding* of the matrices. As our second contribution, we develop a compiler that turns *any* sparse argument for a linear relation into a dynamic one, while preserving the zero-knowledge property of the underlying scheme.

Instantiated for Plonk and R1CS-lite, our techniques yield universal dynamic zkSNARKs with at most 20 group elements per proof and 23 verifier pairings—over $6.5\times$ and $7.8\times$ fewer than the state of the art—as well as asymptotically faster updates. We also show how both of our constructions can be de-amortized.

Keywords: zero-knowledge · succinct arguments · dynamic zkSNARKs · sparse arguments

1 Introduction

Zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) are central to modern cryptography, enabling scalable blockchain architectures and privacy-preserving protocols. Despite a rich variety of constructions [Gro16, GWC19, CHM⁺20, Set20, GLS⁺23, NS24, ACFY25] built from different techniques and assumptions, nearly all zkSNARKs are strictly static: updating a proof π for an instance-witness pair (x, w) to a proof π' for a slightly modified pair (x', w') requires re-executing the full proving algorithm from scratch—even if (x, w) and (x', w') differ by just one bit.

Dynamic zkSNARKs, introduced by Wang, Papamanthou, Srinivasan and Papadopoulos [WPSP26b], address this inefficiency. They extend standard zkSNARKs with an *Update* algorithm that produces an updated proof π' in time *sublinear* in the statement size, provided the Hamming distance between (x, w) and (x', w') is small. The primitive is useful whenever the same statement is proved repeatedly while its witness changes only locally, as when a large data set is certified to satisfy some property and a few of its records are subsequently edited. Dynamic zkSNARKs also imply other cryptographic primitives, such as *sparse* zkSNARKs—where proving time scales with the Hamming weight of the witness—and bounded incremental verifiable computation without recursion [WPSP26b].

Limitations of Existing Constructions. Existing dynamic zkSNARKs fall into two categories, and suffer from a sharp trade-off between universality and concrete efficiency.

1. **Circuit-specific constructions.** Dynark [ZOZ26] and Delphus [WPSP26a] build on top of Groth16 [Gro16]. While concretely efficient, they inherit Groth16’s main drawback: reliance on a non-universal, circuit-specific trusted setup. Delphus is moreover restricted to the designated-verifier setting.
2. **Universal constructions.** The protocols of [WPSP26b] support universal setups, but at a severe cost: in the most efficient variant the verifier requires over 180 pairings and the prover commits to more than 130 polynomials, despite achieving asymptotically constant proof size and verification time. Furthermore, these numbers are lower bounds: [WPSP26b] focus on the non-zero-knowledge, non-universal variants of their protocols, only sketching universality and zero-knowledge. Moreover, they present their construction using *symmetric pairings*, for which no efficient instantiation is known at standard security levels. Finally, their constructions are tied to a restricted variant of the Plonk constraint system that lacks support for basic operations such as multiplication by public constants.

This leads to our first research question.

Can we construct universal dynamic zkSNARKs that are concretely efficient and support standard constraint systems?

The core building block of [WPSP26b] is a *sparse permutation argument*: given a permutation σ , it proves that $a_\ell = a_{\sigma(\ell)}$ for all ℓ in time proportional to the Hamming weight of $\mathbf{a}$. Permutation arguments are what Plonk-style systems need, which is why the constructions of [WPSP26b] are confined to that family. The other widely used constraint systems, such as R1CS-lite [CFF⁺21], instead rely on a *lincheck* protocol proving a matrix-vector relation $M\mathbf{a} = \mathbf{b}$. Supporting them would therefore require a sparse lincheck, but the techniques of [WPSP26b] are specific to permutations. This motivates our second research question.

Can we construct sparse zkSNARKs for linear relations?

1.1 Our Contributions

Our answer to the second question is affirmative for a class of matrices that we argue is essentially the largest possible, and it yields an affirmative answer to the first as well. Our contributions are summarized as follows.

- **Sparse zkSNARKs for linear relations.** Our central contribution is a sparse zkSNARK for relations $M^{(j)}\mathbf{a} = \mathbf{b}^{(j)}$, $j \in [l]$, where each matrix has at most $V^{(j)}$ non-zero entries per column. This restriction on the matrix structure is natural, and a similar assumption was made in [ZOZ26]: if a column of $M^{(j)}$ is dense, $\mathbf{b}^{(j)}$ can easily result in a dense vector even when $\mathbf{a}$ is highly sparse. We build this protocol by replacing standard matrix-dependent dense lincheck polynomials with *witness-dependent* auxiliary polynomials that preserve the sparsity.
- **A compiler from sparse to dynamic, with zero-knowledge.** We give a compiler turning any sparse zkSNARK for a linear relation into a dynamic one. The natural way of updating a proof exploits linearity: the difference of the original and the updated statement is again a valid statement of the relation, which can be proven with the sparse zkSNARK. However, a naive application does not result in zero-knowledge: it is possible to distinguish between fresh and updated proofs. Our solution again exploits linearity and consists of making the prover interpret its statement as an *update to itself*. This way, the distribution of fresh and updated proofs will be the same by construction.
- **Concretely efficient universal dynamic zkSNARKs.** Combining the above with the dynamic argument for Hadamard products of [ZOZ26], we obtain universal dynamic zkSNARKs for the standard Plonk constraint system [GWC19] and for a variant of R1CS-lite for circuits of bounded fan-out³ [RZ21]. Compared to [WPSP26b] (see Table 1), the prover sends 20 commitments instead of more than 130 and the verifier performs 23 pairings instead of more than 180, improvements of $6.5\times$ and $7.8\times$; we also improve the update complexity asymptotically in both the semi-dynamic and the fully-dynamic settings, and achieve the latter in the *de-amortized* setting.

³ Note that circuits of arbitrary fan-out can be represented in this constraint system by adding identity gates to duplicate wire values, at the cost of an increase in the circuit size.

1.2 Related Work

To the best of our knowledge, the only works on general purpose dynamic zkSNARKs are [WPSP26b], [ZOZ26] and [WPSP26a], listed in Table 3 with their complexities. The dynamic zkSNARK of [ZOZ26] is fully compatible with Groth16: it shares the same trusted setup, prover and verifier. The one of [WPSP26a] diverges from Groth16 and requires a designated verifier.

Wang et al. [WPSP26b] present two⁴ universal dynamic zkSNARKs: Dynalog and Dynaverse. Dynalog relies on a hierarchical data structure to achieve polylogarithmic update times, but its proof size and verifier time are polylogarithmic as well. Next we compare in detail with Dynaverse, which remains the most directly comparable universal dynamic zkSNARK to our work.

Comparison with Dynaverse [WPSP26b]. Dynaverse uses a variant of the Plonk constraint system. It encodes an arithmetic circuit with n addition and n multiplication gates using a redundant trace $\mathbf{z} = (s_1, \dots, s_6) \in \mathbb{F}^{6n}$, and circuit satisfiability requires four checks: (i) correct wiring via a permutation σ , so that $z_i = z_{\sigma(i)}$; (ii) addition gate validity $s_{1,i} + s_{2,i} = s_{3,i}$; (iii) multiplication gate validity $s_{4,i}s_{5,i} = s_{6,i}$; and (iv) consistency checks ensuring that each $\mathbf{s}_t$ is the correct subvector of $\mathbf{z}$. We highlight the differences with our constructions and give the full comparison of our schemes against it in Table 1.

- **Constraint system flexibility.** The constraint system used by Dynaverse lacks support for basic operations such as addition or multiplication by public constants, so these must be encoded explicitly as part of the instance. This inflates the witness length and degrades efficiency. Using the standard Plonk and R1CS-lite constraint systems, we avoid this overhead entirely.
- **Linear constraints.** To prove (i), Dynaverse relies on Dynamo, a sparse permutation argument. We extend its underlying idea to a much broader class of linear relations, and in doing so also handle several matrices at once.
- **Non-linear constraints.** For (iii), Dynaverse uses an “anchor” technique: the updater shows that the multiplication gates hold in the updated coordinates only, and pairs this with a static anchor proof. We instead adapt the protocol of [ZOZ26], which is simpler, achieves an optimal proof size of a single group element, and improves the asymptotics of updates in the fully-dynamic setting.
- **Expensive consistency checks.** In Dynaverse the consistency checks (iv) account for roughly two thirds of both the proof size and the verifier’s pairings. They appear inherent to Plonk-style systems: the sparse permutation argument operates on the full extended witness,⁵ whereas the gate constraints apply locally to the individual wire vectors. Our Plonk construction faces the same structural problem but resolves it differently: instead of enforcing subvector correctness with a dedicated and expensive protocol, we express the relation between the extended witness and the wire vectors as cyclic shifts—which are particularly structured linear maps—and our linear argument batches them together with the permutation into a single check. The consistency checks thus cost nothing in proof size or verifier pairings. Our R1CS-lite construction avoids them altogether, by the nature of the constraint system.
- **De-amortization.** Because Dynaverse relies on the anchor technique, its update complexities hold only in an amortized sense: proofs can be updated efficiently as long as the new witness stays close to the anchor witness, and once that distance is exceeded it becomes cheaper to recompute the proof from scratch and amortize the cost over the previous updates. Our dynamic linear arguments use a similar anchor technique, but we de-amortize them following [ZOZ26].

Finally, a smaller proof does not in general mean a faster prover—multilinear zkSNARKs often outperform univariate ones despite larger proofs. In this setting, however, the two go together: updating a proof reduces essentially to computing and sending KZG commitments [KZG10] to sparse polynomials, so fewer commitments directly means faster updates.

1.3 Technical Overview

Following the framework of Ràfols and Zapico [RZ21], constructing a universal zkSNARK reduces to two independent building blocks: an argument for non-linear relations, such as Hadamard products, and an

⁴ They also present another dynamic non-interactive argument, but it is not succinct.

⁵ This is not the case in standard Plonk [GWC19], where the permutation argument works directly on the left, right and output wire vectors; that permutation argument, however, is not sparse.

<i>Constraint System</i>	Dynaverse [WPSP26b]	THIS WORK	
	Plonk variant	Plonk	V-R1CS-lite
<i>Asymptotic costs</i>			
Prove	$n \log n$	$n \log n$	$n \log n$
BuildData	$n \log n$	$n \log n$	$n \log n$
Update (semi-dyn.)	$k \log n + k \log^2 k$	$\log n + k \log^2 k$	$\log n + kV \log^2(kV)$
Update (fully-dyn.)	$k\sqrt{n \log n} + \sqrt{n \log n} \log^2 n$	$k\sqrt{n \log n} + k \log^2 k$	$kV\sqrt{n \log n} + kV \log^2(kV)$
Fully-dyn. regime	amortized	de-amortized	de-amortized
<i>Concrete costs</i>			
Proof size	$131 \mathbb{G}_1$	$19 \mathbb{G}_1 + \mathbb{G}_2 + 2 \mathbb{F}$	$18 \mathbb{G}_1 + \mathbb{G}_2 + 2 \mathbb{F}$
Verify: pairings	180	23 (13 batched)	19 (10 batched)
Verify: other ops	$ \mathbf{x} + \log n$	$ \mathbf{x} + \log n$	$ \mathbf{x} + \log n + V$

Table 1. Comparison with Dynaverse [WPSP26b]. O notation throughout; k is the Hamming distance between the original and the updated witness, and V the fan-out of the circuit. All three schemes are universal with constant proof size and number of pairings. The Dynaverse costs are lower bounds, taken from its non-universal, non-zero-knowledge, symmetric-pairing variant (Section B); batched counts are from Section D.

argument for linear relations. Since a Groth16 proof is essentially a Hadamard product proof, for the first block we adapt the dynamic zkSNARK of [ZOZ26]. The missing piece is an efficient *dynamic argument for linear relations*, which we obtain by first designing a sparse argument and then applying our compiler. We describe the ideas for a single lincheck; our actual linear relations involve several matrices at once, which we address at the end of this overview.

The standard lincheck. Let $\mathbb{H} = \{h_1, \dots, h_n\}$ be a multiplicative subgroup of order n , let $\nu_{\mathbb{H}}(X)$ be its vanishing polynomial and let $\boldsymbol{\lambda}(X)$ be the vector of Lagrange polynomials $(\lambda_1(X), \dots, \lambda_n(X))$ over $\mathbb{H}$. Let $M(X, Y) = \boldsymbol{\lambda}(Y)^\top M \boldsymbol{\lambda}(X)$ be the Lagrange encoding of a matrix $M \in \mathbb{F}^{n \times n}$. Given $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$, the standard lincheck reduces $M\mathbf{a} = \mathbf{b}$ to the single inner-product relation $(\mathbf{r}^\top M)\mathbf{a} = \mathbf{r}^\top \mathbf{b}$, where $\mathbf{r} = \boldsymbol{\lambda}(r)$ for $r \leftarrow \mathbb{F}$. Writing $a(X), b(X)$ for the low-degree extensions of $\mathbf{a}, \mathbf{b}$ and setting $t(X) = M(X, r)$, the protocol decomposes into two tasks: (i) assuming $t(X)$ is correctly defined, prove that $\sum_{\ell} t(h_{\ell})a(h_{\ell}) = \sum_{\ell} \lambda_{\ell}(r)b(h_{\ell})$, which the univariate sumcheck does; and (ii) prove that $t(X) = M(X, r)$.

The problem: witness independence. The polynomial $t(X) = M(X, r)$ depends only on M and is therefore dense in general, even when the witness is sparse. Committing to it alone makes the prover linear in n , which rules out this approach in the sparse setting.

Our sparse lincheck. To exploit sparsity we make the auxiliary polynomials depend on the witness. Rewriting $(\mathbf{r}^\top M)\mathbf{a} = \mathbf{r}^\top \mathbf{b}$ coordinate-wise as

$$\sum_{\ell=1}^n (\mathbf{r}^\top M)_{\ell} a_{\ell} = \sum_{\ell=1}^n r_{\ell} b_{\ell},$$

suggests replacing $t(X)$ by the two auxiliary vectors $\tilde{\mathbf{p}} = (\mathbf{r}^\top M) \circ \mathbf{a}$ and $\mathbf{p} = \mathbf{r} \circ \mathbf{b}$. Every coordinate of these is multiplied by a coordinate of the witness, so both inherit its sparsity. Let $p(X)$ and $\tilde{p}(X)$ be their low-degree extensions. The protocol again decomposes into two tasks: (i) assuming $p(X), \tilde{p}(X)$ are correctly defined, prove the compressed relation, now of the form $\sum_{\ell} \tilde{p}(h_{\ell}) = \sum_{\ell} p(h_{\ell})$, via the univariate sumcheck; and (ii) prove that $p(X)$ and $\tilde{p}(X)$ take the claimed values over $\mathbb{H}$. We treat the second task through two zero-checks over $\mathbb{H}$.

Correctness of $p(X)$. Since $\lambda_{\ell}(X) = \frac{h_{\ell}}{n} \frac{X^n - 1}{X - h_{\ell}}$, we have $p(h_{\ell}) = b(h_{\ell}) \frac{h_{\ell}}{n} \frac{r^n - 1}{r - h_{\ell}}$ for each ℓ . Clearing denominators, $p(X)$ is well formed if and only if there exists $q(X)$ with

$$n p(X)(r - X) - (r^n - 1) b(X) X = q(X) \nu_{\mathbb{H}}(X).$$

Correctness of $\tilde{p}(X)$. Here $\tilde{p}(\mathbf{h}_\ell) = a(\mathbf{h}_\ell)(\mathbf{r}^\top M)_\ell$, and

$$(\mathbf{r}^\top M)_\ell = \sum_{i: M_{i\ell} \neq 0} \frac{\mathbf{h}_i}{n} \frac{r^n - 1}{r - \mathbf{h}_i} M_{i\ell} = \frac{r^n - 1}{n} \sum_{i: M_{i\ell} \neq 0} \frac{\mathbf{h}_i}{r - \mathbf{h}_i} M_{i\ell}.$$

The key idea here stems from [RZ21], where it was used to prove well-formedness of $t(X) = M(X, r)$ in the standard lincheck. It consists of defining the rational function $C_\ell(Y) = \sum_{i: M_{i\ell} \neq 0} \frac{\mathbf{h}_i}{Y - \mathbf{h}_i} M_{i\ell}$ and expressing its evaluation at r as $C_\ell(r) = \hat{R}_r(\mathbf{h}_\ell) / \hat{I}_r(\mathbf{h}_\ell)$, where $\hat{R}_r(X)$ and $\hat{I}_r(X)$ are public polynomials, whose commitments can be computed efficiently by the verifier. Consequently $\tilde{p}(\mathbf{h}_\ell) = a(\mathbf{h}_\ell) \frac{r^n - 1}{n} \frac{\hat{R}_r(\mathbf{h}_\ell)}{\hat{I}_r(\mathbf{h}_\ell)}$, and clearing the denominator gives the zero-check

$$n \tilde{p}(X) \hat{I}_r(X) - (r^n - 1) a(X) \hat{R}_r(X) = \tilde{q}(X) \nu_{\mathbb{H}}(X).$$

Sparsity analysis. Sending a KZG commitment to $p(X)$ and $\tilde{p}(X)$ is cheap, because they are sparse in the Lagrange basis, but the same cannot be said for $q(X)$ and $\tilde{q}(X)$. However, we will see that $q(X)$ is sparse in the monomial basis and that the prover can commit to $\tilde{q}(X)$ in time proportional to the sparsity of $\mathbf{a}$ by preprocessing some public information.

Extension to several matrices and rational encodings. In the main body we work with several matrices at once, that is, $M^{(j)} \mathbf{a} = \mathbf{b}^{(j)}$ for $j \in [l]$. The protocol above extends to this setting at essentially no cost: a challenge $\delta \leftarrow \mathbb{F}$ compresses the l relations into the single relation $(\sum_j \delta^{j-1} M^{(j)}) \mathbf{a} = \sum_j \delta^{j-1} \mathbf{b}^{(j)}$. We also formalize the decomposition of the rational function $C_\ell(Y)$ above with the notion of *rational encoding*: specifying one is all that is needed to apply the protocol to a new family of matrices. In Section E we illustrate this by giving a high-level overview of how our techniques apply to CCS [STW23].

From sparse lincheck to dynamic lincheck. Given a sparse lincheck, we build a dynamic one as follows. Suppose the prover holds a valid proof π for $M\mathbf{a} = \mathbf{b}$ and wants a proof for $M\mathbf{a}' = \mathbf{b}'$. The updater forms the difference vectors $\mathbf{a}^* = \mathbf{a}' - \mathbf{a}$ and $\mathbf{b}^* = \mathbf{b}' - \mathbf{b}$ and produces a proof π^* for $M\mathbf{a}^* = \mathbf{b}^*$. The cost of π^* scales with the sparsity of $\mathbf{a}^*$, which is exactly the Hamming distance between $\mathbf{a}$ and $\mathbf{a}'$. The updated proof consists of the commitments to $\mathbf{a}^*$ and $\mathbf{b}^*$, the difference proof π^* , and the original proof π ; by linearity, a verifier accepting both π and π^* is convinced that $M\mathbf{a}' = \mathbf{b}'$. In the language of [ZOZ26] this is a *semi-dynamic* SNARK, one that can update proofs produced by **Prove** but not proofs produced by **Update**; a compiler from [ZOZ26] turns it into a *fully-dynamic* SNARK, i.e., one that can update proofs regardless of how they were generated.

The zero-knowledge problem, and how we solve it. The construction just described is not zero-knowledge, and the reason is structural rather than a matter of blinding: a proof produced by **Prove** is a single sparse proof, whereas a proof produced by **Update** additionally carries the difference commitments and a second sparse proof. The two are distinguishable at a glance, however zero-knowledge the underlying sparse argument may be.

Our solution is that the prover should never prove the statement it is given, but a re-randomisation of it. Concretely, **Prove** samples a random opening of the zero vector, subtracts it from the statement, and proves the resulting re-randomised statement; the difference between the two is then a commitment to the zero vector with uniformly random opening, which is distributed exactly like the difference commitments of a genuine update. Since a fresh proof is now the update of a statement to itself, fresh and updated proofs are produced by the same algorithm and are identical in shape by construction, and zero-knowledge of the compiled scheme follows from that of the sparse argument alone.

Organization. The remainder of the paper is organised as follows. We review the necessary background in Section 2. In Section 3 we present the compiler from sparse to dynamic zkSNARKs for linear relations. In Section 4 we present our sparse linear arguments. In Section 5 we introduce the dynamic SNARKs for the Hadamard product and the gate equation of Plonk. We conclude in Section 6 with our dynamic zkSNARKs for the Plonk and R1CS-lite constraint systems.

2 Preliminaries

Basic Notation. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \dots, n\}$. If $m \in \mathbb{N}$ and $m > n$, $[n, m]$ denotes the set $\{n, n+1, \dots, m-1, m\}$. For a set S , $|S|$ denotes its cardinality and $s \leftarrow S$ represents that s is an element sampled uniformly at random from S .

We denote by $\mathbb{F}$ a finite field, by $\mathbb{F}[X]$ the ring of polynomials in the variable X over $\mathbb{F}$, and by $\mathbb{F}^{\leq d}[X]$, the space of polynomials in $\mathbb{F}[X]$ of degree less or equal to d . Vectors will be denoted in boldface. For vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$, we denote their Hadamard product by $\mathbf{a} \circ \mathbf{b}$. The support of $\mathbf{a} \in \mathbb{F}^n$ is $\text{Supp}(\mathbf{a}) = \{\ell \in [n] \mid a_\ell \neq 0\}$, the set of indices of the non-zero entries of $\mathbf{a}$.

A *universal relation* (or *indexed relation*) $\mathcal{R}$ is a set of triples $(i, \mathbf{x}, \mathbf{w})$, where i is the *index*, $\mathbf{x}$ is the *instance* and $\mathbf{w}$ the *witness*. We may consider universal relations $\mathcal{R}$ that contain a single i . In this case we will directly write $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ and ignore i .

Algebraic Preliminaries. Given $f(X), g(X) \in \mathbb{F}[X]$, we write $\lfloor f(X)/g(X) \rfloor$ for the quotient of the Euclidean division of $f(X)$ by $g(X)$. We will repeatedly use the linearity of polynomial division: given $\alpha_i \in \mathbb{F}$ and $f_i(X) \in \mathbb{F}[X]$ for $i \in [m]$, we have

$$\sum_{i=1}^m \alpha_i \left\lfloor \frac{f_i(X)}{g(X)} \right\rfloor = \left\lfloor \frac{\sum_{i=1}^m \alpha_i f_i(X)}{g(X)} \right\rfloor.$$

Let $\mathbb{H} = \langle \omega \rangle \subseteq \mathbb{F}^\times$ be a subgroup of roots of unity of order n , and write $\mathbb{H} = \{\mathbf{h}_1, \dots, \mathbf{h}_n\}$, where $\mathbf{h}_\ell = \omega^{\ell-1}$ for every $\ell \in [n]$. We denote by $\nu_{\mathbb{H}}(X) = X^n - 1$ the vanishing polynomial over $\mathbb{H}$, and by $\lambda_\ell(X) = \frac{\mathbf{h}_\ell}{n} \frac{X^n - 1}{X - \mathbf{h}_\ell}$ the Lagrange polynomials. The randomized low-degree extension over $\mathbb{H}$ of a vector $\mathbf{a} \in \mathbb{F}^n$ is the polynomial $a(X) = \sum_{\ell=1}^n a_\ell \lambda_\ell(X) + \rho_a \nu_{\mathbb{H}}(X)$ for $\rho_a \leftarrow \mathbb{F}$.

Lemma 1 (Univariate sumcheck, [BCR⁺19]). *Let $p(X) \in \mathbb{F}[X]$. Then $\sum_{\ell=1}^n p(\mathbf{h}_\ell) = \sigma$ if and only if there exist polynomials $q(X), g(X) \in \mathbb{F}[X]$, with $\deg(g) < n-1$, such that*

$$p(X) = q(X)\nu_{\mathbb{H}}(X) + Xg(X) + \frac{\sigma}{n}.$$

Bilinear Groups. A *bilinear group* is a tuple $\mathbf{gk} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathcal{P}_1, \mathcal{P}_2)$, where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order q , $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a non-degenerate bilinear map, known as the *pairing*, and $\mathcal{P}_1, \mathcal{P}_2$ are generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively. Elements in $\mathbb{G}_\gamma$ are denoted implicitly as $[a]_\gamma = a\mathcal{P}_\gamma$, where $\gamma \in \{1, 2, T\}$ and $\mathcal{P}_T = e(\mathcal{P}_1, \mathcal{P}_2)$. Moreover, group operations and the pairing are efficiently computable, and there is no efficient algorithm to compute an isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

Cryptographic Assumptions. We analyze the security of our constructions in the *Algebraic Group Model* (AGM) of Fuchsbauer, Kiltz and Loss [FKL18]. This is an idealized model in which every adversary $\mathcal{A}$ is restricted to be *algebraic*, in the sense that every group element output by $\mathcal{A}$ is accompanied by a representation of that group element in terms of all the group elements that $\mathcal{A}$ has seen so far. We also use the following cryptographic assumption.

Definition 1 (((q_1, q_2) -PDL assumption). *For fixed $q_1, q_2 \in \mathbb{N}$, the (q_1, q_2) -PDL assumption holds if for every polynomial time adversary, the following probability is negligible in λ :*

$$\Pr \left[\mathbf{gk} \leftarrow \mathcal{G}(1^\lambda), \mathbf{x} \leftarrow \mathbb{F}^*, \mathbf{x}' \leftarrow \mathcal{A}(\mathbf{gk}, \{[x^i]_1\}_{i=1}^{q_1}, \{[x^i]_2\}_{i=1}^{q_2}) \mid \mathbf{x} = \mathbf{x}' \right].$$

KZG Commitments. A commitment scheme is a tuple of algorithms $\text{CS} = (\text{KeyGen}, \text{Com})$ where the first algorithm samples a commitment key ck and the second algorithm, upon input of the commitment key, a message m and opening material ρ , outputs a commitment c . We assume familiarity with the standard notions of *hiding* and *binding*. Next, we recall the celebrated KZG polynomial commitment scheme [KZG10].

Definition 2 (KZG). Let $\mathbf{gk} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathcal{P}_1, \mathcal{P}_2)$ be a bilinear group. The KZG polynomial commitment scheme for the message space $\mathbb{F}^{\leq n}[X]$ is made up of the following algorithms.

- **KeyGen**($1^\lambda, n$): on input $(1^\lambda, n)$, where n is the degree bound, sample $\tau \leftarrow \mathbb{F}$ and output

$$\mathbf{ck} = \left(\{[\tau^i]_1\}_{i \in [0, n]}, \{[\tau^i]_2\}_{i \in [0, n]} \right).$$

- **Com**($\mathbf{ck}, p$): on input the commitment key $\mathbf{ck}$, and a polynomial $p(X)$ of degree at most n , outputs a commitment $c = [p(\tau)]_1$. The opening ρ is empty and therefore omitted.

Although KZG is not hiding, it can be used to construct a hiding vector commitment scheme by committing to a randomized low degree extension of the vector. More concretely, let $\mathbf{a} \in \mathbb{F}^n$ and let $\mathbb{H}$ be a multiplicative subgroup of $\mathbb{F}$ of order n . The committer samples $\rho \leftarrow \mathbb{F}$, and defines $a(X) = \sum_{\ell=1}^n a_\ell \lambda_\ell(X) + \rho \nu_{\mathbb{H}}(X)$. It outputs the commitment $\mathbf{Com}(\mathbf{ck}, \mathbf{a}, \rho) = [a(\tau)]_1$.

For the rest of the paper, $\mathbf{ck}$ will represent the KZG commitment key and $\mathbf{Com}$ the KZG hiding vector commitment scheme. We have written that the commitment is given in $\mathbb{G}_1$, but the same holds for $\mathbb{G}_2$.

2.1 Semi-dynamic zkSNARKs

Next, we introduce the notion of universal semi-dynamic zkSNARK [ZOZ26]. Intuitively, a zkSNARK is semi-dynamic if it is possible to update proofs output by **Prove**, but not necessarily those output by **Update**. In contrast, a fully-dynamic zkSNARK (or simply, a dynamic zkSNARK) ensures that proofs can be updated, independently of how they were generated. The formal definition of fully-dynamic zkSNARK is given in Section A. We also remark that there is one main difference between the zero-knowledge notion of [ZOZ26] and the one from [WPSP26b]. For [WPSP26b], zero-knowledge means that a proof reveals no information about the witness; for [ZOZ26], it is additionally required that proofs generated by the **Prove** algorithm are indistinguishable from those generated by **Update**. We will focus on the stronger definition of [ZOZ26].

Definition 3. A universal semi-dynamic zkSNARK for an indexed relation $\mathcal{R}$ consists of the following PPT algorithms

$$\Pi = (\text{KeyGen}, \text{Derive}, \text{Prove}, \text{BuildData}, \text{Update}, \text{Verify}) :$$

- **KeyGen**(1^λ) $\rightarrow$ ($\mathbf{srs} = (\mathbf{pk}, \mathbf{vk})$, $\mathbf{trap}$): On input the security parameter 1^λ , outputs the universal structured reference string $\mathbf{srs}$ containing the universal prover key $\mathbf{pk}$, a verifier key $\mathbf{vk}$, and a trapdoor $\mathbf{trap}$ needed for the simulation.
- **Derive**($i, \mathbf{srs}$) $\rightarrow \mathbf{srs}_i = (\mathbf{pk}_i, \mathbf{vk}_i)$: On input an index i and the universal structured reference string $\mathbf{srs}$, outputs specialized structured reference string $\mathbf{srs}_i$ containing the specialized prover key $\mathbf{pk}_i$ and verifier key $\mathbf{vk}_i$ for the index i .
- **Prove**($\mathbf{pk}_i, \mathbf{x}, \mathbf{w}$) $\rightarrow (\pi, \mathbf{rand})$: On input the specialized prover key $\mathbf{pk}_i$ and instance $\mathbf{x}$ and witness $\mathbf{w}$, outputs the proof π and randomness $\mathbf{rand}$.
- **BuildData**($\mathbf{pk}_i, \mathbf{x}, \mathbf{w}, \mathbf{rand}$) $\rightarrow \mathbf{data}$: On input $\mathbf{pk}_i$, instance $\mathbf{x}$, witness $\mathbf{w}$, and prover's randomness $\mathbf{rand}$, outputs a witness-dependent data structure $\mathbf{data}$ used for updates.
- **Update**($\mathbf{pk}_i, \mathbf{x}, \mathbf{w}, \mathbf{x}', \mathbf{w}', \pi, \mathbf{data}$) $\rightarrow \pi'$: On input the specialized prover key $\mathbf{pk}_i$, valid extended witnesses $(\mathbf{x}, \mathbf{w}), (\mathbf{x}', \mathbf{w}')$ ⁶, proof π for $(i, \mathbf{x}, \mathbf{w})$ generated with **Prove** and data structure $\mathbf{data}$, outputs updated proof π' .
- **Verify**($\mathbf{vk}_i, \mathbf{x}, \pi$) $\rightarrow 0/1$: On input the specialized verifier key $\mathbf{vk}_i$, instance $\mathbf{x}$ and proof π output by **Prove** or **Update**, outputs accept or reject.

A semi-dynamic zkSNARK Π satisfies the following properties:

- **Completeness:** It is statistically complete if for any $(i, \mathbf{x}, \mathbf{w}), (i, \mathbf{x}', \mathbf{w}') \in \mathcal{R}$:

$$\Pr \left[\begin{array}{l} \text{Verify}(\mathbf{vk}_i, \mathbf{x}, \pi) = 1 \\ \text{Verify}(\mathbf{vk}_i, \mathbf{x}', \pi') = 1 \end{array} \middle| \begin{array}{l} (\mathbf{srs}, \mathbf{trap}) \leftarrow \text{KeyGen}(1^\lambda) \\ (\mathbf{pk}_i, \mathbf{vk}_i) \leftarrow \text{Derive}(i, \mathbf{srs}) \\ (\pi, \mathbf{rand}) \leftarrow \text{Prove}(\mathbf{pk}_i, \mathbf{x}, \mathbf{w}) \\ \mathbf{data} \leftarrow \text{BuildData}(\mathbf{pk}_i, \mathbf{x}, \mathbf{w}, \mathbf{rand}) \\ \pi' \leftarrow \text{Update}(\mathbf{pk}_i, \mathbf{x}, \mathbf{w}, \mathbf{x}', \mathbf{w}', \pi, \mathbf{data}) \end{array} \right] \geq 1 - \text{negl}(\lambda)$$

⁶ As noted in [WPSP26b], **Update** needs only the modified positions of the new instance-witness pair rather than the complete pair. We adopt this convention to simplify notation.

- **Knowledge Soundness:** It is knowledge sound if for any PPT adversary $\mathcal{A}$ there exists an extractor Ext such that:

$$\Pr \left[\begin{array}{l} \text{Verify}(\text{vk}_i, x, \pi) = 1 \\ (i, x, w) \notin \mathcal{R} \end{array} \middle| \begin{array}{l} (\text{srs}, \text{trap}) \leftarrow \text{KeyGen}(1^\lambda) \\ i \leftarrow \mathcal{A}(\text{srs}) \\ (\text{pk}_i, \text{vk}_i) \leftarrow \text{Derive}(i, \text{srs}) \\ (x, \pi) \leftarrow \mathcal{A}(i, \text{srs}) \\ w \leftarrow \text{Ext}(i, \text{srs}, x, \pi) \end{array} \right] \leq \text{negl}(\lambda)$$

- **Zero-knowledge:** It is zero-knowledge if there exists a simulator Sim such that for all PPT adversaries $\mathcal{A}$, every index i and every $(i, x, w), (i, x', w') \in \mathcal{R}$ we have

$$\Pr \left[\begin{array}{l} \mathcal{A}(\text{srs}, \text{trap}, i, x, w, \pi) = 1 \end{array} \middle| \begin{array}{l} (\text{srs}, \text{trap}) \leftarrow \text{KeyGen}(1^\lambda) \\ (\text{pk}_i, \text{vk}_i) \leftarrow \text{Derive}(i, \text{srs}) \\ (\pi, \text{rand}) \leftarrow \text{Prove}(\text{pk}_i, x, w) \end{array} \right]$$

$$\approx \Pr \left[\begin{array}{l} \mathcal{A}(\text{srs}, \text{trap}, i, x, w, \pi) = 1 \end{array} \middle| \begin{array}{l} (\text{srs}, \text{trap}) \leftarrow \text{KeyGen}(1^\lambda) \\ (\text{pk}_i, \text{vk}_i) \leftarrow \text{Derive}(i, \text{srs}) \\ \pi \leftarrow \text{Sim}(\text{trap}, i, x) \end{array} \right]$$

and

$$\Pr \left[\begin{array}{l} \mathcal{A}(\text{srs}, \text{trap}, i, x, w, x', w', \pi') = 1 \end{array} \middle| \begin{array}{l} (\text{srs}, \text{trap}) \leftarrow \text{KeyGen}(1^\lambda) \\ (\text{pk}_i, \text{vk}_i) \leftarrow \text{Derive}(i, \text{srs}) \\ (\pi', \text{rand}') \leftarrow \text{Prove}(\text{pk}_i, x', w') \end{array} \right] \approx$$

$$\Pr \left[\begin{array}{l} \mathcal{A}(\text{srs}, \text{trap}, i, x, w, x', w', \pi') = 1 \end{array} \middle| \begin{array}{l} (\text{srs}, \text{trap}) \leftarrow \text{KeyGen}(1^\lambda) \\ (\text{pk}_i, \text{vk}_i) \leftarrow \text{Derive}(i, \text{srs}) \\ (\pi, \text{rand}) \leftarrow \text{Prove}(\text{pk}_i, x, w) \\ \text{data} \leftarrow \text{BuildData}(\text{pk}_i, x, w, \text{rand}) \\ \pi' \leftarrow \text{Update}(\text{pk}_i, x, w, x', w', \pi, \text{data}) \end{array} \right]$$

- **Succinctness:** The proof size $|\pi|$ is polynomial in $(\lambda + \log |w|)$ and the running time of Verify is polynomial in $(\lambda + |x| + \log |w|)$.
- **Updatability:** Let k be the Hamming distance between (x, w) and (x', w') . If $k = o(|x| + |w|)$, then the running time of Update is sublinear in the running time of Prove .

The standard definition of zkSNARK can be easily recovered from the definition of semi-dynamic zkSNARK and is therefore omitted. Next, we give the formal definition of *sparse* zkSNARK as given in [WPSP26b], which are zkSNARKs where the prover complexity scales with the Hamming weight of the witness.

Definition 4 (Sparse zkSNARK). A zkSNARK is sparse if there exists a function $f(|x| + |w|) = o(|x| + |w|)$ such that for all $0 \leq k \leq |x| + |w|$, the running time of $\text{Prove}(\text{pk}_i, x, w)$ is $O(kf(|x| + |w|))$, where k is the Hamming weight of (x, w) .

From semi-dynamic to fully-dynamic. The work of [ZOZ26] gives a compiler that turns semi-dynamic zkSNARKs into fully-dynamic ones in the de-amortized setting. Intuitively, the updater needs to compute the new proof and part of the new data during each proof update. For this to work, we need to assume that the BuildData algorithm can be computed stepwise, across a fixed number of b steps. Let us explain in more detail.

Assume that the prover holds a sequence of instance-witness pairs $((x_0, w_0), (x_1, w_1), \dots)$ for which it wants to generate proofs. We will further assume that the Hamming distance between (x_j, w_j) and (x_{j+1}, w_{j+1}) is at most k , for some small value k^7 . Throughout, k always denotes this distance; when we need to refer to the sparsity of the base witness on which BuildData is run we write k_0 instead. This way, the prover can generate a base proof for (x_0, w_0) by running $\pi_0 \leftarrow \text{Prove}(\text{pk}_i, x_0, w_0)$, and compute $\text{data} \leftarrow \text{BuildData}(\text{pk}_i, x_0, w_0, \text{rand})$ in order to be able to use (x_0, w_0) as a base instance-witness pair for further updates. Specifically, an updater can then run $\text{Update}(\text{pk}_i, x_0, w_0, x_j, w_j, \pi_0, \text{data})$ to obtain subsequent proofs. Observe that the further the updated

⁷ If the distance between two consecutive instance-witness pairs is large, updates will be expensive, negating the benefits of dynamic zkSNARKs.

instance gets from (x_0, w_0) the higher the cost of the update will be, eventually reaching the threshold in which running the **Prove** algorithm to generate the proof is more cost-effective than performing updates. If we call b this limit distance, it directly follows that the amortized cost will be of $O(\frac{kT(\text{BuildData})}{b} + T(\text{Update}_{\text{semi},b}))$, where $T(\text{BuildData})$ denotes the running time of **BuildData** and $T(\text{Update}_{\text{semi},b})$ the one of the semi-dynamic **Update** algorithm when the Hamming distance between instances is b . This is how Dynaverse achieves its asymptotic cost for updates [WPSP26b].

As the amortized approach can be undesirable in settings where the update time is required to be stable, [ZOZ26] show how to de-amortize the general construction. First, we need to assume that the **BuildData** algorithm can be computed step-wise (i.e. we can divide it into b steps of the same running time). Consider the division of the sequence of instance-witnesses pairs into blocks $(B_1, B_2, \dots)$ such that the first pair in B_j and the first one in B_{j+1} (denoted (x_{B_j}, w_{B_j}) and $(x_{B_{j+1}}, w_{B_{j+1}})$, respectively) are at distance at most b . Now, instead of performing the updates in a block B_j using the preprocessed data associated to (x_{B_j}, w_{B_j}) , the updater will do it using $(x_{B_{j-1}}, w_{B_{j-1}})$ —observe that the increase of the Hamming distance from b to $2b$ does not affect the asymptotics—and, besides running the semi-dynamic **Update** algorithm, it will run k steps of the step-wise **BuildData**. Indeed, a block spans Hamming distance b and consecutive pairs are at distance at most k , so a block contains b/k updates; since **BuildData** is divided into b steps, each update must advance it by k of them, at a cost of $kT(\text{BuildData})/b$.

Theorem 1 ([ZOZ26]). *Let Π_{semi} be a semi-dynamic zkSNARK whose **BuildData** can be computed step-wise. There exists an algorithm $\text{Update}_{\text{fully}}$, of running time $O(kT(\text{BuildData})/b + T(\text{Update}_{\text{semi},b}))$, such that replacing $\text{Update}_{\text{semi}}$ by $\text{Update}_{\text{fully}}$ in Π_{semi} yields a fully-dynamic zkSNARK in the de-amortized setting. Here k denotes the Hamming distance between the original (x, w) and the updated (x', w') , and b is the number of steps.*

Commit-and-Prove (Semi) Dynamic zkSNARKs. Commit-and-Prove zkSNARKs (CP-SNARKs) are zkSNARKs whose relations verify predicates over commitments [CFQ19]. More formally, given a commitment scheme $\text{CS} = (\text{KeyGen}, \text{Com})$ with commitment key $\text{ck} \leftarrow \text{CS.KeyGen}(1^\lambda)$, we consider (universal) relations $\hat{\mathcal{R}}_{\text{ck}}$ parametrized by the commitment key. Given an index $\hat{i}_{\text{ck}}$, the instance witness pairs $(\hat{x}, \hat{w})$ are of the form $\hat{x} = (x, c_1, \dots, c_l)$, $\hat{w} = (m_1, \dots, m_l, \rho_1, \dots, \rho_l)$ ⁸ such that

$$(\hat{i}_{\text{ck}}, \hat{x}, \hat{w}) \in \hat{\mathcal{R}}_{\text{ck}} \iff (i, x, w) \in \mathcal{R} \wedge \text{Com}(\text{ck}, m_j, \rho_j) = c_j \quad \forall j \in [l],$$

where $w = (m_1, \dots, m_l)$, and i is an index of $\mathcal{R}$, a universal relation which can be efficiently derived from $\hat{\mathcal{R}}_{\text{ck}}$.

A commit-and-prove (semi) dynamic zkSNARK for the commitment scheme CS and universal relation $\mathcal{R}$ is simply a (semi) dynamic zkSNARK $\text{CP} = (\text{KeyGen}, \text{Derive}, \text{BuildData}, \text{Prove}, \text{Update}, \text{Verify})$ for $\hat{\mathcal{R}}_{\text{ck}}$. The only difference is that the key generation algorithm $\text{KeyGen}(1^\lambda, \text{ck}) \rightarrow (\text{srs}, \text{trap})$ will additionally take as input the commitment key from the commitment scheme.

3 A Compiler from Sparse to Dynamic zkSNARKs for Linear Relations

In this section, we show how to build dynamic zkSNARKs from sparse zkSNARKs for *linear relations*, as defined next. Looking ahead, Section 4 constructs such arguments for a general family of matrices, and instantiates them for Plonk and V-R1CS-lite.

Definition 5. *A linear relation is a relation of the form*

$$\mathcal{R}_{\text{ck}}^{\text{lin}} = \left\{ (i, x, w) \left| \begin{array}{l} i = (\mathbb{F}, (n, l) \in \mathbb{N}^2, \mathbf{M} = (M^{(1)}, \dots, M^{(l)})) \in (\mathbb{F}^{n \times n})^l \\ x = [a, b^{(1)}, \dots, b^{(l)}]_1 \in \mathbb{G}_1^{l+1} \\ w = ((\mathbf{a}, \mathbf{b}^{(1)}, \dots, \mathbf{b}^{(l)}), (\rho_a, \rho_{b^{(1)}}, \dots, \rho_{b^{(l)}})) \in (\mathbb{F}^n)^{l+1} \times \mathbb{F}^{l+1} \\ \forall j \in [l] : M^{(j)} \mathbf{a} = \mathbf{b}^{(j)}, \\ \text{Com}(\text{ck}, \mathbf{a}, \rho_a) = [a]_1, \text{Com}(\text{ck}, \mathbf{b}^{(j)}, \rho_{b^{(j)}}) = [b^{(j)}]_1 \quad \forall j \in [l] \end{array} \right. \right\}.$$

For simplicity we assume the vector $\mathbf{a}$ is shared across every matrix and that all matrices have the same dimension; this suffices for our purposes and the generalization is immediate. The number of matrices l should be understood as a small constant (in our protocols we will have at most $l = 3$).

⁸ In general, CP-SNARKs can have a part of the witness which is not committed, but for our purposes this is enough.

The naive compiler. Recall from Section 1.3 that a sparse argument for $\mathcal{R}_{\text{ck}}^{\text{lin}}$ already yields a semi-dynamic one: the prover publishes an *initial proof* π^{ini} for (x, w) , and the updater publishes the difference commitments $[a^*, (b^*)^{(1)}, \dots, (b^*)^{(l)}]_1$, where $a^* = a' - a$ and $(b^*)^{(j)} = (b')^{(j)} - b^{(j)}$, together with a sparse proof π^* for them. This is sound by linearity, but not zero-knowledge in the sense of Definition 3, since fresh and updated proofs can be distinguished. We repair it by making Prove send $l + 1$ randomized commitments to the zero vector and the corresponding proof for $M^{(j)}\mathbf{0} = \mathbf{0}$.

Linearity and null pairs. Since KZG commitments are linear, component-wise addition of instance-witness pairs is well defined and $\mathcal{R}_{\text{ck}}^{\text{lin}}$ is closed under it. This is the structure the compiler exploits. For $\text{rand} = (\sigma_a, \sigma_{b^{(1)}}, \dots, \sigma_{b^{(l)}}) \in \mathbb{F}^{l+1}$ we define the *null pair* $Z(\text{rand})$ as

$$Z(\text{rand}) = \left([\sigma_a \nu_{\mathbb{H}}(\tau), \sigma_{b^{(1)}} \nu_{\mathbb{H}}(\tau), \dots, \sigma_{b^{(l)}} \nu_{\mathbb{H}}(\tau)]_1, (\mathbf{0}, \dots, \mathbf{0}, \sigma_a, \sigma_{b^{(1)}}, \dots, \sigma_{b^{(l)}}) \right) \in \mathcal{R}_{\text{ck}}^{\text{lin}}.$$

A null pair has two important features we leverage: its witness vectors are $\mathbf{0}$, so a sparse prover can show $Z(\text{rand}) \in \mathcal{R}_{\text{ck}}^{\text{lin}}$ efficiently, and for uniform $\text{rand} \leftarrow \mathbb{F}^{l+1}$ its commitments are uniform in $\mathbb{G}_1^{l+1}$.

The compiler. The pair actually proved by the initial proof π^{ini} is never (x, w) but the re-randomization of it $(x^{\text{ini}}, w^{\text{ini}}) = (x, w) - Z(\text{rand})$, which keeps the vectors untouched and shifts each opening by $\rho \mapsto \rho - \sigma$. This is what BuildData computes, outputting $\text{data} = (\pi^{\text{ini}}, \text{rand})$. Having access to rand lets the updater—which by definition, already holds (x, w) —rebuild $(x^{\text{ini}}, w^{\text{ini}})$ with $O(l) = O(1)$ group operations. To give a proof for (x', w') , the updater interprets it as an update of the *re-randomized* pair, i.e. $(x^*, w^*) = (x', w') - (x^{\text{ini}}, w^{\text{ini}})$, rather than of (x, w) —necessarily so, since π^{ini} is a proof for x^{ini} . The shift is absorbed entirely into the openings: the vectors in w^* are still $a' - a$, $(b')^{(j)} - b^{(j)}$, while $[a^*, (b^*)^{(1)}, \dots, (b^*)^{(l)}]_1$ becomes uniform in $\mathbb{G}_1^{l+1}$. Finally, Prove behaves exactly like Update: a fresh proof is the update of (x, w) *to itself*. Indeed, $(x, w) - ((x, w) - Z(\text{rand})) = Z(\text{rand})$, so the dummy update is exactly a null pair with uniform openings.

Zero-knowledge is then immediate: fresh and updated proofs are outputs of the same algorithm on instances uniform in $\mathbb{G}_1^{l+1}$, so only the underlying witnesses differ, and these are hidden by the zero-knowledge of Π_{sparse} .

- $\text{KeyGen}(\text{ck}) \rightarrow \text{srs}$: output $\text{srs} = (\text{srs}^{\text{sparse}}, \text{ck})$ for $\text{srs}^{\text{sparse}} \leftarrow \text{Sparse.KeyGen}(\text{ck})$.
 $// \text{ck} = ([1, \tau, \dots, \tau^n]_1, [1, \tau, \dots, \tau^n]_2)$
- $\text{Derive}(\text{srs}, i) \rightarrow (\text{pk}_i, \text{vk}_i)$: output $(\text{pk}_i, \text{vk}_i) \leftarrow \text{Sparse.Derive}(\text{srs}, i)$.

$//$ Prove is defined in terms of BuildData and Update, given first.

- $\text{BuildData}(\text{pk}_i, x, w, \text{rand}) \rightarrow \text{data}$
 1. Parse $\text{rand} = (\sigma_a, \sigma_{b^{(1)}}, \dots, \sigma_{b^{(l)}}) \in \mathbb{F}^{l+1}$ and let $(x^{\text{ini}}, w^{\text{ini}}) = (x, w) - Z(\text{rand})$. $// [a^{\text{ini}}]_1 = [a]_1 - \sigma_a [\nu_{\mathbb{H}}(\tau)]_1$, $\rho_{a^{\text{ini}}} = \rho_a - \sigma_a$, $a^{\text{ini}} = a$
 2. Output $\text{data} = (\pi^{\text{ini}}, \text{rand})$ for $\pi^{\text{ini}} \leftarrow \text{Sparse.Prove}(\text{pk}_i, x^{\text{ini}}, w^{\text{ini}})$.
- $\text{Update}(\text{pk}_i, x, w, x', w', \pi, \text{data}) \rightarrow \pi'$ $// \pi$ is not used
 1. Parse $\text{data} = (\pi^{\text{ini}}, \text{rand})$ and let $(x^{\text{ini}}, w^{\text{ini}}) = (x, w) - Z(\text{rand})$.
 2. Let $(x^*, w^*) = (x', w') - (x^{\text{ini}}, w^{\text{ini}})$. $// a^* = a' - a$, $\rho_{a^*} = \rho'_a - \rho_{a^{\text{ini}}}$
 3. Output $\pi' = (x^*, \pi^*, \pi^{\text{ini}})$ for $\pi^* \leftarrow \text{Sparse.Prove}(\text{pk}_i, x^*, w^*)$.
- $\text{Prove}(\text{pk}_i, x, w) \rightarrow (\pi, \text{rand})$
 1. Sample $\text{rand} \leftarrow \mathbb{F}^{l+1}$ and compute $\text{data} \leftarrow \text{BuildData}(\text{pk}_i, x, w, \text{rand})$.
 2. Output (π, rand) for $\pi \leftarrow \text{Update}(\text{pk}_i, x, w, x, w, \perp, \text{data})$. $//$ identity update; here $(x^*, w^*) = Z(\text{rand})$
- $\text{Verify}(\text{vk}_i, x', \pi') \rightarrow 0/1$
 1. Parse $\pi' = (x^*, \pi^*, \pi^{\text{ini}})$ with $x^* = [a^*, (b^*)^{(1)}, \dots, (b^*)^{(l)}]_1$, and let $x^{\text{ini}} = x' - x^*$.
 2. Check $\text{Sparse.Verify}(\text{vk}_i, x^*, \pi^*) \stackrel{?}{=} 1$ and $\text{Sparse.Verify}(\text{vk}_i, x^{\text{ini}}, \pi^{\text{ini}}) \stackrel{?}{=} 1$.

Fig. 1. Compiler from sparse zkSNARKs to semi-dynamic zkSNARKs for $\mathcal{R}_{\text{ck}}^{\text{lin}}$.

Theorem 2. Let Π_{sparse} be a sparse zkSNARK for $\mathcal{R}_{\text{ck}}^{\text{lin}}$. Then the protocol of Figure 1 is a semi-dynamic zkSNARK for $\mathcal{R}_{\text{ck}}^{\text{lin}}$. Write $T(k)$ for the running time of **Sparse.Prove** on a witness whose vectors have k non-zero entries in total, let k be that quantity for w and k^* that for $w' - w$. Then

- **Prove** runs in time $T(k) + T(0) + O(l)$.
- **BuildData** runs in time $T(k) + O(l)$.
- **Update** runs in time $T(k^*) + O(l + k^*)$.
- **Verify** performs $O(l)$ operations in $\mathbb{G}_1$ and two invocations of **Sparse.Verify**.
- Proofs output by **Prove** and by **Update** both consist of $l + 1$ elements of $\mathbb{G}_1$ and two proofs of Π_{sparse} .

Before proving the theorem we give some elementary facts that we will use. Recall that $Z(\text{rand})$ denotes the null pair with openings $\text{rand} = (\sigma_a, \sigma_{b(1)}, \dots, \sigma_{b(l)}) \in \mathbb{F}^{l+1}$, and write

$$Z_x(\text{rand}) = [\sigma_a \nu_{\mathbb{H}}(\tau), \sigma_{b(1)} \nu_{\mathbb{H}}(\tau), \dots, \sigma_{b(l)} \nu_{\mathbb{H}}(\tau)]_1 \in \mathbb{G}_1^{l+1}$$

for its instance part. For every index i and every rand we have $(i, Z(\text{rand})) \in \mathcal{R}_{\text{ck}}^{\text{lin}}$, since $M^{(j)} \mathbf{0} = \mathbf{0}$ for all $j \in [l]$ and $\text{Com}(\text{ck}, \mathbf{0}, \sigma) = \sigma[\nu_{\mathbb{H}}(\tau)]_1$.

Lemma 2 (Uniformity). Assume $\nu_{\mathbb{H}}(\tau) \neq 0$ and let $\text{rand} \leftarrow \mathbb{F}^{l+1}$. Then $Z_x(\text{rand})$ is uniformly distributed on $\mathbb{G}_1^{l+1}$.

Proof. The group $\mathbb{G}_1$ has prime order and $[\nu_{\mathbb{H}}(\tau)]_1 \neq [0]_1$, so the map $\sigma \mapsto \sigma[\nu_{\mathbb{H}}(\tau)]_1$ is a bijection from $\mathbb{F}$ to $\mathbb{G}_1$. The $l + 1$ coordinates of rand are independent and uniform, hence so are their images. $\square$

Proposition 1. The protocol from Figure 1 is complete.

Proof. Let $(i, x, w), (i, x', w') \in \mathcal{R}_{\text{ck}}^{\text{lin}}$ and $\text{rand} \in \mathbb{F}^{l+1}$. Applying linearity first to (x, w) and $Z(\text{rand})$, and then to (x', w') and $(x^{\text{ini}}, w^{\text{ini}})$, both

$$(i, x^{\text{ini}}, w^{\text{ini}}) = (i, x, w) - Z(\text{rand}), \quad (i, x^*, w^*) = (i, x', w') - (i, x^{\text{ini}}, w^{\text{ini}})$$

lie in $\mathcal{R}_{\text{ck}}^{\text{lin}}$. Hence every pair on which **Sparse.Prove** is invoked is a valid one, and π^{ini}, π^* are accepted by **Sparse.Verify** on x^{ini} and x^* by completeness of Π_{sparse} . On input (x', π') the verifier computes $x' - x^*$, which equals x^{ini} by definition of x^* , so it runs **Sparse.Verify** on exactly the two instances that were proved and both checks pass. Proofs output by **Prove** are the special case $(x', w') = (x, w)$, for which $(x^*, w^*) = (x, w) - ((x, w) - Z(\text{rand})) = Z(\text{rand})$, again a valid pair. Therefore Π is complete. $\square$

Proposition 2. The protocol from Figure 1 is knowledge sound.

Proof. Let $\mathcal{A}$ be a PPT adversary that on input (i, srs) outputs (x', π') with $\text{Verify}(\text{vk}_i, x', \pi') = 1$. Parse $\pi' = (x^*, \pi^*, \pi^{\text{ini}})$ and let $x^{\text{ini}} = x' - x^*$ be the instance computed by the verifier. Define two PPT adversaries against Π_{sparse} : $\mathcal{B}_1$ runs $\mathcal{A}$ and outputs (x^*, π^*) , and $\mathcal{B}_2$ runs $\mathcal{A}$ and outputs $(x^{\text{ini}}, \pi^{\text{ini}})$. Since **Verify** accepts only if both **Sparse.Verify** checks pass, whenever $\mathcal{A}$ wins so do $\mathcal{B}_1$ and $\mathcal{B}_2$.

Let $\text{Ext}_1, \text{Ext}_2$ be the extractors guaranteed by the knowledge soundness of Π_{sparse} for $\mathcal{B}_1, \mathcal{B}_2$, and let Ext run both on the transcript of $\mathcal{A}$ and output $w' = w^{\text{ini}} + w^*$, where $w^* \leftarrow \text{Ext}_1$ and $w^{\text{ini}} \leftarrow \text{Ext}_2$. Except with negligible probability, we have (i, x^*, w^*) and $(i, x^{\text{ini}}, w^{\text{ini}})$ in $\mathcal{R}_{\text{ck}}^{\text{lin}}$, and then using linearity gives $(i, x^{\text{ini}} + x^*, w') \in \mathcal{R}_{\text{ck}}^{\text{lin}}$. As $x^{\text{ini}} + x^* = x'$ by definition of x^{ini} , the extracted w' is a witness for x' . A union bound over the two extraction failures concludes the proof. $\square$

Proposition 3. The protocol from Figure 1 is zero-knowledge.

Proof. Intuitively, the simulator of Figure 2 behaves as the **Update** algorithm would, except that it knows no witness for either of the two pairs it must prove, and therefore invokes **Sparse.Sim** in place of **Sparse.Prove**. The argument requires $\nu_{\mathbb{H}}(\tau) \neq 0$, which fails with probability $n/|\mathbb{F}|$; we condition on this event throughout. Denote by $\varepsilon_{\text{sparse}}(\lambda)$ the maximal advantage of a PPT distinguisher against the zero-knowledge of Π_{sparse} .

We claim that for every index i and every $(i, x, w), (i, x', w') \in \mathcal{R}_{\text{ck}}^{\text{lin}}$, each of the two distributions

- (a) π' , where $(\pi', \text{rand}) \leftarrow \text{Prove}(\text{pk}_i, x', w')$;

Fig. 2. Simulator for the protocol of Figure 1

$\text{Sim}(\text{trap}, i, \mathbf{x}' = [a', (b')^{(1)}, \dots, (b')^{(l)}]_1) \rightarrow \pi'$
1. Sample $\text{rand} = (\sigma_a, \sigma_{b^{(1)}}, \dots, \sigma_{b^{(l)}}) \leftarrow \mathbb{F}^{l+1}$ and compute $\mathbf{x}^* = Z_x(\text{rand})$. 2. Compute $\mathbf{x}^{\text{ini}} = \mathbf{x}' - \mathbf{x}^*$. 3. Compute $\pi^* \leftarrow \text{Sparse.Sim}(\text{trap}, i, \mathbf{x}^*)$. 4. Compute $\pi^{\text{ini}} \leftarrow \text{Sparse.Sim}(\text{trap}, i, \mathbf{x}^{\text{ini}})$. 5. Output $\pi' = (\mathbf{x}^*, \pi^*, \pi^{\text{ini}})$.

(b) $\pi' \leftarrow \text{Update}(\text{pk}_i, \mathbf{x}, \mathbf{w}, \mathbf{x}', \mathbf{w}', \pi, \text{data})$, where $(\pi, \text{rand}) \leftarrow \text{Prove}(\text{pk}_i, \mathbf{x}, \mathbf{w})$ and $\text{data} \leftarrow \text{BuildData}(\text{pk}_i, \mathbf{x}, \mathbf{w}, \text{rand})$,

is at distance at most $2\varepsilon_{\text{sparse}}(\lambda) + n/|\mathbb{F}|$ from $\text{Sim}(\text{trap}, i, \mathbf{x}')$. The first condition of the zero-knowledge notion in Definition 3 is exactly (a), and the second follows from (a) and (b) by the triangle inequality.

By definition of Prove (see Figure 1), both (a) and (b) are outputs of Update ; write them as $\pi' = (\mathbf{x}^*, \pi^*, \pi^{\text{ini}})$ with $\mathbf{x}^{\text{ini}} = \mathbf{x} - Z_x(\text{rand})$ and $\mathbf{x}^* = \mathbf{x}' - \mathbf{x}^{\text{ini}}$, where $\mathbf{x} = \mathbf{x}'$ in case (a). We argue about each element output by the simulator.

- *The element $\mathbf{x}^*$.* In a real execution $\mathbf{x}^* = \mathbf{x}' - \mathbf{x} + Z_x(\text{rand})$, which by Lemma 2 is uniform on $\mathbb{G}_1^{l+1}$, and it determines $\mathbf{x}^{\text{ini}} = \mathbf{x}' - \mathbf{x}^*$. In the simulated execution $\mathbf{x}^* = Z_x(\text{rand})$ is uniform on $\mathbb{G}_1^{l+1}$ by the same lemma, and again determines $\mathbf{x}^{\text{ini}}$. The two are therefore identically distributed, in case (a) as well as in case (b).
- *The proofs π^{ini} and π^* .* Conditioned on $\mathbf{x}^*$, both instances are fixed, and they lie in the language: using linearity, the pairs $(\mathbf{x}^{\text{ini}}, \mathbf{w}^{\text{ini}})$ and $(\mathbf{x}^*, \mathbf{w}^*)$ computed by Update belong to $\mathcal{R}_{\text{ck}}^{\text{lin}}$. Consider the hybrids H_0 , the real distribution; H_1 , in which π^{ini} is replaced by $\text{Sparse.Sim}(\text{trap}, i, \mathbf{x}^{\text{ini}})$; and H_2 , in which π^* is additionally replaced by $\text{Sparse.Sim}(\text{trap}, i, \mathbf{x}^*)$.

For H_0 and H_1 , let $\mathcal{B}_3$ be the distinguisher against Π_{sparse} that, on input $(\text{srs}, \text{trap})$ and given $i, \mathbf{x}, \mathbf{w}, \mathbf{x}', \mathbf{w}'$, samples $\text{rand} \leftarrow \mathbb{F}^{l+1}$, forms $(\mathbf{x}^{\text{ini}}, \mathbf{w}^{\text{ini}})$ and $(\mathbf{x}^*, \mathbf{w}^*)$, submits $(i, \mathbf{x}^{\text{ini}}, \mathbf{w}^{\text{ini}})$ to its challenger, computes $\pi^* \leftarrow \text{Sparse.Prove}(\text{pk}_i, \mathbf{x}^*, \mathbf{w}^*)$ itself, and forwards $(\mathbf{x}^*, \pi^*, \pi^{\text{ini}})$. It is a legitimate adversary by the previous paragraph, and it simulates H_0 or H_1 according to its challenge, so the two hybrids differ by at most $\varepsilon_{\text{sparse}}(\lambda)$.

For H_1 and H_2 , let $\mathcal{B}_4$ proceed as $\mathcal{B}_3$ but submit $(i, \mathbf{x}^*, \mathbf{w}^*)$ and compute $\pi^{\text{ini}} \leftarrow \text{Sparse.Sim}(\text{trap}, i, \mathbf{x}^{\text{ini}})$ itself, which it can do because the zero-knowledge experiment hands it trap . Again the two hybrids differ by at most $\varepsilon_{\text{sparse}}(\lambda)$.

In H_2 both proof components are produced by Sparse.Sim from $\mathbf{x}^{\text{ini}}$ and $\mathbf{x}^*$ alone, and by the first bullet these instances have the same joint distribution as in $\text{Sim}(\text{trap}, i, \mathbf{x}')$, namely $(\mathbf{x}' - u, u)$ for u uniform on $\mathbb{G}_1^{l+1}$. Hence H_2 and the simulated distribution coincide, which proves the claim and the theorem. $\square$

4 Sparse zkSNARKs for Linear Relations

The goal of this section is to construct *sparse* zkSNARKs for linear relations, that is, protocols for proving $\mathbf{b}^{(j)} = M^{(j)} \mathbf{a}$ for $j \in [l]$ whose prover complexity depends on the sparsity of the witness rather than on the ambient dimension n .

In Section 4.1 we give the general protocol that admits a non-sparse prover and in Section 4.2 we describe how to turn it into a sparsity-dependent one. In Section 4.3 we instantiate the protocol for the linear relation arising in the Plonk constraint system and in Section 4.4 for the bounded fan-out R1CS-lite one. These sparse arguments are then compiled into semi-dynamic and fully-dynamic zkSNARKs using Theorems 2 and 1, respectively.

We recall first a useful definition from [RZ21].

Definition 6. For $n_1, n_2 \in \mathbb{N}$ and $V \in [n_1]$, let $\mathcal{M}_{n_1, n_2}(V)$ denote the set of matrices $M \in \mathbb{F}^{n_1 \times n_2}$ with at most V non-zero entries per column. If $n_1 = n_2 = n$ we write $\mathcal{M}_n(V)$, and if $V = 1$ we say that M is basic.

Each $M^{(j)}$ lies in $\mathcal{M}_n(V^{(j)})$, so if $\mathbf{a}$ has sparsity k then $\mathbf{b}^{(j)}$ has sparsity at most $kV^{(j)}$; we write $V = \sum_{j=1}^l V^{(j)}$. Looking ahead, we will have $V = 3$ in our Plonk construction and $V = O(\log n)$ in our R1CS-lite one.

4.1 The General Protocol

Instead of proving each lincheck separately, we batch them with a challenge $\delta \leftarrow \mathbb{F}$ and prove $\sum_{j=1}^l \delta^{j-1} M^{(j)} \mathbf{a} = \sum_{j=1}^l \delta^{j-1} \mathbf{b}^{(j)}$. Reducing this matrix-vector product to an inner product against $\mathbf{r} = \lambda(r)$ for $r \leftarrow \mathbb{F} \setminus \mathbb{H}^9$ gives

$$\sum_{\ell=1}^n \left(\mathbf{r}^\top \sum_{j=1}^l \delta^{j-1} M^{(j)} \right)_\ell a_\ell = \sum_{\ell=1}^n r_\ell \left(\sum_{j=1}^l \delta^{j-1} b_\ell^{(j)} \right).$$

Let $\tilde{\mathbf{p}} = (\mathbf{r}^\top \sum_j \delta^{j-1} M^{(j)}) \circ \mathbf{a}$ and $\mathbf{p} = \mathbf{r} \circ (\sum_j \delta^{j-1} \mathbf{b}^{(j)})$, and let $p(X), \tilde{p}(X)$ be their randomized low-degree extensions over $\mathbb{H}$. The lincheck then decomposes into two tasks: (i) prove the compressed relation $\sum_\ell \tilde{p}(\mathbf{h}_\ell) = \sum_\ell p(\mathbf{h}_\ell)$ via the univariate sumcheck, and (ii) prove that $p(X)$ and $\tilde{p}(X)$ are well formed. The first uses a trick from [RZ21, CFF⁺24] that avoids a separate degree check; the second reduces to two zero-checks over $\mathbb{H}$, which we batch.

Sumcheck. We blind the sumcheck following [CFF⁺21]. The prover samples $\rho_{S,1}, \rho_{S,2} \leftarrow \mathbb{F}$ and sets $S(X) = \rho_{S,1}X + \rho_{S,2}\nu_{\mathbb{H}}(X)$, so that $\sum_\ell S(\mathbf{h}_\ell) = 0$. With a challenge $\eta \leftarrow \mathbb{F}$, the identity $\sum_\ell (p(\mathbf{h}_\ell) - \tilde{p}(\mathbf{h}_\ell) + \eta S(\mathbf{h}_\ell)) = 0$ holds whenever $\sum_\ell p(\mathbf{h}_\ell) - \tilde{p}(\mathbf{h}_\ell) = 0$, and by Lemma 1, since the blinded $p(X), \tilde{p}(X), S(X)$ have degree n ,

$$p(X) - \tilde{p}(X) + \eta S(X) = Xg(X) + h\nu_{\mathbb{H}}(X)$$

for a unique $g(X)$ of degree at most $n-2$ and $h \in \mathbb{F}$. To avoid a degree check on $g(X)$ the prover commits to $G(X) = g(X)(X^2 - 1)$ and sends h , so that

$$(p(X) - \tilde{p}(X) + \eta S(X))(X^2 - 1) = XG(X) + h\nu_{\mathbb{H}}(X)(X^2 - 1). \quad (1)$$

Correctness of $p(X)$. Write $b^{(j)}(X)$ for the randomized low-degree extension of $\mathbf{b}^{(j)}$. Since $\lambda_\ell(X) = \frac{\mathbf{h}_\ell}{n} \frac{X^n - 1}{X - \mathbf{h}_\ell}$, we have $p(\mathbf{h}_\ell) = \frac{\mathbf{h}_\ell}{n} \frac{r^n - 1}{r - \mathbf{h}_\ell} (\sum_j \delta^{j-1} b^{(j)}(\mathbf{h}_\ell))$. Clearing denominators, which is an equivalence because $r \notin \mathbb{H}$, $p(X)$ is well formed if and only if there exists $q(X)$ with

$$np(X)(r - X) - (r^n - 1) \left(\sum_{j=1}^l \delta^{j-1} b^{(j)}(X) \right) X = q(X) \nu_{\mathbb{H}}(X). \quad (2)$$

Correctness of $\tilde{p}(X)$. Let $a(X)$ be the randomized low-degree extension of $\mathbf{a}$. Here $\tilde{p}(\mathbf{h}_\ell) = a(\mathbf{h}_\ell) (\mathbf{r}^\top \sum_j \delta^{j-1} M^{(j)})_\ell$, and since

$$(\mathbf{r}^\top M^{(j)})_\ell = \sum_{i: M_{i\ell}^{(j)} \neq 0} \frac{\mathbf{h}_i}{n} \frac{r^n - 1}{r - \mathbf{h}_i} M_{i\ell}^{(j)},$$

the ℓ -th coordinate equals $\frac{r^n - 1}{n} C_{\delta, \ell}(r)$ for the rational function

$$C_{\delta, \ell}(Y) = \sum_{j=1}^l \delta^{j-1} \sum_{i: M_{i\ell}^{(j)} \neq 0} \frac{\mathbf{h}_i}{Y - \mathbf{h}_i} M_{i\ell}^{(j)}. \quad (3)$$

The protocol is driven by a representation of these n rational functions as a ratio of two polynomials drawn from a common, matrix-dependent basis. This is formalized in the following definition.

Definition 7 (Rational encoding). Let $\mathbf{i} = (\mathbb{F}, (n, l), \mathbf{M})$ be an index. A rational encoding of $\mathbf{i}$ is a tuple $\text{RE} = (\mathcal{E}, \mathcal{X}, \alpha, \beta)$ where

⁹ We will see why we need $r \notin \mathbb{H}$ below. Since in our instantiations $|\mathbb{H}|/|\mathbb{F}|$ is negligible, this does not affect soundness.

- $\mathcal{E} = (e_1, \dots, e_t)$ is a list of index polynomials of degree at most $n - 1$ depending only on i ;
- $\mathcal{X} = (X^j)_{j \in J}$, for some $J \subseteq \{0, \dots, n - 1\}$, is a list of monomials, independent of i ; we set $u = |J|$ and $d = \max(J \cup \{1\})^{10}$;
- $\alpha = (\alpha_s)_{s \in [t+u]}$, $\beta = (\beta_s)_{s \in [t+u]}$ are maps $\alpha_s : \mathbb{F} \rightarrow \mathbb{F}$, $\beta_s : \mathbb{F}^2 \rightarrow \mathbb{F}$ such that, given δ, r , the vectors $(\alpha_s(r))_s$ and $(\beta_s(r, \delta))_s$ are computable in $O(t + u)$ field operations;

such that, writing $(f_1, \dots, f_{t+u})$ for the concatenation of $\mathcal{E}$ and $\mathcal{X}$ and setting

$$\hat{I}_r(X) = \sum_{s=1}^{t+u} \alpha_s(r) f_s(X), \quad \hat{R}_{\delta,r}(X) = \sum_{s=1}^{t+u} \beta_s(r, \delta) f_s(X), \quad (4)$$

we have, for every $r \in \mathbb{F} \setminus \mathbb{H}$ —the excluded set containing every pole of every $C_{\delta,\ell}(Y)$ —every $\delta \in \mathbb{F}$ and every $\ell \in [n]$,

$$\hat{I}_r(\mathbf{h}_\ell) \neq 0 \quad \text{and} \quad C_{\delta,\ell}(r) = \frac{\hat{R}_{\delta,r}(\mathbf{h}_\ell)}{\hat{I}_r(\mathbf{h}_\ell)}. \quad (5)$$

The split between $\mathcal{E}$ and $\mathcal{X}$ is what determines the preprocessing: the index polynomials are dense and matrix-dependent, so their commitments must be produced by Derive in order to achieve succinct verification, whereas the commitments to the monomials are already part of ck and need no preprocessing. Given Definition 7 we have $\tilde{p}(\mathbf{h}_\ell) = a(\mathbf{h}_\ell) \frac{r^n - 1}{n} \frac{\hat{R}_{\delta,r}(\mathbf{h}_\ell)}{\hat{I}_r(\mathbf{h}_\ell)}$, so $\tilde{p}(X)$ is well formed if and only if there exists $\tilde{q}(X)$ with

$$n \tilde{p}(X) \hat{I}_r(X) - (r^n - 1) a(X) \hat{R}_{\delta,r}(X) = \tilde{q}(X) \nu_{\mathbb{H}}(X). \quad (6)$$

Batching, proof and verification. Equations (2) and (6) are zero-checks over $\mathbb{H}$, so we batch them under η and the prover sends only a commitment to $Q(X) = q(X) + \eta \tilde{q}(X)$. The proof is

$$\pi = ([S(\tau), p(\tau), \tilde{p}(\tau), G(\tau), Q(\tau)]_1, h).$$

By (4) the verifier computes, in $O(t + u)$ group operations,

$$[\hat{I}_r(\tau)]_2 = \sum_s \alpha_s(r) [f_s(\tau)]_2, \quad [\hat{R}_{\delta,r}(\tau)]_2 = \sum_s \beta_s(r, \delta) [f_s(\tau)]_2,$$

assuming that $[e_s(\tau)]_2$ and $[\tau^j]_2$ are included in vk_i . It then tests (2), (6) and (1) with pairings; as these share the $\mathbb{G}_2$ argument $[\tau]_2$, it batches them with $\xi \leftarrow \mathbb{F}$ and checks

$$\begin{aligned} & e(n[p(\tau)]_1, [r - \tau]_2) + \eta \left(e(n[\tilde{p}(\tau)]_1, [\hat{I}_r(\tau)]_2) - e((r^n - 1)[a(\tau)]_1, [\hat{R}_{\delta,r}(\tau)]_2) \right) \\ & + \xi e([p(\tau)]_1 - [\tilde{p}(\tau)]_1 + \eta[S(\tau)]_1, [\tau^2 - 1]_2) \stackrel{?}{=} e([Q(\tau)]_1 + \xi h[\tau^2 - 1]_1, [\nu_{\mathbb{H}}(\tau)]_2) \\ & + e((r^n - 1)(\sum_{j=1}^l \delta^{j-1} [b^{(j)}(\tau)]_1) + \xi[G(\tau)]_1, [\tau]_2) \end{aligned} \quad (7)$$

Prover and Verifier Keys. The verifier needs the commitments $[e_s(\tau), \{\tau^j\}_{j \in J}]_2$ in order to form $[\hat{I}_r(\tau)]_2$ and $[\hat{R}_{\delta,r}(\tau)]_2$ in $O(t + u)$ group operations; these depend on the index and must be preprocessed. The prover, on the other hand, can compute all five of its commitments from the commitment key alone. On input srs and i , Derive therefore outputs

$$\text{pk}_i = (i, [1, \tau, \dots, \tau^n]_1), \quad (8)$$

$$\text{vk}_i = ([1, \tau^2]_1, [1, \tau, \tau^2, \nu_{\mathbb{H}}(\tau), \{\tau^j\}_{j \in J}, \{e_s(\tau)\}_{s=1}^t]_2). \quad (9)$$

The full protocol is given in Figure 3. We remark that, as described, the prover interpolates $p(X), \tilde{p}(X)$ and $G(X)$ from their evaluations over $\mathbb{H}$ and obtains $q(X), \tilde{q}(X)$ by polynomial division, which takes $O(n)$ operations in $\mathbb{G}_1$ and $O(n(t + V + \log n))$ in $\mathbb{F}$; Section 4.2 reduces this to time proportional to the sparsity of $\mathbf{a}$, at the cost of a larger pk_i .

¹⁰ Note that $u \leq d + 1$, since the exponents in J are distinct and bounded by d ; we implicitly assume that u and d are small constants.

- **KeyGen**(ck) $\rightarrow$ srs: output $\text{srs} = ([1, \tau, \dots, \tau^n]_1, [1, \tau, \dots, \tau^n]_2)$.
 - **Derive**(srs, i) $\rightarrow$ (pk_i, vk_i): output pk_i, vk_i as in (8) and (9), relative to a rational encoding RE of i.
-
- **Prove**(pk_i, x = [a, b⁽¹⁾, ..., b^(l)]₁, w = (a, b⁽¹⁾, ..., b^(l), ρ_a, ρ_{b(1)}, ..., ρ_{b(l)})) $\rightarrow$ π
 1. Sample $\rho_{S,1}, \rho_{S,2} \leftarrow \mathbb{F}$ and compute $[S(\tau)]_1 = \rho_{S,1}[\tau]_1 + \rho_{S,2}[\nu_{\mathbb{H}}(\tau)]_1$. // sumcheck mask
 2. Let $r, \delta \leftarrow \text{RO}(\text{vk}_i, x, [S(\tau)]_1)$. // evaluation and matrix-batching challenges
 3. Sample $\rho_p \leftarrow \mathbb{F}$ and compute $[p(\tau)]_1$ for

$$p(X) = \sum_{\ell=1}^n \left(\sum_{j=1}^l \delta^{j-1} b^{(j)}(\mathbf{h}_\ell) \right) \lambda_\ell(r) \lambda_\ell(X) + \rho_p \nu_{\mathbb{H}}(X)$$
 4. Sample $\tilde{\rho}_p \leftarrow \mathbb{F}$ and compute $[\tilde{p}(\tau)]_1$ for

$$\tilde{p}(X) = \sum_{\ell=1}^n a(\mathbf{h}_\ell) \frac{r^n - 1}{n} C_{\delta, \ell}(r) \lambda_\ell(X) + \tilde{\rho}_p \nu_{\mathbb{H}}(X)$$
 5. Let $\eta \leftarrow \text{RO}(\text{vk}_i, x, [S(\tau), p(\tau), \tilde{p}(\tau)]_1)$. // batching challenge
 6. Compute $[G(\tau)]_1$ and $h \in \mathbb{F}$ satisfying

$$(p(X) - \tilde{p}(X) + \eta S(X))(X^2 - 1) = X G(X) + h \nu_{\mathbb{H}}(X)(X^2 - 1)$$
 7. Compute $[q(\tau)]_1$ satisfying

$$n p(X)(r - X) - (r^n - 1) \left(\sum_{j=1}^l \delta^{j-1} b^{(j)}(X) \right) X = q(X) \nu_{\mathbb{H}}(X)$$
 8. Let $\hat{I}_r(X), \hat{R}_{\delta, r}(X)$ be as in (4) and compute $[\tilde{q}(\tau)]_1$ satisfying

$$n \tilde{p}(X) \hat{I}_r(X) - (r^n - 1) a(X) \hat{R}_{\delta, r}(X) = \tilde{q}(X) \nu_{\mathbb{H}}(X)$$
 9. Set $[Q(\tau)]_1 = [q(\tau)]_1 + \eta [\tilde{q}(\tau)]_1$.
 10. Output $\pi = ([S(\tau), p(\tau), \tilde{p}(\tau), G(\tau), Q(\tau)]_1, h)$.
 - **Verify**(vk_i, x, π) $\rightarrow$ 0/1
 1. Recompute $r, \delta, \eta \leftarrow \text{RO}$ as above and sample $\xi \leftarrow \mathbb{F}$.
 2. Compute $[\hat{I}_r(\tau)]_2 = \sum_s \alpha_s(r) [f_s(\tau)]_2$ and $[\hat{R}_{\delta, r}(\tau)]_2 = \sum_s \beta_s(r, \delta) [f_s(\tau)]_2$ from vk_i.
 3. Check (7).

Fig. 3. zkSNARK for $\mathcal{R}_{\text{ck}}^{\text{lin}}$ from a rational encoding $\text{RE} = (\mathcal{E}, \mathcal{X}, \alpha, \beta)$ of i. RO denotes a random oracle.

Theorem 3. Let RE be a rational encoding of i, with t and u as in Definition 7. Figure 3 is a zkSNARK in the AGM and the ROM under the (n, n) -PDL assumption. A proof consists of 5 elements of $\mathbb{G}_1$ and 1 of $\mathbb{F}$. Verify performs $O(t + u + l)$ group operations, $O(t + u + \log n)$ field operations and 6 pairings. In particular the verifier is succinct whenever $t + u = O(\log n)$.

Let us begin by discussing the claimed complexities. For **Verify**, computing $[\hat{I}_r(\tau)]_2$ and $[\hat{R}_{\delta, r}(\tau)]_2$ takes $2(t + u)$ scalar multiplications in $\mathbb{G}_2$ and $O(t + u)$ evaluations of α, β ; forming $\sum_j \delta^{j-1} [b^{(j)}(\tau)]_1$ takes $O(l)$ operations in $\mathbb{G}_1$; computing r^n takes $O(\log n)$ field operations; and (7) involves 6 pairings.

Next, we prove completeness, knowledge soundness and zero-knowledge. Throughout we write $B(X) = \sum_{j=1}^l \delta^{j-1} b^{(j)}(X)$ and $\mathbf{B} = \sum_{j=1}^l \delta^{j-1} \mathbf{b}^{(j)}$, and we use repeatedly the identity $n \lambda_\ell(X)(X - \mathbf{h}_\ell) = \mathbf{h}_\ell(X^n - 1)$, immediate from $\lambda_\ell(X) = \frac{\mathbf{h}_\ell X^n - 1}{n(X - \mathbf{h}_\ell)}$.

Proposition 4 (Completeness). The protocol from Figure 3 is complete.

Proof. Let $(i, x, w) \in \mathcal{R}_{\text{ck}}^{\text{lin}}$, so $\mathbf{b}^{(j)} = M^{(j)} \mathbf{a}$ for every $j \in [l]$ and the commitments open to $\mathbf{a}$ and the $\mathbf{b}^{(j)}$. The challenge r is an output of the random oracle, so $r \notin \mathbb{H}$ except with probability $n/|\mathbb{F}|$; we assume this

from now on, which is what Definition 7 requires. Evaluating the polynomials of Figure 3 at $\mathbf{h}_\ell$ gives

$$p(\mathbf{h}_\ell) = \lambda_\ell(r)B(\mathbf{h}_\ell), \quad \tilde{p}(\mathbf{h}_\ell) = a(\mathbf{h}_\ell) \frac{r^n - 1}{n} C_{\delta, \ell}(r) = a(\mathbf{h}_\ell) \left(\mathbf{r}^\top \sum_{j=1}^l \delta^{j-1} M^{(j)} \right)_\ell.$$

We check the three identities enforced by (7).

- *Well-formedness of $p(X)$.* Expanding $p(\mathbf{h}_\ell)$ in the left-hand side of (2),

$$n p(\mathbf{h}_\ell)(r - \mathbf{h}_\ell) - (r^n - 1)B(\mathbf{h}_\ell)\mathbf{h}_\ell = B(\mathbf{h}_\ell)(n \lambda_\ell(r)(r - \mathbf{h}_\ell) - (r^n - 1)\mathbf{h}_\ell) = 0$$

for every $\ell \in [n]$, by the identity above evaluated at $X = r$. The left-hand side of (2) therefore vanishes on $\mathbb{H}$ and the quotient $q(X)$ exists.

- *Well-formedness of $\tilde{p}(X)$.* By (5) we have $\hat{I}_r(\mathbf{h}_\ell) \neq 0$ and $C_{\delta, \ell}(r) = \hat{R}_{\delta, r}(\mathbf{h}_\ell)/\hat{I}_r(\mathbf{h}_\ell)$, so expanding $\tilde{p}(\mathbf{h}_\ell)$ in the left-hand side of (6),

$$n \tilde{p}(\mathbf{h}_\ell) \hat{I}_r(\mathbf{h}_\ell) - (r^n - 1)a(\mathbf{h}_\ell) \hat{R}_{\delta, r}(\mathbf{h}_\ell) = a(\mathbf{h}_\ell)(r^n - 1)(C_{\delta, \ell}(r) \hat{I}_r(\mathbf{h}_\ell) - \hat{R}_{\delta, r}(\mathbf{h}_\ell)) = 0,$$

so the quotient $\tilde{q}(X)$ exists.

- *Sumcheck.* Since $\mathbf{b}^{(j)} = M^{(j)} \mathbf{a}$ for every j , we have $(\mathbf{r}^\top \sum_j \delta^{j-1} M^{(j)}) \mathbf{a} = \mathbf{r}^\top \mathbf{B}$ for every $\mathbf{r} \in \mathbb{F}^n$ and $\delta \in \mathbb{F}$, that is, $\sum_\ell \tilde{p}(\mathbf{h}_\ell) = \sum_\ell p(\mathbf{h}_\ell)$. As $\sum_\ell S(\mathbf{h}_\ell) = \rho_{S,1} \sum_\ell \mathbf{h}_\ell = 0$, also $\sum_\ell (p(\mathbf{h}_\ell) - \tilde{p}(\mathbf{h}_\ell) + \eta S(\mathbf{h}_\ell)) = 0$, and by Lemma 1, $p(X) - \tilde{p}(X) + \eta S(X) = Xg(X) + h\nu_{\mathbb{H}}(X)$ with $\deg g \leq n-2$. Multiplying by $X^2 - 1$ yields $G(X) = (X^2 - 1)g(X)$ and the scalar h satisfying (1).

With $q(X), \tilde{q}(X), G(X), h$ as above and $Q(X) = q(X) + \eta \tilde{q}(X)$, the three relations hold as polynomial identities, so both sides of (7) agree for every $\xi \in \mathbb{F}$. The completeness error is therefore at most $n/|\mathbb{F}|$. $\square$

Proposition 5 (Knowledge Soundness). *The protocol from Figure 3 is knowledge sound in the AGM and ROM under the (n, n) -PDL assumption.*

Proof. Throughout, the probabilities below are stated for a single random-oracle query; against an adversary making Q queries, each carries a factor Q , since the adversary may run the argument on any of its query prefixes and output the transcript it prefers. Let $\mathcal{A}$ be an algebraic adversary which, on input (i, srs) , outputs an instance $\mathbf{x} = [a, b^{(1)}, \dots, b^{(l)}]_1$ and a proof $\pi = ([S(\tau), p(\tau), \tilde{p}(\tau), G(\tau), Q(\tau)]_1, h)$ accepted the verifier. As $\mathcal{A}$ is algebraic, it therefore also outputs polynomials $a(X), b^{(1)}(X), \dots, b^{(l)}(X), S(X), p(X), \tilde{p}(X), G(X), Q(X)$ of degree at most n explaining each of these elements. Define

$$\begin{aligned} F_1(X) &:= n p(X)(r - X) + \eta(n \tilde{p}(X) \hat{I}_r(X) - (r^n - 1)a(X) \hat{R}_{\delta, r}(X)) \\ &\quad - (r^n - 1)B(X)X - Q(X)\nu_{\mathbb{H}}(X), \\ F_2(X) &:= (p(X) - \tilde{p}(X) + \eta S(X))(X^2 - 1) - XG(X) - h(X^2 - 1)\nu_{\mathbb{H}}(X), \end{aligned} \tag{10}$$

so that, in terms of discrete logarithms, (7) reads $F_1(\tau) + \xi F_2(\tau) = 0$. Since $\deg \hat{I}_r, \deg \hat{R}_{\delta, r} \leq n-1$ by Definition 7, we have $\deg F_1 \leq 2n$ and $\deg F_2 \leq n+2$. The verifier samples $\xi \leftarrow \mathbb{F}$ after π is fixed, so if $(F_1(\tau), F_2(\tau)) \neq (0, 0)$ the check passes with probability at most $1/|\mathbb{F}|$; except with that probability, $F_1(\tau) = F_2(\tau) = 0$. As in most AGM proofs we distinguish two cases: an information-theoretic one, where $F_1(X)$ and $F_2(X)$ are both constantly zero, and a computational one, where either $F_1(X) \neq 0$ or $F_2(X) \neq 0$, which reduces to the PDL assumption.

Information-theoretic case: We first exploit $F_1(X) = 0$. Because the challenges r, δ are derived by the random oracle from $\mathbf{x}$ and $[S(\tau)]_1$, and η from $[p(\tau)]_1, [\tilde{p}(\tau)]_1$ as well, the polynomials $a, b^{(1)}, \dots, b^{(l)}, p, \tilde{p}$ and the scalars r, δ are all fixed before η is sampled; among the polynomials appearing in $F_1(X)$, only $Q(X)$ may depend on η . Evaluating $F_1(X) = 0$ at $X = \mathbf{h}_\ell$ gives us

$$\underbrace{(n p(\mathbf{h}_\ell)(r - \mathbf{h}_\ell) - (r^n - 1)B(\mathbf{h}_\ell)\mathbf{h}_\ell)}_{=: A_{0, \ell}} + \eta \underbrace{(n \tilde{p}(\mathbf{h}_\ell) \hat{I}_r(\mathbf{h}_\ell) - (r^n - 1)a(\mathbf{h}_\ell) \hat{R}_{\delta, r}(\mathbf{h}_\ell))}_{=: A_{1, \ell}} = 0,$$

a degree-one polynomial in η whose coefficients $A_{0,\ell}, A_{1,\ell}$ are fixed before η is sampled. Hence, for each ℓ , either $A_{0,\ell} = A_{1,\ell} = 0$ or the equation holds with probability at most $1/|\mathbb{F}|$ over η ; by a union bound over $\ell \in [n]$, except with probability $n/|\mathbb{F}|$ both vanish for every $\ell \in [n]$. Moreover $r \notin \mathbb{H}$ except with probability $n/|\mathbb{F}|$ per query—an event we must exclude explicitly, since (5) guarantees $\hat{I}_r(\mathbf{h}_\ell) \neq 0$ only outside $\mathbb{H}$. Outside it, $r \neq \mathbf{h}_\ell$ and $\hat{I}_r(\mathbf{h}_\ell) \neq 0$. Solving $A_{0,\ell} = 0$ and $A_{1,\ell} = 0$ therefore gives, for every $\ell \in [n]$,

$$p(\mathbf{h}_\ell) = \frac{(r^n - 1)\mathbf{h}_\ell B(\mathbf{h}_\ell)}{n(r - \mathbf{h}_\ell)} = \lambda_\ell(r)B(\mathbf{h}_\ell) \quad (11)$$

$$\tilde{p}(\mathbf{h}_\ell) = a(\mathbf{h}_\ell) \frac{r^n - 1}{n} \frac{\hat{R}_{\delta,r}(\mathbf{h}_\ell)}{\hat{I}_r(\mathbf{h}_\ell)} = a(\mathbf{h}_\ell) \left(\mathbf{r}^\top \sum_{j=1}^l \delta^{j-1} M^{(j)} \right)_\ell \quad (12)$$

the last equality by (5) and the definition of $C_{\delta,\ell}(Y)$ (see Definition 7).

We now exploit $F_2(X) = 0$, which gives $X G(X) = (X^2 - 1)(p(X) - \tilde{p}(X) + \eta S(X) - h\nu_{\mathbb{H}}(X))$. As $\gcd(X, X^2 - 1) = 1$, we get $G(X) = g(X)(X^2 - 1)$ for some $g(X)$ with $\deg g \leq n - 2$, and cancelling $X^2 - 1$,

$$p(X) - \tilde{p}(X) + \eta S(X) = X g(X) + h \nu_{\mathbb{H}}(X).$$

By Lemma 1 this means $\sum_\ell (p(\mathbf{h}_\ell) - \tilde{p}(\mathbf{h}_\ell)) + \eta \sum_\ell S(\mathbf{h}_\ell) = 0$. Again $S(X), p(X), \tilde{p}(X)$ are fixed before η , so except with probability $1/|\mathbb{F}|$ both sums vanish, and in particular $\sum_\ell p(\mathbf{h}_\ell) = \sum_\ell \tilde{p}(\mathbf{h}_\ell)$. Substituting Equations (11) and (12) and writing $\mathbf{a} = (a(\mathbf{h}_\ell))_\ell$, $\mathbf{b}^{(j)} = (b^{(j)}(\mathbf{h}_\ell))_\ell$, this reads

$$\mathbf{r}^\top \left(\sum_{j=1}^l \delta^{j-1} (\mathbf{b}^{(j)} - M^{(j)} \mathbf{a}) \right) = 0.$$

Consider the bivariate polynomial

$$P(X, Y) = \sum_{\ell=1}^n \left(\sum_{j=1}^l Y^{j-1} (b_\ell^{(j)} - (M^{(j)} \mathbf{a})_\ell) \right) \lambda_\ell(X),$$

of total degree at most $n+l-2$, whose coefficients are fixed before r and δ are sampled; the displayed equation says $P(r, \delta) = 0$. By the Schwartz–Zippel lemma, either $P(X, Y)$ is the zero polynomial or this happens with probability at most $(n+l-2)/|\mathbb{F}|$. Since $\{\lambda_\ell(X)\}_{\ell \in [n]}$ and $\{Y^{j-1}\}_{j \in [l]}$ are linearly independent, $P \equiv 0$ forces $\mathbf{b}^{(j)} = M^{(j)} \mathbf{a}$ for every $j \in [l]$.

It remains to produce the openings. Setting

$$\rho_a = \frac{a(X) - \sum_{\ell=1}^n a(\mathbf{h}_\ell) \lambda_\ell(X)}{\nu_{\mathbb{H}}(X)}, \quad \rho_{b^{(j)}} = \frac{b^{(j)}(X) - \sum_{\ell=1}^n b^{(j)}(\mathbf{h}_\ell) \lambda_\ell(X)}{\nu_{\mathbb{H}}(X)},$$

we get $\text{Com}(\text{ck}, \mathbf{a}, \rho_a) = [a]_1$ and $\text{Com}(\text{ck}, \mathbf{b}^{(j)}, \rho_{b^{(j)}}) = [b^{(j)}]_1$. The extractor outputs the witness $\mathbf{w} = (\mathbf{a}, \mathbf{b}^{(1)}, \dots, \mathbf{b}^{(l)}, \rho_a, \rho_{b^{(1)}}, \dots, \rho_{b^{(l)}})$, and $(i, \mathbf{x}, \mathbf{w}) \in \mathcal{R}_{\text{ck}}^{\text{lin}}$. Collecting the error terms, extraction fails with probability at most $(3n+l)/|\mathbb{F}|$.

Computational case: Here one of $F_1(X), F_2(X)$ is non-zero while $F_1(\tau) = F_2(\tau) = 0$. We assume $F_1(X) \neq 0$, the other case being analogous. Recall that $\deg F_1 \leq 2n$. Let $\mathcal{B}(\{[\tau^i]_1, [\tau^i]_2\}_{i=0}^n)$ be the (n, n) -PDL adversary that computes srs , pk_i and vk_i from its challenge—possible because, by (8) and (9), every element of the keys is of the form $[f(\tau)]_1$ or $[f(\tau)]_2$ with $\deg f \leq n$ and coefficients computable from i —and then runs $\mathcal{A}$ on (i, srs) , simulating the random oracle. As $\mathcal{A}$ is algebraic, $\mathcal{B}$ obtains the representations of all its output elements and can compute the coefficients of $F_1(X)$. It then runs a probabilistic polynomial-time root-finding algorithm to obtain the at most $2n$ roots $x_1, \dots, x_{2n}$ of $F_1(X)$ and returns the one satisfying $x_i[1]_1 = [\tau]_1$, thereby breaking the (n, n) -PDL assumption. $\square$

Proposition 6 (Zero-knowledge). *The protocol from Figure 3 is zero-knowledge.*

Proof. Intuitively, the simulator from Figure 4 behaves as the honest prover would on the witness $\mathbf{a} = \mathbf{b}^{(1)} = \dots = \mathbf{b}^{(l)} = \mathbf{0}$. Since in general this is not the case, the quotients $q(X)$ and $\tilde{q}(X)$ need not be polynomials, so

the simulator computes their commitments using the trapdoor τ to make the verifier accept. It queries the random oracle exactly where the honest prover does and never programs it, and it fails only when $\nu_{\mathbb{H}}(\tau) = 0$, as steps 8 and 10 divide by it. We condition on the events $\nu_{\mathbb{H}}(\tau) \neq 0$, $\tau^2 \neq 1$, $\eta \neq 0$ and $r \notin \mathbb{H}$, the last of which is needed for the honest prover to be defined; they fail with probability at most $n/|\mathbb{F}|$, $2/|\mathbb{F}|$, $1/|\mathbb{F}|$ and $n/|\mathbb{F}|$ respectively.

Write $p(X) = P_0(X) + \rho_p \nu_{\mathbb{H}}(X)$ and $\tilde{p}(X) = \tilde{P}_0(X) + \tilde{\rho}_p \nu_{\mathbb{H}}(X)$ with $\deg P_0, \deg \tilde{P}_0 \leq n-1$, and $S(X) = \rho_{S,1}X + \rho_{S,2}\nu_{\mathbb{H}}(X)$. By the sumcheck bullet above, $\sum_{\ell} (P_0 - \tilde{P}_0)(h_{\ell}) = 0$, so Lemma 1 gives $P_0(X) - \tilde{P}_0(X) = X g_1(X)$ with $\deg g_1 \leq n-2$, and

$$p(X) - \tilde{p}(X) + \eta S(X) = X(g_1(X) + \eta \rho_{S,1}) + (\rho_p - \tilde{\rho}_p + \eta \rho_{S,2})\nu_{\mathbb{H}}(X). \quad (13)$$

Hence $G(X) = (g_1(X) + \eta \rho_{S,1})(X^2 - 1)$ and $h = \rho_p - \tilde{\rho}_p + \eta \rho_{S,2}$ are, in a real execution, the unique polynomial and scalar satisfying (1). The simulator is the special case $P_0 = \tilde{P}_0 = 0$, whence $g_1 = 0$, which is exactly steps 6 and 7 of Figure 4. We argue about each element output by the simulator.

- *The elements $S(\tau)$, $p(\tau)$ and $\tilde{p}(\tau)$.* Since $\nu_{\mathbb{H}}(\tau) \neq 0$, the maps $\rho_p \mapsto P_0(\tau) + \rho_p \nu_{\mathbb{H}}(\tau)$ and $\tilde{\rho}_p \mapsto \tilde{P}_0(\tau) + \tilde{\rho}_p \nu_{\mathbb{H}}(\tau)$ are bijections of $\mathbb{F}$, and so is $\rho_{S,2} \mapsto \rho_{S,1}\tau + \rho_{S,2}\nu_{\mathbb{H}}(\tau)$ for each fixed $\rho_{S,1}$. As the four blinders are independent and uniform, the triple $(S(\tau), p(\tau), \tilde{p}(\tau))$ is uniform on $\mathbb{F}^3$ in a real execution, and steps 1, 3 and 4 of the simulator produce the same distribution. The challenges r, δ, η are oracle outputs on this prefix and are therefore equally distributed in the two experiments.
- *The element $G(\tau)$.* Notice that $\rho_{S,1}$ remains uniform on $\mathbb{F}$, even conditioned on $S(\tau)$. Since $\eta(\tau^2 - 1) \neq 0$, the map $\rho_{S,1} \mapsto (g_1(\tau) + \eta \rho_{S,1})(\tau^2 - 1)$ is a bijection of $\mathbb{F}$, so $G(\tau)$ is uniform on $\mathbb{F}$ in a real execution, as it is in the simulated one. Hence $(S(\tau), p(\tau), \tilde{p}(\tau), G(\tau))$ is uniform on $\mathbb{F}^4$ in both.
- *The values h and $Q(\tau)$.* In a real execution both are uniquely determined by the elements computed previously, and the simulator chooses exactly those values. Indeed, writing the second of the two equations batched by ξ in terms of discrete logarithms,

$$(p(\tau) - \tilde{p}(\tau) + \eta S(\tau))(\tau^2 - 1) = G(\tau)\tau + h(\tau^2 - 1)\nu_{\mathbb{H}}(\tau),$$

and cancelling $\tau^2 - 1$ using $G(X) = g(X)(X^2 - 1)$ gives $h = (p(\tau) - \tilde{p}(\tau) + \eta S(\tau) - g(\tau)\tau)/\nu_{\mathbb{H}}(\tau)$, which is uniquely determined since $\nu_{\mathbb{H}}(\tau) \neq 0$; substituting the simulator's values yields $h = \rho - \tilde{\rho} + \eta \rho_{S,2}$, as claimed in step 7. Finally, the first equation (10) in the knowledge soundness proof and split as in the information-theoretic case, determines $q(\tau)$ and $\tilde{q}(\tau)$ uniquely, again because $\nu_{\mathbb{H}}(\tau) \neq 0$; these are steps 8 and 10, and therefore $[Q(\tau)]_1$ is distributed as in a real execution.

Conditioned on the four events above the two transcripts are identically distributed, so the statistical distance is at most $(2n+3)/|\mathbb{F}|$. $\square$

4.2 The Sparse Prover

The protocol of Figure 3 is complete, knowledge sound and zero-knowledge for any rational encoding, but computing π as described takes $O(n \log n)$ operations. We now show that with suitable precomputation the prover can run in time proportional to k , the sparsity of the vector $\mathbf{a} \in \mathbb{F}^n$. We remark that the proof π , the verification equation and Theorem 3 are unaffected, and only the way the prover evaluates steps 3, 4 and 6–8 of Figure 3, together with the contents of $\mathbf{pk}_i$, are modified. Which precomputation is needed is exactly what the following lemmas determine. Everything rests on the following basic identity.

Lemma 3. *Let $f(X) \in \mathbb{F}[X]$ with $\deg f \leq n-1$ and let $\ell \in [n]$. Then*

$$\left\lfloor \frac{\lambda_{\ell}(X)f(X)}{\nu_{\mathbb{H}}(X)} \right\rfloor = \frac{h_{\ell}}{n} \cdot \frac{f(X) - f(h_{\ell})}{X - h_{\ell}}, \quad \lambda_{\ell}(X)f(X) \bmod \nu_{\mathbb{H}}(X) = f(h_{\ell})\lambda_{\ell}(X).$$

In particular $\lfloor X^j \lambda_{\ell}(X) / \nu_{\mathbb{H}}(X) \rfloor = \frac{h_{\ell}}{n} \sum_{i=0}^{j-1} h_{\ell}^{j-1-i} X^i$, computable in $O(j)$ field operations. Moreover $\lambda_{\ell}(0) = \frac{1}{n}$.

Fig. 4. Simulator for the protocol of Figure 3

$\text{Sim}(\tau, i, x = [a, b^{(1)}, \dots, b^{(l)}]_1) \rightarrow \pi$
1. Sample $\rho_{S,1}, \rho_{S,2} \leftarrow \mathbb{F}$ and compute $S(\tau) = \rho_{S,1}\tau + \rho_{S,2}\nu_{\mathbb{H}}(\tau)$ 2. Compute challenges $r, \delta \leftarrow \text{RO}$ 3. Sample $\rho \leftarrow \mathbb{F}$ and compute $p(\tau) = \rho \nu_{\mathbb{H}}(\tau)$ 4. Sample $\tilde{\rho} \leftarrow \mathbb{F}$ and compute $\tilde{p}(\tau) = \tilde{\rho} \nu_{\mathbb{H}}(\tau)$ 5. Compute challenge $\eta \leftarrow \text{RO}$ 6. Compute $G(\tau) = \eta \rho_{S,1}(\tau^2 - 1)$ 7. Compute $h = \rho - \tilde{\rho} + \eta \rho_{S,2}$ 8. Compute $[q(\tau)]_1 = \frac{n p(\tau)(r - \tau)[1]_1 - (r^n - 1)\tau \sum_{j=1}^l \delta^{j-1} [b^{(j)}]_1}{\nu_{\mathbb{H}}(\tau)}$ 9. Compute $\hat{I}_r(\tau)$ and $\hat{R}_{\delta,r}(\tau)$, where $\hat{I}_r(X)$ and $\hat{R}_{\delta,r}(X)$ are as in (4) 10. Compute $[\tilde{q}(\tau)]_1 = \frac{n \tilde{p}(\tau) \hat{I}_r(\tau)[1]_1 - (r^n - 1) \hat{R}_{\delta,r}(\tau)[a]_1}{\nu_{\mathbb{H}}(\tau)}$ 11. Compute $[Q(\tau)]_1 = [q(\tau)]_1 + \eta [\tilde{q}(\tau)]_1$ 12. Output $\pi = ([S(\tau), p(\tau), \tilde{p}(\tau), G(\tau), Q(\tau)]_1, h)$

Proof. Since $\lambda_{\ell}(X) = \frac{h_{\ell} \nu_{\mathbb{H}}(X)}{n X - h_{\ell}}$, we have $\frac{\lambda_{\ell}(X)f(X)}{\nu_{\mathbb{H}}(X)} = \frac{h_{\ell} f(X)}{n X - h_{\ell}} = \frac{h_{\ell} f(X) - f(h_{\ell})}{n X - h_{\ell}} + \frac{h_{\ell} f(h_{\ell})}{n X - h_{\ell}}$. Multiplying by $\nu_{\mathbb{H}}(X)$ gives

$$\lambda_{\ell}(X)f(X) = \nu_{\mathbb{H}}(X) \cdot \frac{h_{\ell} f(X) - f(h_{\ell})}{n X - h_{\ell}} + f(h_{\ell})\lambda_{\ell}(X),$$

and the last term has degree $n - 1 < n$. The monomial case is $f(X) = X^j$, and $\lambda_{\ell}(0) = \frac{1}{n}$ follows from the definition of Lagrange polynomial. $\square$

Lemma 4 (Computing $p(X)$). *The polynomial $p(X)$ from Step 3 has at most kV non-zero coefficients in the Lagrange basis, computable in $O(kV + \log n)$ time.*

Proof. The prover computes $\frac{r^n - 1}{n}$ once in $O(\log n)$. Since $\mathbf{a}$ has sparsity k and $M^{(j)} \in \mathcal{M}_n(V^{(j)})$, i.e., it has at most $V^{(j)}$ non-zero entries per column, then $\mathbf{b}^{(j)} = M^{(j)}\mathbf{a}$ has sparsity at most $kV^{(j)}$. Therefore, there are at most $k \sum_{j=1}^l V^{(j)} = kV$ non-zero terms $\frac{h_{\ell}}{r - h_{\ell}} b^{(j)}(h_{\ell})$ across all j and ℓ , each computable in $O(1)$.¹¹ $\square$

Lemma 5 (Computing $\tilde{p}(X)$). *The polynomial $\tilde{p}(X)$ from Step 4 has at most k non-zero coefficients in the Lagrange basis, computable in $O(kV + \log n)$ time.*

Proof. At most k values of ℓ satisfy $a(h_{\ell}) \neq 0$. For each of the non zero values, the prover computes $\tilde{p}(h_{\ell}) = a(h_{\ell}) \frac{r^n - 1}{n} \sum_{j=1}^l \delta^{j-1} \sum_{i: M_{i\ell}^{(j)} \neq 0} \frac{h_i}{r - h_i} M_{i\ell}^{(j)}$. Again, $\frac{r^n - 1}{n}$ requires $O(\log n)$ and only needs to be computed once. The sum has at most V terms, each computable in $O(1)$. $\square$

Lemma 6 (Computing $G(X)$ and h). *Let $G_{\ell}(X) = (\lambda_{\ell}(X) - \frac{1}{n}) \frac{X^2 - 1}{X}$ for $\ell \in [n]$, a polynomial of degree n . Then, we can write h and $G(X)$ from Equation (1) as $h = \rho_p - \tilde{\rho}_p + \eta \rho_{S,2}$ and*

$$G(X) = \sum_{\ell=1}^n (p(h_{\ell}) - \tilde{p}(h_{\ell})) G_{\ell}(X) + \eta \rho_{S,1}(X^2 - 1),$$

so $G(X)$ is a linear combination of the $G_{\ell}(X)$ and $X^2 - 1$ with $O(kV)$ non-zero coefficients, computable in $O(kV + \log n)$ time. Computing h takes $O(1)$.

Proof. Write $\varphi(X) = p(X) - \tilde{p}(X) + \eta S(X) = \sum_{\ell} c_{\ell} \lambda_{\ell}(X) + \rho \nu_{\mathbb{H}}(X)$ with

$$c_{\ell} = p(h_{\ell}) - \tilde{p}(h_{\ell}) + \eta \rho_{S,1} h_{\ell}, \quad \rho = \rho_p - \tilde{\rho}_p + \eta \rho_{S,2}.$$

¹¹ This requires inverting $O(kV)$ field elements, which is concretely expensive; we mitigate it with a Montgomery batch inversion [Mon87,BBCT22].

Matching the coefficient of X^n in $\varphi(X) = Xg(X) + h\nu_{\mathbb{H}}(X)$ directly gives $h = \rho$. Moreover $Xg(X) = \sum_{\ell} c_{\ell} \lambda_{\ell}(X)$, and by Lemma 3 $\lambda_{\ell}(0) = \frac{1}{n}$, so with $g_{\ell}(X) = (\lambda_{\ell}(X) - \frac{1}{n})/X$ we get

$$\sum_{\ell} c_{\ell} \lambda_{\ell}(X) = X \sum_{\ell} c_{\ell} g_{\ell}(X) + \frac{1}{n} \sum_{\ell} c_{\ell}$$

and the last sum vanishes by the sumcheck condition (see Equation (1)). Hence $g(X) = \sum_{\ell} c_{\ell} g_{\ell}(X)$ and $G(X) = g(X) \cdot (X^2 - 1) = \sum_{\ell} c_{\ell} G_{\ell}(X)$. Using

$$\sum_{\ell} h_{\ell} g_{\ell}(X) = (\sum_{\ell} h_{\ell} \lambda_{\ell}(X) - \frac{1}{n} \sum_{\ell} h_{\ell})/X = X/X = 1,$$

since $\sum_{\ell} h_{\ell} \lambda_{\ell}(X) = X$ and $\sum_{\ell} h_{\ell} = 0$, yields the stated expression, whose non-zero coefficients are indexed by those ℓ with $p(h_{\ell}) \neq 0$ or $\tilde{p}(h_{\ell}) \neq 0$, of which there are $O(kV)$. $\square$

Lemma 7 (Computing $q(X)$). *The polynomial $q(X)$ from Equation (2) has degree 1 and its coefficients are computable in $O(kV + \log n)$ time.*

Proof. By linearity of Euclidean division, $q(X) = q_1(X) - q_2(X)$ for $q_1(X) = \lfloor n p(X)(r - X)/\nu_{\mathbb{H}}(X) \rfloor$ and $q_2(X) = \lfloor (r^n - 1)X(\sum_j \delta^{j-1} b^{(j)}(X))/\nu_{\mathbb{H}}(X) \rfloor$. Expanding in the Lagrange basis and using Lemma 3, namely $\lfloor X \lambda_{\ell}(X)/\nu_{\mathbb{H}}(X) \rfloor = \frac{h_{\ell}}{n}$ and $\lfloor \lambda_{\ell}(X)/\nu_{\mathbb{H}}(X) \rfloor = 0$,

$$q_1(X) = n \rho_p(r - X) - \sum_{\ell=1}^n h_{\ell} p(h_{\ell}), \quad q_2(X) = (r^n - 1) \left(X \sum_{j=1}^l \delta^{j-1} \rho_{b^{(j)}} + \sum_{j=1}^l \delta^{j-1} \sum_{\ell=1}^n b_{\ell}^{(j)} \frac{h_{\ell}}{n} \right).$$

Both are computable in $O(kV + \log n)$ as in Lemmas 4 and 5. $\square$

The strategy for computing $\tilde{q}(X)$ from Equation (6) in time proportional to the sparsity is a bit more involved. First, we write $\tilde{q}(X) = \tilde{q}_1(X) - \tilde{q}_2(X)$, where $\tilde{q}_1(X) = \lfloor n \tilde{p}(X) \hat{I}_r(X)/\nu_{\mathbb{H}}(X) \rfloor$ and $\tilde{q}_2(X) = \lfloor (r^n - 1) a(X) \hat{R}_{\delta,r}(X)/\nu_{\mathbb{H}}(X) \rfloor$. Note that we cannot simply precompute the quotients $\lfloor \lambda_{\ell}(X) \hat{I}_r(X)/\nu_{\mathbb{H}}(X) \rfloor$, as $\hat{I}_r(X)$ depends on the challenge r . However, by definition of rational encoding (see Definition 7), the polynomial $\hat{I}_r(X)$ decomposes as a linear combination of challenge independent polynomials (same holds for $\hat{R}_{\delta,r}(X)$), i.e., $\hat{I}_r(X) = \sum_{s=1}^{t+u} \alpha_s(r) f_s(X)$, and therefore

$$\begin{aligned} \tilde{q}_1(X) &= \sum_{s=1}^{t+u} \left(n \alpha_s(r) \sum_{\ell=1}^n \tilde{p}(h_{\ell}) \lfloor \lambda_{\ell}(X) f_s(X)/\nu_{\mathbb{H}}(X) \rfloor \right) + n \tilde{\rho}_p \hat{I}_r(X). \\ \tilde{q}_2(X) &= \sum_{s=1}^{t+u} \left((r^n - 1) \beta_s(r, \delta) \sum_{\ell=1}^n a(h_{\ell}) \lfloor \lambda_{\ell}(X) f_s(X)/\nu_{\mathbb{H}}(X) \rfloor \right) + (r^n - 1) \rho_a \hat{R}_{\delta,r}(X). \end{aligned}$$

Remark 1 (Only $\mathcal{E}$ requires preprocessing). For $\ell \in [n]$ and $s \in [t+u]$ let $q_{\ell}^{(s)}(X) = \lfloor \lambda_{\ell}(X) f_s(X)/\nu_{\mathbb{H}}(X) \rfloor$. Recall that the polynomials $f_s(X)$ given by the rational encoding consists of some dense, matrix-dependent polynomials (those in $\mathcal{E}$) and some monomials (those in $\mathcal{X}$). The quotients with $s > t$ (i.e. the ones associated with the monomials) need not be precomputed. Indeed, if $f_s(X) = X^j$ for some $j \in J$, Lemma 3 shows that $q_{\ell}^{(s)}(X)$ is a polynomial of degree $j - 1$ —the zero polynomial when $j = 0$ —that can be computed in $O(j)$ operations. Given that $j \leq d$ and d is assumed to be a small constant, the prover can compute this polynomial online, only using the commitment key ck and without any further precomputations.

Lemma 8 (Computing $\tilde{q}(X)$). *With $q_{\ell}^{(s)}(X)$ as in Remark 1, the polynomial $\tilde{q}(X)$ from Equation (6) can be computed as*

$$\tilde{q}(X) = \sum_{s=1}^{t+u} \left(n \alpha_s(r) \sum_{\ell=1}^n \tilde{p}(h_{\ell}) q_{\ell}^{(s)}(X) - (r^n - 1) \beta_s(r, \delta) \sum_{\ell=1}^n a(h_{\ell}) q_{\ell}^{(s)}(X) \right) + n \tilde{\rho}_p \hat{I}_r(X) - (r^n - 1) \rho_a \hat{R}_{\delta,r}(X),$$

that is, a combination of $O(k(t+u))$ of the $q_{\ell}^{(s)}(X)$ together with $\hat{I}_r(X)$ and $\hat{R}_{\delta,r}(X)$, whose coefficients are computable in $O(k(t+ud) + \log n)$ time.

Proof. Expand $\tilde{p}(X)$ and $a(X)$ in the Lagrange basis, apply (4) and use the linearity of Euclidean division together with $\lfloor \nu_{\mathbb{H}}(X)f_s(X)/\nu_{\mathbb{H}}(X) \rfloor = f_s(X)$. There are at most k indices ℓ with $a(h_\ell) \neq 0$, and the same holds for $\tilde{p}(h_\ell)$ by Lemma 5, so at most $k(t+u)$ terms survive; all of the $\alpha_s(r)$ and $\beta_s(r, \delta)$ can be computed in $O(t+u)$ so the all coefficients can be computed in $O(k(t+u) + \log n)$. The terms $q_\ell^{(s)}(X)$ with $s > t$ cost a further $O(ud)$ per index ℓ by Remark 1. $\square$

The augmented proving key. By the lemmas above, the prover can compute its desired five commitments from linear combinations of the elements

$$[\lambda_\ell(\tau), G_\ell(\tau), \{q_\ell^{(s)}(\tau)\}_{s \in [t]}]_1, \quad \ell \in [n],$$

together with $[e_s(\tau)]_1$ and, by Remark 1, the elements $[1, \tau, \dots, \tau^{d-1}]_1$ for the monomial part. Only the index-dependent ones require preprocessing, so we let **Derive** additionally output them, replacing the prover key of (8) by

$$\mathbf{pk}_i = \left(i, [1, \tau, \dots, \tau^n]_1, [e_s(\tau)]_{s=1}^t, [\lambda_\ell(\tau), G_\ell(\tau), \{q_\ell^{(s)}(\tau)\}_{s=1}^t]_{\ell=1}^n \right). \quad (14)$$

We remark that the n elements $\{[\lambda_\ell(\tau)]_1\}_\ell$ can be computed with $O(n \log n)$ operations by standard techniques [FK23, WPSP26b, ZOZ26], and the same holds for $\{[G_\ell(\tau)]_1\}_\ell$ and $\{[q_\ell^{(s)}(\tau)]_1\}_\ell$. The verification key of (9) remains unchanged.

Proposition 7 (Sparse Prover Complexity). *Let RE be a rational encoding of i , with t, u and d as in Definition 7, and let $\mathbf{pk}_i$ be as in (14). Then*

- *Derive performs $O(tn \log n)$ group operations and outputs $\mathbf{pk}_i$ of size $O(tn)$.*
- *Prove performs $O(k(t+V) + t+u+d)$ operations in $\mathbb{G}_1$ and $O(k(t+ud+V) + \log n)$ in $\mathbb{F}$, where k is the number of non-zero entries of $\mathbf{a}$.*

Moreover Theorem 3 continues to hold with $\mathbf{pk}_i$ as in (14).

The proof of this result is an immediate consequence of Lemmas 5 through 8.

Therefore, obtaining a sparse zkSNARK for the linear relation corresponding to an index i amounts to providing a rational encoding for the latter, that is, specifying the index polynomials $\mathcal{E}$, the monomials $\mathcal{X}$ and the scalar maps α, β of Definition 7, and pairing it with the general protocol of Figure 3. We do this for Plonk in Section 4.3, where $t = 3$, $u = 3$ and $d = 2$, so that both prover and verifier are optimal, and for V-R1CS-lite in Section 4.4, where $t = 3V + 1$, $V = O(\log n)$, $u = 0$ and $d = 1$.

4.3 Instantiation for the Linear Relation of Plonk

In this section, we provide the rational encoding and derive the corresponding sparse zkSNARK for the linear relation arising in the Plonk constraint system (we will justify why this is the linear relation in Section 6.1). It consists of a permutation and two shifts, as specified next.

Definition 8 (Plonk's Linear Relation). *The Plonk linear relation is*

$$\mathcal{R}_{\text{ck}}^{\text{Plonk, lin}} = \left\{ \left(i = (\mathbb{F}, n, \sigma) \right. \right. \\ \left. \left. \begin{array}{l} \mathbf{x} = [w, f_\omega, f_{\omega^2}]_1 \\ \mathbf{w} = (\mathbf{w}, \mathbf{f}_\omega, \mathbf{f}_{\omega^2}, \rho_w, \rho_{f_\omega}, \rho_{f_{\omega^2}}) \end{array} \right) \right\} \left\{ \begin{array}{l} \sigma : [n] \rightarrow [n] \text{ permutation} \\ w_\ell = w_{\sigma(\ell)}, \quad \forall \ell \in [n] \\ \mathbf{f}_\omega = (w_2, w_3, \dots, w_n, w_1) \\ \mathbf{f}_{\omega^2} = (w_3, w_4, \dots, w_1, w_2) \\ \text{Com}(\text{ck}, \mathbf{w}, \rho_w) = [w]_1 \\ \text{Com}(\text{ck}, \mathbf{f}_\omega, \rho_{f_\omega}) = [f_\omega]_1 \\ \text{Com}(\text{ck}, \mathbf{f}_{\omega^2}, \rho_{f_{\omega^2}}) = [f_{\omega^2}]_1 \end{array} \right\}.$$

Clearly, this is a linear relation: let $M_\sigma, M_\omega, M_{\omega^2} \in \mathbb{F}^{n \times n}$ be given by

$$(M_\sigma)_{\ell, \sigma(\ell)} = (M_\omega)_{\ell, \ell+1} = (M_{\omega^2})_{\ell, \ell+2} = 1 \quad \text{for } \ell \in [n],$$

all other entries being zero and all indices taken modulo n , so that $(M_\sigma \mathbf{w})_\ell = w_{\sigma(\ell)}$, $(M_\omega \mathbf{w})_\ell = w_{\ell+1}$ and $(M_{\omega^2} \mathbf{w})_\ell = w_{\ell+2}$. Definition 8 then reads

$$\mathbf{w} = M_\sigma \mathbf{w}, \quad \mathbf{f}_\omega = M_\omega \mathbf{w}, \quad \mathbf{f}_{\omega^2} = M_{\omega^2} \mathbf{w},$$

which is the special case of $\mathcal{R}_{\text{ck}}^{\text{lin}}$ with $l = 3$, $\mathbf{a} = \mathbf{w}$, $\mathbf{M} = (M_\sigma, M_\omega, M_{\omega^2})$ and $(\mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \mathbf{b}^{(3)}) = (\mathbf{w}, \mathbf{f}_\omega, \mathbf{f}_{\omega^2})$. Here $\mathbf{b}^{(1)} = \mathbf{a}$ and $\rho_{b^{(1)}} = \rho_w$, so the instance carries three group elements rather than four, and the term $\sum_{j=1}^3 \delta^{j-1} [b^{(j)}(\tau)]_1$ of (7) is $[w(\tau)]_1 + \delta[f_\omega(\tau)]_1 + \delta^2[f_{\omega^2}(\tau)]_1$. Each of the three matrices has at most one non-zero entry per column and is therefore basic in the sense of Definition 6, so $V^{(1)} = V^{(2)} = V^{(3)} = 1$ and $V = 3$; in column ℓ that entry sits in row $\sigma^{-1}(\ell)$, $\ell - 1$ and $\ell - 2$ respectively.

The rational functions $C_{\delta,\ell}(Y)$. We instantiate the rational function of Equation (3) with $l = 3$ and $\mathbf{M} = (M_\sigma, M_\omega, M_{\omega^2})$, which gives

$$C_{\delta,\ell}(Y) = \sum_{i:(M_\sigma)_{i\ell} \neq 0} \frac{h_i}{Y - h_i} + \delta \sum_{i:(M_\omega)_{i\ell} \neq 0} \frac{h_i}{Y - h_i} + \delta^2 \sum_{i:(M_{\omega^2})_{i\ell} \neq 0} \frac{h_i}{Y - h_i},$$

since all non-zero entries of each matrix are equal to 1. Each of the three matrices contributes exactly one summand, in rows $\sigma^{-1}(\ell)$, $\ell - 1$ and $\ell - 2$, respectively. Given that $h_{\ell-1} = \omega^{-1}h_\ell$, $h_{\ell-2} = \omega^{-2}h_\ell$, we have

$$\begin{aligned} C_{\delta,\ell}(Y) &= \frac{h_{\sigma^{-1}(\ell)}}{Y - h_{\sigma^{-1}(\ell)}} + \delta \frac{\omega^{-1}h_\ell}{Y - \omega^{-1}h_\ell} + \delta^2 \frac{\omega^{-2}h_\ell}{Y - \omega^{-2}h_\ell} \\ &= \frac{h_{\sigma^{-1}(\ell)}}{Y - h_{\sigma^{-1}(\ell)}} + \delta \frac{h_\ell}{\omega Y - h_\ell} + \delta^2 \frac{h_\ell}{\omega^2 Y - h_\ell}. \end{aligned} \quad (15)$$

Expanding and writing $C_{\delta,\ell}(Y)$ as a single fraction will give us precisely its rational encoding (see Definition 7). The common denominator is the product of the three linear factors, of degree 3 in Y , and the numerator has degree 2. Expanding them will produce coefficients that are the monomials $1, h_\ell, h_\ell^2$ and the products $h_{\sigma^{-1}(\ell)}, h_{\sigma^{-1}(\ell)}h_\ell, h_{\sigma^{-1}(\ell)}h_\ell^2$. The latter three depend on σ (and correspond to the set $\mathcal{E}$ from the rational encoding) and must be preprocessed, so we let $e_1(X), e_2(X), e_3(X)$ be the polynomials of degree at most $n - 1$ such that for $\ell \in [n]$

$$e_1(h_\ell) = h_{\sigma^{-1}(\ell)}, \quad e_2(h_\ell) = h_{\sigma^{-1}(\ell)}h_\ell, \quad e_3(h_\ell) = h_{\sigma^{-1}(\ell)}h_\ell^2. \quad (16)$$

The three coefficients $1, h_\ell, h_\ell^2$ are the polynomials $1, X, X^2$ (corresponding to the set $\mathcal{X}$ of monomials from the rational encoding) evaluated at h_ℓ and need no preprocessing.

Lemma 9 (Plonk's rational encoding). *Let $\mathcal{E} = (e_1(X), e_2(X), e_3(X))$ and $\mathcal{X} = (1, X, X^2)$, so that*

$$(f_1(X), \dots, f_6(X)) = (e_1(X), e_2(X), e_3(X), 1, X, X^2),$$

and let

$$\hat{I}_r(X) = \sum_{s=1}^6 \alpha_s(r) f_s(X), \quad \hat{R}_{\delta,r}(X) = \sum_{s=1}^6 \beta_s(r, \delta) f_s(X), \quad (17)$$

where

$$\begin{aligned} \alpha_1(r) &= -\omega^3 r^2, & \alpha_2(r) &= (\omega + \omega^2)r, & \alpha_3(r) &= -1, \\ \alpha_4(r) &= \omega^3 r^3, & \alpha_5(r) &= -(\omega + \omega^2)r^2, & \alpha_6(r) &= r, \\ \beta_1(r, \delta) &= \omega^3 r^2, & \beta_2(r, \delta) &= -((\omega + \omega^2) + \delta\omega^2 + \delta^2\omega)r, & \beta_3(r, \delta) &= 1 + \delta + \delta^2, \\ \beta_4(r, \delta) &= 0, & \beta_5(r, \delta) &= (\delta\omega^2 + \delta^2\omega)r^2, & \beta_6(r, \delta) &= -(\delta + \delta^2)r. \end{aligned}$$

Then $\text{RE}^{\text{Plonk}} = (\mathcal{E}, \mathcal{X}, \alpha, \beta)$ is a rational encoding of i in the sense of Definition 7, with $t = 3$, $u = 3$ and $d = 2$.

We omit the proof, as it simply consists in basic algebraic manipulations. One only needs to evaluate at $Y = r$ in Equation (15), expand, and group the terms accordingly.

Since t , u and d are constant, Theorem 3 and Proposition 7 specialise as follows.

Corollary 1 (Plonk’s sparse lincheck). *The protocol of Figure 3 instantiated with RE^{Plonk} is a sparse zkSNARK for $\mathcal{R}_{\text{ck}}^{\text{lin,Plonk}}$ in the AGM and the ROM under the (n, n) -PDL assumption. The complexities are as follows.*

- Derive performs $O(n \log n)$ group operations and outputs pk_i of size $O(n)$ and vk_i of size $O(1)$.
- Prove performs $O(k)$ operations in $\mathbb{G}_1$ and $O(k + \log n)$ in $\mathbb{F}$, where k is the number of non-zero entries of w .
- Verify performs $O(1)$ group operations, $O(\log n)$ field operations and 6 pairings.
- A proof consists of 5 elements of $\mathbb{G}_1$ and 1 of $\mathbb{F}$.

Writing $q_\ell^{(s)}(X) = \lfloor \lambda_\ell(X) e_s(X) / \nu_\mathbb{H}(X) \rfloor$ for $s \in \{1, 2, 3\}$, exactly as in Remark 1, the keys (14) and (9) become¹²

$$\text{pk}_i = \left([\{\tau^i\}_{i=0}^n, e_1(\tau), e_2(\tau), e_3(\tau), \{\lambda_\ell(\tau), G_\ell(\tau), \{q_\ell^{(s)}(\tau)\}_{s=1}^3\}_{\ell=1}^n]_1 \right) \quad (18)$$

$$\text{vk}_i = \left([1, \tau^2]_1, [1, \tau, \tau^2, \nu_\mathbb{H}(\tau), e_1(\tau), e_2(\tau), e_3(\tau)]_2 \right). \quad (19)$$

The quotients associated with $\mathcal{X}$ are supplied by Remark 1 and do not appear in pk_i , while the elements $[1, \tau, \tau^2]_2$ that the verifier needs for the monomial part of $[\hat{I}_r(\tau)]_2$ and $[\hat{R}_{\delta,r}(\tau)]_2$ are already in vk_i .

Theorem 2 and Corollary 1 directly yield a semi-dynamic zkSNARK for $\mathcal{R}_{\text{ck}}^{\text{lin,Plonk}}$.

Corollary 2 (Plonk’s semi-dynamic lincheck). *There exists a semi-dynamic zkSNARK for $\mathcal{R}_{\text{ck}}^{\text{lin,Plonk}}$ with the following complexities.*

- Derive performs $O(n \log n)$ operations.
- Prove performs $O(n)$ operations.
- BuildData performs $O(n)$ operations¹³.
- Update performs $O(k + \log n)$ operations, where k is the Hamming distance between the witnesses.
- The verifier performs $O(\log n)$ field operations and 12 pairings.
- The proof consists of 13 elements of $\mathbb{G}_1$ and 2 of $\mathbb{F}$.

In order to compile this into a fully-dynamic zkSNARK with Theorem 1, we need to show that BuildData can be computed step-wise. In this case, BuildData consists in running the sparse prover from Corollary 1 (see Figure 1). This can be done, as we have just seen that the prover is basically computing MSM against precomputed group elements—which admit a step-wise computation. We provide the details in Section C.

Corollary 3 (Plonk’s fully dynamic lincheck). *There exists a fully dynamic zkSNARK for $\mathcal{R}_{\text{ck}}^{\text{lin,Plonk}}$ with update time $O(k\sqrt{n \log n})$ in the de-amortized setting. The rest of the complexities are identical to those of Corollary 2.*

4.4 Instantiation for the Linear Relation of V-R1CS-lite

In this section, we specialize our construction for the linear relation arising in V-R1CS-lite. Again, the justification for the specific relation will become clear in Section 6.2.

Definition 9. *The V-R1CS-lite linear relation $\mathcal{R}_{\text{ck}}^{\text{lin}, V\text{-R1CS-lite}}$*

$$\mathcal{R}_{\text{ck}}^{\text{lin}, V\text{-R1CS-lite}} = \left\{ \left(\begin{array}{l} i = (\mathbb{F}, n, V, A, B) \\ x = [z, z_A, z_B]_1 \\ w = (z, z_A, z_B, \rho_z, \rho_A, \rho_B) \end{array} \right) \mid \begin{array}{l} \begin{pmatrix} A \\ B \end{pmatrix} \in \mathcal{M}_{2n, n}(V) \\ z, z_A, z_B \in \mathbb{F}^n \\ z_A = Az \\ z_B = Bz \\ \text{Com}(\text{ck}, z, \rho_z) = [z]_1 \\ \text{Com}(\text{ck}, z_A, \rho_A) = [z_A]_1 \\ \text{Com}(\text{ck}, z_B, \rho_B) = [z_B]_1 \end{array} \right\}. \quad (20)$$

Clearly, Definition 9 is a special case of $\mathcal{R}_{\text{ck}}^{\text{lin}}$ with $l = 2$, $\mathbf{a} = \mathbf{z}$, $\mathbf{M} = (A, B)$ and $(\mathbf{b}^{(1)}, \mathbf{b}^{(2)}) = (z_A, z_B)$.

¹² Technically, the prover does not need all the powers of $\{\tau^i\}_{i=0}^n$ after the precomputation. Since in our final dynamic zkSNARK construction the prover still needs to store the whole KZG commitment key, we keep it this way.

¹³ Both Prove and BuildData run in time proportional to the Hamming weight of the witness, but we write the general, dense case.

Encoding the stacked matrix. Following [RZ21] we decompose the stacked matrix $\begin{pmatrix} A \\ B \end{pmatrix} = \sum_{i=1}^V \begin{pmatrix} A_i \\ B_i \end{pmatrix}$ into V basic matrices, which is possible precisely because every column of the stack has at most V non-zero entries. Since a basic matrix is described by one value and one row index per column, the decomposition is captured by $2V$ functions $\{\mathbf{v}_i : \mathbb{H} \rightarrow \mathbb{F}, \mathbf{f}_i : \mathbb{H} \rightarrow [2n]\}_{i \in [V]}$, where $\mathbf{v}_i(\mathbf{h}_\ell) = \begin{pmatrix} A_i \\ B_i \end{pmatrix}_{\mathbf{f}_i(\mathbf{h}_\ell), \ell}$ is the non-zero entry of column ℓ of the i -th basic matrix and $\mathbf{f}_i(\mathbf{h}_\ell)$ its row index in the stack. The first n rows of the stack correspond to A and the second to B , so we split $[V]$ per column as

$$\mathcal{V}_\ell^A = \{i \in [V] \mid 1 \leq \mathbf{f}_i(\mathbf{h}_\ell) \leq n\}, \quad \mathcal{V}_\ell^B = \{i \in [V] \mid n+1 \leq \mathbf{f}_i(\mathbf{h}_\ell) \leq 2n\},$$

and write the corresponding sets of row indices in $[n]$ as $S_\ell^A = \{\mathbf{f}_i(\mathbf{h}_\ell) \mid i \in \mathcal{V}_\ell^A\}$ and $S_\ell^B = \{\mathbf{f}_i(\mathbf{h}_\ell) - n \mid i \in \mathcal{V}_\ell^B\}$. Their union $S_\ell = S_\ell^A \cup S_\ell^B$ has at most V elements.

The rational functions $C_{\delta, \ell}(Y)$. We instantiate the rational function of Equation (3) with $l = 2$ and $\mathbf{M} = (A, B)$, which gives

$$C_{\delta, \ell}(Y) = \sum_{i: A_{i\ell} \neq 0} \frac{\mathbf{h}_i}{Y - \mathbf{h}_i} A_{i\ell} + \delta \sum_{i: B_{i\ell} \neq 0} \frac{\mathbf{h}_i}{Y - \mathbf{h}_i} B_{i\ell}.$$

Grouping the non-zero entries of column ℓ by the basic matrix they come from, and recalling that $\mathbf{f}_i(\mathbf{h}_\ell)$ indexes a row of A when $i \in \mathcal{V}_\ell^A$ and a row of B shifted by n when $i \in \mathcal{V}_\ell^B$,

$$C_{\delta, \ell}(Y) = \sum_{i \in \mathcal{V}_\ell^A} \mathbf{v}_i(\mathbf{h}_\ell) \frac{\mathbf{h}_{\mathbf{f}_i(\mathbf{h}_\ell)}}{Y - \mathbf{h}_{\mathbf{f}_i(\mathbf{h}_\ell)}} + \delta \sum_{i \in \mathcal{V}_\ell^B} \mathbf{v}_i(\mathbf{h}_\ell) \frac{\mathbf{h}_{\mathbf{f}_i(\mathbf{h}_\ell) - n}}{Y - \mathbf{h}_{\mathbf{f}_i(\mathbf{h}_\ell) - n}}. \quad (21)$$

Every pole of (21) is of the form $\mathbf{h}_s$ with $s \in S_\ell$. The common denominator is $\hat{I}_\ell(Y) = \prod_{s \in S_\ell} (Y - \mathbf{h}_s)$, of degree at most V . As in [RZ21], we define

$$\begin{aligned} \hat{I}_\ell(Y) &= \prod_{s \in S_\ell} (Y - \mathbf{h}_s) = \sum_{j=0}^V \hat{I}_{\ell j} Y^j, \\ \hat{R}_\ell^A(Y) &= \sum_{i \in \mathcal{V}_\ell^A} \mathbf{v}_i(\mathbf{h}_\ell) \mathbf{h}_{\mathbf{f}_i(\mathbf{h}_\ell)} \prod_{s \in S_\ell \setminus \{\mathbf{f}_i(\mathbf{h}_\ell)\}} (Y - \mathbf{h}_s) = \sum_{j=0}^{V-1} \hat{R}_{\ell j}^A Y^j, \\ \hat{R}_\ell^B(Y) &= \sum_{i \in \mathcal{V}_\ell^B} \mathbf{v}_i(\mathbf{h}_\ell) \mathbf{h}_{\mathbf{f}_i(\mathbf{h}_\ell) - n} \prod_{s \in S_\ell \setminus \{\mathbf{f}_i(\mathbf{h}_\ell) - n\}} (Y - \mathbf{h}_s) = \sum_{j=0}^{V-1} \hat{R}_{\ell j}^B Y^j, \end{aligned}$$

so that $C_{\delta, \ell}(Y) \hat{I}_\ell(Y) = \hat{R}_\ell^A(Y) + \delta \hat{R}_\ell^B(Y)$. Unlike the Plonk case, the coefficients $\hat{I}_{\ell j}, \hat{R}_{\ell j}^A, \hat{R}_{\ell j}^B$ have no closed form in $\mathbf{h}_\ell$ for a general index, so all of them must be preprocessed. We pack them into the polynomials of degree at most $n-1$

$$v_j^{\hat{I}}(X) = \sum_{\ell=1}^n \hat{I}_{\ell j} \lambda_\ell(X), \quad v_j^{\hat{R}, A}(X) = \sum_{\ell=1}^n \hat{R}_{\ell j}^A \lambda_\ell(X), \quad v_j^{\hat{R}, B}(X) = \sum_{\ell=1}^n \hat{R}_{\ell j}^B \lambda_\ell(X). \quad (22)$$

Lemma 10 (V-R1CS-lite rational encoding). *Let $\mathcal{E} = (\{v_j^{\hat{I}}\}_{j=0}^V, \{v_j^{\hat{R}, A}, v_j^{\hat{R}, B}\}_{j=0}^{V-1})$, let $\mathcal{X}$ be empty, and let*

$$\begin{aligned} \hat{I}_r(X) &= \sum_{j=0}^V \alpha_{\hat{I}, j}(r) v_j^{\hat{I}}(X) + \sum_{j=0}^{V-1} \left(\alpha_{A, j}(r) v_j^{\hat{R}, A}(X) + \alpha_{B, j}(r) v_j^{\hat{R}, B}(X) \right), \\ \hat{R}_{\delta, r}(X) &= \sum_{j=0}^V \beta_{\hat{I}, j}(r, \delta) v_j^{\hat{I}}(X) + \sum_{j=0}^{V-1} \left(\beta_{A, j}(r, \delta) v_j^{\hat{R}, A}(X) + \beta_{B, j}(r, \delta) v_j^{\hat{R}, B}(X) \right), \end{aligned} \quad (23)$$

where $\alpha_{\hat{I}, j}(r) = r^j$ and $\alpha_{A, j} = \alpha_{B, j} = 0$, while $\beta_{\hat{I}, j} = 0$, $\beta_{A, j}(r, \delta) = r^j$ and $\beta_{B, j}(r, \delta) = \delta r^j$; that is,

$$\hat{I}_r(X) = \sum_{j=0}^V r^j v_j^{\hat{I}}(X), \quad \hat{R}_{\delta, r}(X) = \sum_{j=0}^{V-1} r^j (v_j^{\hat{R}, A}(X) + \delta v_j^{\hat{R}, B}(X)).$$

Then $\text{RE}^{V\text{-R1CS-lite}} = (\mathcal{E}, \mathcal{X}, \alpha, \beta)$ is a rational encoding of i in the sense of Definition 7, with $t = 3V + 1$, $u = 0$ and $d = 1$.

We omit the proof, as it is almost identical to the one from [RZ21].

Theorem 3 and Proposition 7 specialise as follows.

Corollary 4 (V-R1CS-lite sparse lincheck). *The protocol of Figure 3 instantiated with $\text{RE}^{V\text{-R1CS-lite}}$ is a sparse zkSNARK for $\mathcal{R}_{\text{ck}}^{\text{lin}, V\text{-R1CS-lite}}$ in the AGM and the ROM under the (n, n) -PDL assumption. The complexities are as follows.*

- Derive performs $O(Vn \log n)$ group operations and outputs pk_i of size $O(nV)$ and vk_i of size $O(V)$.
- Prove performs $O(kV)$ operations in $\mathbb{G}_1$ and $O(kV + \log n)$ in $\mathbb{F}$, where k is the number of non-zero entries of z .
- Verify performs $O(V)$ group operations, $O(\log n)$ field operations and 6 pairings.
- A proof consists of 5 elements of $\mathbb{G}_1$ and 1 of $\mathbb{F}$.

In particular, the verifier is succinct when $V = O(\log n)$.

Writing $q_{\ell,j}^{\hat{I}}(X) = \lfloor \lambda_{\ell}(X) v_j^{\hat{I}}(X) / \nu_{\mathbb{H}}(X) \rfloor$ and similarly for $q_{\ell,j}^{\hat{R},A}(X)$ and $q_{\ell,j}^{\hat{R},B}(X)$, the keys (14) and (9) become

$$\text{pk}_i = \left([1, \tau, \dots, \tau^n]_1, [\{v_j^{\hat{I}}(\tau)\}_{j=0}^V, \{v_j^{\hat{R},A}(\tau), v_j^{\hat{R},B}(\tau)\}_{j=0}^{V-1}]_1 \right), \quad (24)$$

$$\text{vk}_i = \left([1, \tau^2]_1, [1, \tau, \tau^2, \nu_{\mathbb{H}}(\tau), \{v_j^{\hat{I}}(\tau)\}_{j=0}^V, \{v_j^{\hat{R},A}(\tau), v_j^{\hat{R},B}(\tau)\}_{j=0}^{V-1}]_2 \right). \quad (25)$$

The sparse zkSNARK can be transformed into a semi-dynamic zkSNARK and a fully-dynamic zkSNARK in the de-amortized setting just like in the Plonk case.

Corollary 5 (V-R1CS-lite semi-dynamic lincheck). *There exists a semi-dynamic zkSNARK for $\mathcal{R}_{\text{ck}}^{\text{lin}, V\text{-R1CS-lite}}$ with the following complexities.*

- Derive performs $O(Vn \log n)$ operations.
- Prove performs $O(nV)$ operations.
- BuildData performs $O(nV)$ operations.
- Update performs $O(kV + \log n)$ operations, where k is the Hamming distance between the witnesses.
- The verifier performs $O(V + \log n)$ operations and 12 pairings.
- The proof consists of 13 elements of $\mathbb{G}_1$ and 2 of $\mathbb{F}$.

Corollary 6 (V-R1CS-lite fully dynamic lincheck). *There exists a fully dynamic zkSNARK for $\mathcal{R}_{\text{ck}}^{\text{lin}, \text{Plonk}}$ with update time $O(kV\sqrt{n \log n})$ in the de-amortized setting. The rest of the complexities are identical to those of Corollary 5.*

5 Dynamic SNARKs for the Hadamard and Gate Relations

In this section we present our final building blocks: a dynamic SNARK for the Hadamard relation and for the gate relation of Plonk. Our protocol for Hadamard products is essentially the one of [ZOZ26] and is given mostly for completeness. The only modification is that, while [ZOZ26] is tailored to the trusted setup of Groth16 [Gro16], our construction is adapted to the KZG trusted setup. The the gate relation of Plonk can be interpreted as a sum of Hadamard products, but in this case it will be *linear in the witness*, leading to a simpler, more efficient dynamic SNARK.

5.1 Dynamic SNARK for the Hadamard Relation

As in [ZOZ26], we will start by presenting a semi-dynamic SNARK for the Hadamard relation, which is then turned into a dynamic one.

Definition 10 (Hadamard Relation). We define the Hadamard relation as

$$\mathcal{R}_{\text{ck}}^{\text{had}} = \left\{ \left(\mathbf{x} = ([a, c]_1, [b]_2), \mathbf{w} = (\mathbf{a}, \mathbf{b}, \mathbf{c}, \rho_a, \rho_b, \rho_c) \right) \mid \begin{array}{l} \mathbf{a} \circ \mathbf{b} = \mathbf{c} \\ \text{Com}(\text{ck}, \mathbf{a}, \rho_a) = [a]_1 \\ \text{Com}(\text{ck}, \mathbf{b}, \rho_b) = [b]_2 \\ \text{Com}(\text{ck}, \mathbf{c}, \rho_c) = [c]_1 \end{array} \right\}$$

Semi-dynamic Hadamard. The Hadamard relation $\mathbf{a} \circ \mathbf{b} = \mathbf{c}$ is captured by the polynomial identity $a(X)b(X) - c(X) = q(X)\nu_{\mathbb{H}}(X)$, where, for $v \in \{a, b, c\}$, $v(X) = \sum_{\ell=1}^n v_{\ell}\lambda_{\ell}(X) + \rho_v\nu_{\mathbb{H}}(X)$. Accordingly, the proof consists of the single group element $\pi = [q(\tau)]_1$, which the verifier checks by testing $e([a]_1, [b]_2) - e([c]_1, [1]_2) = e(\pi, [\nu_{\mathbb{H}}(\tau)]_2)$.

Let $(\mathbf{a}', \mathbf{b}', \mathbf{c}')$ be an updated witness with corresponding polynomials $v'(X) = \sum_{\ell=1}^n v'_{\ell}\lambda_{\ell}(X) + \rho'_v\nu_{\mathbb{H}}(X)$, $v \in \{a, b, c\}$. We describe how to update the proof π for $\mathbf{x} = ([a, c]_1, [b]_2)$ into a proof π' for $\mathbf{x}' = ([a', c']_1, [b']_2)$. Define $v^*(X) = v'(X) - v(X)$, $\rho_v^* = \rho'_v - \rho_v$, and let $I_v = \{\ell \in [n] : v_{\ell} \neq v'_{\ell}\}$ and $I = I_a \cup I_b$. We split the left term of the updated Hadamard polynomial identity as follows:

$$\begin{aligned} a'(X)b'(X) - c'(X) &= (a(X) + a^*(X))(b(X) + b^*(X)) - c(X) - c^*(X) \\ &= \underbrace{a(X)b(X) - c(X)}_{\text{original identity}} + \underbrace{a^*(X)b(X) + a(X)b^*(X) - c^*(X)}_{\text{projection terms}} + \underbrace{a^*(X)b^*(X)}_{\text{cross term}}. \end{aligned}$$

To commit to the quotient $q'(X) = \frac{a'(X)b'(X) - c'(X)}{\nu_{\mathbb{H}}(X)}$, by the linearity of quotients in polynomial division and the additively homomorphic property of KZG, $[q'(\tau)]_1$ can be computed from $[q(\tau)]_1$ by adding a commitment to the quotients of the projection and cross terms divided by $\nu_{\mathbb{H}}(X)$. We detail how to compute these below.

Projection terms. It follows from the exposition that the quotients associated to the projection terms are $q_a^*(X) + q_b^*(X) - \rho_c^*$, where

$$q_a^*(X) = \left\lfloor \frac{a^*(X)b(X)}{\nu_{\mathbb{H}}(X)} \right\rfloor, \quad q_b^*(X) = \left\lfloor \frac{b^*(X)a(X)}{\nu_{\mathbb{H}}(X)} \right\rfloor.$$

Since

$$q_a^*(X) = \sum_{i \in I_a} a^*(\mathbf{h}_i) \left\lfloor \frac{\lambda_i(X)b(X)}{\nu_{\mathbb{H}}(X)} \right\rfloor + \rho_a^*b(X),$$

a commitment to $q_a^*(X)$ can be computed in time $O(|I_a|)$, assuming the prover stores $[b(\tau)]_1$ and precomputed quotients $\{[q_{\ell}^b(\tau)]_1\}_{\ell=1}^n$, for $q_{\ell}^b(X) = \left\lfloor \frac{\lambda_{\ell}(X)b(X)}{\nu_{\mathbb{H}}(X)} \right\rfloor$. The commitment to $q_b^*(X)$ is computed analogously. Moreover, the preprocessing can be efficiently computed in $O(n \log n)$ [ZOZ26]. In total, the prover needs to store $2n + 1$ $\mathbb{G}_1$ group elements to update the projection terms in $O(|I|)$ scalar multiplications.

The cross-term. We now explain how to commit to the quotient of the cross term, $q^{\text{cross}}(X)$. We have

$$q^{\text{cross}}(X) = \left\lfloor \frac{a^*(X)b^*(X)}{\nu_{\mathbb{H}}(X)} \right\rfloor = \sum_{(i,j) \in I_a \times I_b} a^*(\mathbf{h}_i)b^*(\mathbf{h}_j) \left\lfloor \frac{\lambda_i(X)\lambda_j(X)}{\nu_{\mathbb{H}}(X)} \right\rfloor + \rho_a^*b^*(X) + \rho_b^*a^*(X) - \rho_a^*\rho_b^*\nu_{\mathbb{H}}(X).$$

We can compute the commitment to the last three summands in time $O(|I|)$, assuming we have precomputed $\{[\lambda_{\ell}(\tau)]_{\ell=1}^n, \nu_{\mathbb{H}}(\tau)]_1$. The polynomial $q_1^{\text{cross}}(X) = \sum_{(i,j) \in I_a \times I_b} a^*(\mathbf{h}_i)b^*(\mathbf{h}_j) \left\lfloor \frac{\lambda_i(X)\lambda_j(X)}{\nu_{\mathbb{H}}(X)} \right\rfloor$ is not sparse in the Lagrange basis even if both $a^*(X)$ and $b^*(X)$ are sparse. The main innovation of Dynark is the theorem below:

Theorem 4 ([ZOZ26]). *Let $i, j \in [n]$. Then, the following equality holds:*

$$\left\lfloor \frac{\lambda_i(X)\lambda_j(X)}{\nu_{\mathbb{H}}(X)} \right\rfloor = \begin{cases} \frac{h_{j-i}}{n(1-h_{j-i})}\lambda_i(X) + \frac{h_{i-j}}{n(1-h_{i-j})}\lambda_j(X) =: \phi_{i,j}(X) & \text{if } i \neq j \\ -\sum_{j \in [n] \setminus \{i\}} \phi_{i,j}(X) =: \phi_{i,i}(X) & \text{otherwise} \end{cases}$$

It follows that if $\{\phi_{\ell,\ell}(\tau), \lambda_{\ell}(\tau)\}_{\ell=1}^n$ are preprocessed, the commitment to $q_1^{\text{cross}}(X)$ can be computed in time $O(|I| \log^2 |I|)$ via sparse convolutions. Similar techniques can be used in the preprocessing to achieve a complexity of $O(n \log n)$.

The full protocol is given in Figure 5.

• KeyGen(ck) $\rightarrow$ srs: output srs = ck. // ck = $([1, \tau, \dots, \tau^n]_1, [1, \tau, \dots, \tau^n]_2)$ • Derive(srs, $i = (\mathbb{F}, n)$) $\rightarrow$ (pk_i, vk_i): output pk_i = $(\text{ck}, [\nu_{\mathbb{H}}(\tau), \{\lambda_{\ell}(\tau), \phi_{\ell,\ell}(\tau)\}_{\ell=1}^n]_1)$ and vk_i = $[1, \nu_{\mathbb{H}}(\tau)]_2$ for $\phi_{\ell,\ell}(X)$ as in Theorem 4.
• Prove(pk_i, x, w) $\rightarrow$ (π, rand) 1. For $v \in \{a, b, c\}$, let $v(X) = \sum_{\ell=1}^n v_{\ell} \lambda_{\ell}(X) + \rho_v \nu_{\mathbb{H}}(X)$. 2. Output $(\pi, \text{rand}) = ([q(\tau)]_1, \perp)$ for $q(X) = (a(X)b(X) - c(X))/\nu_{\mathbb{H}}(X)$. • BuildData(pk_i, x, w, rand) $\rightarrow$ data 1. Compute $[b]_1 = \sum_{\ell=1}^n b_{\ell} [\lambda_{\ell}(\tau)]_1 + \rho_b [\nu_{\mathbb{H}}(\tau)]_1$. 2. For $v \in \{a, b\}$ and $\ell \in [n]$, compute $[q_{\ell}^v(\tau)]_1$, where $q_{\ell}^v(X) = [\lambda_{\ell}(X)v(X)/\nu_{\mathbb{H}}(X)]$. 3. Output $\text{data} = [b, \{q_{\ell}^a(\tau), q_{\ell}^b(\tau)\}_{\ell=1}^n]_1$. • Update(pk_i, x, w, x', w', π, data) $\rightarrow$ π' 1. Parse $\text{data} = [b, \{q_{\ell}^a(\tau), q_{\ell}^b(\tau)\}_{\ell=1}^n]_1$ and $\pi = [q(\tau)]_1$. 2. For $v \in \{a, b\}$ let $\mathbf{v}^* = \mathbf{v}' - \mathbf{v}$, and for $v \in \{a, b, c\}$ let $\rho_v^* = \rho_v' - \rho_v$. Let $I = \text{Supp}(\mathbf{a}^*) \cup \text{Supp}(\mathbf{b}^*)$. 3. Compute $[q_a^*(\tau)]_1 = \sum_{i \in \text{Supp}(\mathbf{a}^*)} a_i^* [q_i^b(\tau)]_1 + \rho_a^* [b]_1$. 4. Compute $[q_b^*(\tau)]_1 = \sum_{j \in \text{Supp}(\mathbf{b}^*)} b_j^* [q_j^a(\tau)]_1 + \rho_b^* [a]_1$. 5. For every $i \in I$, compute $\alpha_i = a_i^* b_i^*$ and $\beta_i = \rho_a^* b_i^* + \rho_b^* a_i^* + \sum_{j \in I \setminus \{i\}} (a_i^* b_j^* + a_j^* b_i^*) \frac{h_{j-i}}{n(1-h_{j-i})}.$ 6. Compute $[q^{\text{cross}}(\tau)]_1 = \sum_{i \in I} (\alpha_i [\phi_{i,i}(\tau)]_1 + \beta_i [\lambda_i(\tau)]_1) + \rho_a^* \rho_b^* [\nu_{\mathbb{H}}(\tau)]_1$. 7. Output $\pi' = [q(\tau)]_1 + [q_a^*(\tau)]_1 + [q_b^*(\tau)]_1 + [q^{\text{cross}}(\tau)]_1 - \rho_c^* [1]_1$. • Verify(vk_i, x, π) $\rightarrow$ 0/1 1. Accept if and only if $e([a]_1, [b]_2) - e([c]_1, [1]_2) \stackrel{?}{=} e(\pi, [\nu_{\mathbb{H}}(\tau)]_2)$.

Fig. 5. Semi-dynamic SNARK for $\mathcal{R}_{\text{ck}}^{\text{had}}$.

Theorem 5 (Semi-dynamic Hadamard). *The protocol from Figure 5 is a semi-dynamic SNARK for $\mathcal{R}_{\text{ck}}^{\text{had}}$ in the AGM under the (n, n) -PDL assumption with the following complexities:*

- **Prove** performs $O(n)$ operations in $\mathbb{G}_1$ and $O(n \log n)$ in $\mathbb{F}$, and outputs a proof of a single $\mathbb{G}_1$ element.
- **BuildData** performs $O(n \log n)$ operations in $\mathbb{G}_1$, and outputs $2n + 1$ elements in $\mathbb{G}_1$.
- **Update** performs $O(|I|)$ operations in $\mathbb{G}_1$ and $O(|I| \log^2 |I|)$ in $\mathbb{F}$. The updated proof consists of a single $\mathbb{G}_1$ element.
- **Verify** performs 3 pairings.

In Figure 5 we have written a Derive algorithm. Technically, the Hadamard relation is not a universal and therefore Derive is not needed. We include it here to enforce consistency with the rest of the zkSNARKs, where the Lagrange polynomials and the rest of the public preprocessing was computed by Derive.

Remark 2 (On zero-knowledge). This protocol is not zero-knowledge, as the proof is a deterministic function on the witness. This is not a problem when we use this CP-SNARK in a bigger protocol, where the randomness to produce the commitments is sampled online and is not part of the witness.

Fully-dynamic Hadamard. To build their dynamic SNARK, [ZOZ26] notice that the preprocess algorithm can be computed using stepwise DFTs, which implies that they can use Theorem 1. However, instead of directly using the compiler as a black-box, they perform some optimizations to achieve asymptotic improvements. We refer to [ZOZ26] for the details.

Theorem 6 (Fully-dynamic Hadamard, [ZOZ26]). *There is a fully-dynamic SNARK for $\mathcal{R}_{\text{ck}}^{\text{had}}$ in the AGM under the (n, n) -PDL assumption with the following complexities in the de-amortized setting:*

- Prove performs $O(n)$ operations in $\mathbb{G}_1$ and $O(n \log n)$ in $\mathbb{F}$, and outputs a proof of a single $\mathbb{G}_1$ element.
- Preprocess performs $O(n \log n)$ operations in $\mathbb{G}_1$, and outputs $2n + 1$ elements in $\mathbb{G}_1$.
- Update performs $O(k\sqrt{n \log n})$ operations in $\mathbb{G}_1$ and $O(k \log^2 k)$ in $\mathbb{F}$, where k is the Hamming distance between $(\mathbf{x}, \mathbf{w})$ and $(\mathbf{x}', \mathbf{w}')$. The updated proof consists of a single $\mathbb{G}_1$ element.
- Verify performs 3 pairings.

5.2 Dynamic SNARK for the Gate Relation

We begin by defining the concrete relation. Its relationship with the Plonk constraint system will become clear in Section 6.1. As in previous sections, let $\mathbb{H} = \langle \omega \rangle$ be a multiplicative subgroup of order n , which we assume divisible by 3. Let $\mathbb{K} = \langle \gamma \rangle \subseteq \mathbb{H}$ be the subgroup of order $m = \frac{n}{3}$ generated by $\gamma = \omega^3$; we write $\{\mu_i(X)\}_{i=1}^m$ for its Lagrange basis, as opposed to $\{\lambda_\ell(X)\}_{\ell=1}^n$ for $\mathbb{H}$, and $\nu_{\mathbb{K}}(X) = X^m - 1$ for its vanishing polynomial.

Definition 11 (Gate Relation). *Let $n_0 \leq m$. The gates relation $\mathcal{R}_{\text{ck}}^{\text{gates}}$ consists of tuples $(i, \mathbf{x}, \mathbf{w})$ where the index is $i = (\mathbb{F}, n, n_0, (\mathbf{q}_L, \mathbf{q}_R, \mathbf{q}_M, \mathbf{q}_O, \mathbf{q}_C) \in (\mathbb{F}^m)^5)$, the instance is $\mathbf{x} = (\mathbf{x} \in \mathbb{F}^{n_0}, [w, f_\omega, \varphi, f_{\omega^2}]_1)$ and the witness is $\mathbf{w} = ((\mathbf{w}, \mathbf{f}_\omega, \varphi, \mathbf{f}_{\omega^2}) \in (\mathbb{F}^n)^4, (\rho_w, \rho_{f_\omega}, \rho_\varphi, \rho_{f_{\omega^2}}) \in \mathbb{F}^4)$. The tuple is in the relation if and only if $[v]_1 = \text{Com}(\text{ck}, \mathbf{v}, \rho_v)$ for every $v \in \{w, f_\omega, \varphi, f_{\omega^2}\}$ and, for all $i \in [0, m-1]$,*

$$q_{L,i+1}w_{3i+1} + q_{R,i+1}f_{\omega,3i+1} + q_{M,i+1}\varphi_{3i+1} + q_{O,i+1}f_{\omega^2,3i+1} + q_{C,i+1} = \chi_i, \quad (26)$$

where $\chi_i = x_{i+1}$ for $i < n_0$ and $\chi_i = 0$ for $i \geq n_0$.

Note that the public input $\mathbf{x}$ sits in the instance and is *not* committed: the verifier receives it in the clear and encodes it itself.

Computing the proof. For $v \in \{w, f_\omega, \varphi, f_{\omega^2}\}$ we write, as in the rest of the paper, $v(X) = \sum_{\ell=1}^n v_\ell \lambda_\ell(X) + \rho_v \nu_{\mathbb{H}}(X)$ for the randomized low-degree extension of $\mathbf{v}$ over $\mathbb{H}$, and we set

$$q_J(X) = \sum_{i=1}^m q_{J,i} \mu_i(X) \quad (J \in \{L, R, M, O, C\}), \quad x(X) = \sum_{i=1}^{n_0} x_i \mu_i(X),$$

so that q_J and x have degree at most $m-1$. Since $\gamma^i = \omega^{3i} = \mathbf{h}_{3i+1}$ lies in $\mathbb{H}$, $v(\gamma^i) = v_{3i+1}$; likewise $x(\gamma^i) = \chi_i$ for every $i \in [0, m-1]$. Consequently (26) says exactly that the polynomial

$$q_L(X)w(X) + q_R(X)f_\omega(X) + q_M(X)\varphi(X) + q_O(X)f_{\omega^2}(X) + q_C(X) - x(X)$$

vanishes on $\mathbb{K} = \{\gamma^i\}_{i=0}^{m-1}$, i.e. that it is divisible by $\nu_{\mathbb{K}}(X)$. The m gate equations are therefore equivalent to the single polynomial identity

$$q_L(X)w(X) + q_R(X)f_\omega(X) + q_M(X)\varphi(X) + q_O(X)f_{\omega^2}(X) + q_C(X) - x(X) = Q(X) \nu_{\mathbb{K}}(X) \quad (27)$$

for some $Q(X) \in \mathbb{F}[X]$. As $\deg(q_J(X)v(X)) \leq n + m - 1$, the quotient satisfies $\deg Q \leq n - 1$ and the commitment key ck allows us to send a commitment to $Q(X)$.

Assume that **Derive** outputs the commitments $[q_C(\tau)]_1$, $\{[\mu_i(\tau)]_1\}_{i=1}^{n_0}$ and $[q_L(\tau), q_R(\tau), q_M(\tau), q_O(\tau)]_2$. The proof for this relation is then simply $\pi = [Q(\tau)]_1$: the verifier computes $[x(\tau)]_1 = \sum_{i=1}^{n_0} x_i [\mu_i(\tau)]_1$ and checks (27) with the pairing equation

$$\begin{aligned} & e([w]_1, [q_L(\tau)]_2) + e([f_\omega]_1, [q_R(\tau)]_2) + e([\varphi]_1, [q_M(\tau)]_2) + e([f_{\omega^2}]_1, [q_O(\tau)]_2) \\ &= e([x(\tau)]_1 - [q_C(\tau)]_1, [1]_2) + e(\pi, [\nu_{\mathbb{K}}(\tau)]_2). \end{aligned} \quad (28)$$

Updating the proof. Note that (27) is completely linear with respect to the witness polynomials. In the terminology of [ZOZ26], this implies that the new terms requiring updates are entirely *projection terms*, with no cross term. Moreover, the projection terms are taken against the index polynomials $q_J(X)$, which are the same across all executions of the protocol. This will lead to a more efficient and simplified version of the protocol from Section 5.1, directly yielding a fully dynamic SNARK.

Let then (x', w') be an updated instance-witness pair, write $v'(X) = \sum_{\ell} v'_\ell \lambda_\ell(X) + \rho'_v \nu_{\mathbb{H}}(X)$ and $x'(X) = \sum_{i=1}^{n_0} x'_i \mu_i(X)$, and define the differences $v^*(X) = v'(X) - v(X)$, $\rho_v^* = \rho'_v - \rho_v$ and $x^*(X) = x'(X) - x(X)$. Subtracting (27) from the corresponding identity for (x', w') , the constant selector $q_C(X)$ cancels and we are left with

$$\overbrace{q_L(X)w^*(X) + q_R(X)f_\omega^*(X) + q_M(X)\varphi^*(X) + q_O(X)f_{\omega^2}^*(X)}^{\text{projection terms}} = x^*(X) + (Q'(X) - Q(X))\nu_{\mathbb{K}}(X).$$

By linearity of the quotient map,

$$Q'(X) = Q(X) + \left\lfloor \frac{q_L(X)w^*(X)}{\nu_{\mathbb{K}}(X)} \right\rfloor + \left\lfloor \frac{q_R(X)f_\omega^*(X)}{\nu_{\mathbb{K}}(X)} \right\rfloor + \left\lfloor \frac{q_M(X)\varphi^*(X)}{\nu_{\mathbb{K}}(X)} \right\rfloor + \left\lfloor \frac{q_O(X)f_{\omega^2}^*(X)}{\nu_{\mathbb{K}}(X)} \right\rfloor - \left\lfloor \frac{x^*(X)}{\nu_{\mathbb{K}}(X)} \right\rfloor.$$

Since $\deg x^* < m = \deg \nu_{\mathbb{K}}$, the last term vanishes. Focusing on the first quotient and applying linearity once more,

$$Q_L^*(X) = \left\lfloor \frac{q_L(X)w^*(X)}{\nu_{\mathbb{K}}(X)} \right\rfloor = \sum_{\ell=1}^n w_\ell^* \left\lfloor \frac{\lambda_\ell(X)q_L(X)}{\nu_{\mathbb{K}}(X)} \right\rfloor + \rho_w^* \frac{\nu_{\mathbb{H}}(X)}{\nu_{\mathbb{K}}(X)} q_L(X),$$

where we used that $w^*(h_\ell) = w_\ell^*$ and that, since $\mathbb{K} \subseteq \mathbb{H}$, the vanishing polynomial $\nu_{\mathbb{K}}(X)$ divides $\nu_{\mathbb{H}}(X)$, with $\frac{\nu_{\mathbb{H}}(X)}{\nu_{\mathbb{K}}(X)} = X^{2m} + X^m + 1$ of degree exactly $2m$. Writing

$$Q_\ell^J(X) = \left\lfloor \frac{\lambda_\ell(X)q_J(X)}{\nu_{\mathbb{K}}(X)} \right\rfloor, \quad \text{aux}_J(X) = \frac{\nu_{\mathbb{H}}(X)}{\nu_{\mathbb{K}}(X)} q_J(X),$$

for $J \in \{L, R, M, O\}$, and assuming the commitments $\{[Q_\ell^J(\tau)]_1\}_{\ell=1}^n$ and $[\text{aux}_L(\tau)]_1$ have been precomputed, we can compute the update $[Q_L^*(\tau)]_1$ in time proportional to the sparsity of w^* :

$$[Q_L^*(\tau)]_1 = \sum_{\ell \in \text{Supp}(w^*)} w_\ell^* [Q_\ell^L(\tau)]_1 + \rho_w^* [\text{aux}_L(\tau)]_1.$$

The terms $Q_R^*(X)$, $Q_M^*(X)$ and $Q_O^*(X)$ are updated analogously.

It remains to argue that this preprocessing is cheap. Note that the quotients $Q_\ell^J(X)$ have $\nu_{\mathbb{K}}(X)$ as denominator, but have the Lagrange of $\mathbb{H}$ in the numerator. This means that we cannot directly apply Lemma 3 and the amortized techniques of [ZOZ26] for computing their commitments. This is easily solved by observing that

$$Q_\ell^J(X) = \left\lfloor \frac{\lambda_\ell(X) \text{aux}_J(X)}{\nu_{\mathbb{H}}(X)} \right\rfloor.$$

Crucially, this preprocessing is entirely witness-independent: it only involves the selectors, which are part of the index. As illustrated in Figure 6, it can therefore be carried out once during the **Derive** phase, eliminating the need for a dedicated **BuildData** algorithm. Furthermore, proofs generated via **Update** can be directly updated again, so we have constructed a fully-dynamic SNARK for $\mathcal{R}_{\text{ck}}^{\text{gates}}$ without invoking Theorem 1.

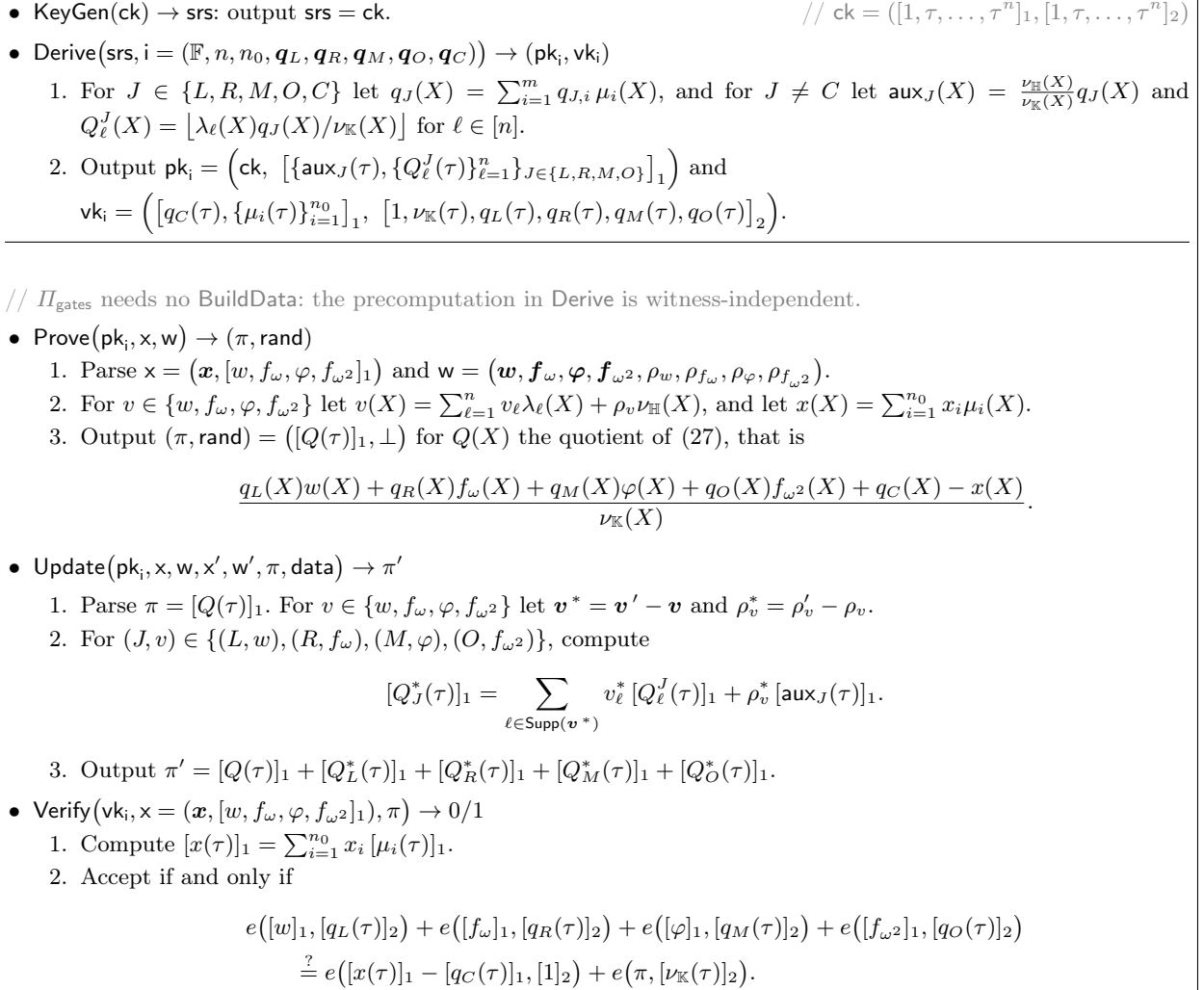


Fig. 6. Dynamic SNARK for $\mathcal{R}_{\text{ck}}^{\text{gates}}$.

Theorem 7. *The protocol from Figure 6 is a fully-dynamic SNARK for $\mathcal{R}_{\text{ck}}^{\text{gates}}$ in the AGM under the (n, n) -PDL assumption with the following complexities:*

- **Derive** performs $O(n \log n)$ operations in $\mathbb{G}_1$ and $O(n)$ in $\mathbb{G}_2$, and outputs pk_i of size $O(n)$ in $\mathbb{G}_1$ and vk_i of size $O(n_0)$ in $\mathbb{G}_1$ and $O(1)$ in $\mathbb{G}_2$.
- **Prove** performs $O(n \log n)$ operations in $\mathbb{F}$ and $O(n)$ in $\mathbb{G}_1$, and outputs a proof consisting of a single $\mathbb{G}_1$ element.
- **Update** performs $O(k)$ operations in $\mathbb{G}_1$, where $k = \sum_v |\text{Supp}(\mathbf{v}' - \mathbf{v})|$ is the Hamming distance between $\mathbf{w}$ and $\mathbf{w}'$. The updated proof consists of a single $\mathbb{G}_1$ element.
- **Verify** performs $O(n_0)$ operations in $\mathbb{G}_1$ and 6 pairings.

Moreover, the protocol requires no BuildData algorithm.

We omit the proof, as it easily follows from our discussion above. Again, the SNARK is not zero-knowledge, since the proof is a deterministic function of the instance, but this does not pose a problem for our final dynamic zkSNARK constructions (see Remark 2).

6 Dynasaurs: Universal Dynamic zkSNARKs for Plonk and V-R1CS-lite

In this section we present our dynamic zkSNARKs for the Plonk and V-R1CS-lite constraints systems in Section 6.1 and Section 6.2, respectively. The basic building blocks will be the dynamic zkSNARKs for the linear relations of Section 4.3 and Section 4.4, and the dynamic SNARKs of Section 5.

6.1 Dynamic zkSNARK for Plonk

We begin by recalling the Plonk constraint system [GWC19].

Definition 12 (Plonk constraint system). *Let $m = \frac{n}{3}$ and $n_0 \leq m$. The relation $\mathcal{R}^{\text{Plonk}}$ consists of tuples $(i, \mathbf{x}, \mathbf{w})$ with index $i = (\mathbb{F}, n, n_0, \mathbf{q}, \sigma)$ for $\mathbf{q} = (\mathbf{q}_L, \mathbf{q}_R, \mathbf{q}_M, \mathbf{q}_O, \mathbf{q}_C) \in (\mathbb{F}^m)^5$, public input $\mathbf{x} = \mathbf{x} \in \mathbb{F}^{n_0}$ and witness $\mathbf{w} = \mathbf{w} \in \mathbb{F}^n$, where $\sigma : [n] \rightarrow [n]$ is a permutation, satisfying:*

1. **Gate constraints:** for all $i \in [0, m - 1]$,

$$q_{L,i+1}w_{3i+1} + q_{R,i+1}w_{3i+2} + q_{M,i+1}w_{3i+1}w_{3i+2} + q_{O,i+1}w_{3i+3} + q_{C,i+1} = \chi_i,$$

where $\chi_i = x_{i+1}$ for $i < n_0$ and $\chi_i = 0$ for $i \geq n_0$.

2. **Copy constraints:** $w_\ell = w_{\sigma(\ell)}$ for all $\ell \in [n]$.

We have slightly modified the indexing convention from [GWC19]. In standard Plonk, a single parameter n denotes both the size of the selector vectors and the number of gate equations, with the permutation defined over $[3n]$. We adapt this notation for consistency with Section 4 and the subsequent section. Furthermore, rather than writing the wire assignments of the i -th gate as w_i , w_{n+i} and w_{2n+i} , we adopt the indexing from [CBBZ23], which places the three wires of a gate in consecutive positions of $\mathbf{w}$.

Problems with the gate constraints. Handling the gate constraints as in [GWC19] is unsuitable for our purposes. Directly translating the gate constraints of Definition 12 into a polynomial identity involves an expression of the form

$$q_L(X)w(X) + q_R(X)w(\omega X) + q_M(X)w(X)w(\omega X) + q_O(X)w(\omega^2 X) + q_C(X) - x(X),$$

which is not update-friendly for two reasons. First, there is a product of three polynomials, which cannot be checked directly with pairings. Second, our building blocks check relations directly on commitments rather than on openings at a random point, and $[w(\omega\tau)]_1$ cannot be derived from $[w(\tau)]_1$.

Our solution. To solve this, we commit to the shifts of $\mathbf{w}$ and to the product term as separate vectors, and split the identity into an equation that is *linear* in the witness—the gate equation of Section 5.2—and a Hadamard product. Crucially, the shifts and the copy constraints exactly correspond to the linear relation of Section 4.3. We specify this next.

As in Section 5.2, let $\mathbb{H} = \langle \omega \rangle$ be a multiplicative subgroup of order n , which we assume divisible by 3. Let $\mathbb{K} = \langle \gamma \rangle \subseteq \mathbb{H}$ be the subgroup of order $m = \frac{n}{3}$ generated by $\gamma = \omega^3$; we write $\{\mu_i(X)\}_{i=1}^m$ for its Lagrange basis, as opposed to $\{\lambda_\ell(X)\}_{\ell=1}^n$ for $\mathbb{H}$, and $\nu_{\mathbb{K}}(X) = X^m - 1$ for its vanishing polynomial.

Let $w(X)$ be the randomized low-degree extension of $\mathbf{w} \in \mathbb{F}^n$ over $\mathbb{H}$, and let $f_\omega(X)$, $f_{\omega^2}(X)$, $\varphi(X)$ be those of the shifted vectors $\mathbf{f}_\omega = (w_2, \dots, w_n, w_1)$, $\mathbf{f}_{\omega^2} = (w_3, \dots, w_1, w_2)$ and of the Hadamard product $\varphi = \mathbf{w} \circ \mathbf{f}_\omega$. The selectors, on the other hand, are encoded over $\mathbb{K}$ as $q_J(X) = \sum_{i=1}^m q_{J,i} \mu_i(X)$ for $J \in \{L, R, M, O, C\}$, and so is the public input, $x(X) = \sum_{i=1}^{n_0} x_i \mu_i(X)$, so that $x(\gamma^i) = x_{i+1}$ for every $i \in [0, m-1]$.

The point of this encoding is that each gate becomes a statement about a *single* evaluation point. Indeed, $f_{\omega, \ell} = w_{\ell+1}$ and $f_{\omega^2, \ell} = w_{\ell+2}$, so the three wires $w_{3i+1}, w_{3i+2}, w_{3i+3}$ of gate i are the entries of $\mathbf{w}, \mathbf{f}_\omega, \mathbf{f}_{\omega^2}$ at the common index $\ell = 3i+1$, and $\varphi_{3i+1} = w_{3i+1} w_{3i+2}$ is the multiplicative term. Since $\mathbf{h}_{3i+1} = \omega^{3i} = \gamma^i$, the i -th gate equation reads

$$q_L(\gamma^i)w(\gamma^i) + q_R(\gamma^i)f_\omega(\gamma^i) + q_M(\gamma^i)\varphi(\gamma^i) + q_O(\gamma^i)f_{\omega^2}(\gamma^i) + q_C(\gamma^i) = x(\gamma^i).$$

As i ranges over $[0, m-1]$, the point γ^i ranges over all of $\mathbb{K}$. The m gate constraints therefore hold if and only if the difference of the two sides vanishes on $\mathbb{K}$, i.e. if and only if it is divisible by $\nu_{\mathbb{K}}(X)$. Summing up, provided $\mathbf{f}_\omega$ and $\mathbf{f}_{\omega^2}$ are the claimed shifts of $\mathbf{w}$, the gate constraints are equivalent to the existence of $Q(X), h(X) \in \mathbb{F}[X]$ such that

$$q_L(X)w(X) + q_R(X)f_\omega(X) + q_M(X)\varphi(X) + q_O(X)f_{\omega^2}(X) + q_C(X) = x(X) + Q(X)\nu_{\mathbb{K}}(X), \quad (29)$$

$$w(X)f_\omega(X) - \varphi(X) = h(X)\nu_{\mathbb{H}}(X). \quad (30)$$

Equation (30) is a standard Hadamard product over $\mathbb{H}$, for which we use the dynamic SNARK of Section 5.1. Equation (29) is exactly the gate equation of Section 5.2, and we use its corresponding dynamic SNARK.

The Plonk dynamic zkSNARK. To sum up, building a dynamic zkSNARK for the Plonk constraint system boils down to proving and updating: (i) the gate equation (29), (ii) the Hadamard equation (30), (iii) the wiring constraints $w_\ell = w_{\sigma(\ell)}$, and (iv) the consistency of $\mathbf{f}_\omega$ and $\mathbf{f}_{\omega^2}$ with $\mathbf{w}$. Items (i), (ii) and (iv) together are equivalent to the gate constraints, while (iii) and (iv) are exactly the conditions of the linear relation of Definition 8. All the prover needs to do, then, is commit to the randomized low-degree extensions of $\mathbf{w}, \mathbf{f}_\omega, \mathbf{f}_{\omega^2}, \varphi$ and invoke the SNARKs for the gate, Hadamard and linear relations; the verifier checks the three proofs, recomputing $[x(\tau)]_1$ from the public input itself. Proofs are updated by calling the **Update** algorithm of each building block, and this is efficient because sparsity is preserved: if $\mathbf{w}$ and $\mathbf{w}'$ are at Hamming distance k , then $\mathbf{f}_\omega$ and $\mathbf{f}_{\omega^2}$ change in k positions each and φ in at most $2k$, so every building block is updated on a difference of Hamming weight $O(k)$.

Remark 3 (Asymmetric Group Commitments). There is one subtle detail that we have ignored: the linear relation requires the commitment to the vector $\mathbf{f}_\omega$ to be in $\mathbb{G}_1$ (see Definition 8), whereas the Hadamard relation needs that the commitment to this vector is in $\mathbb{G}_2$ (see Definition 10). Therefore, the prover needs to send the commitments $[f_\omega]_1$ and $[f_\omega]_2$ to $\mathbf{f}_\omega$ in both groups. The verifier can check that they are consistent with the pairing $e([f_\omega]_1, [1]_2) = e([1]_1, [f_\omega]_2)$. It is exactly at these points that the use of symmetric pairings hides these costs.

Theorem 8 (Dynamic zkSNARK for Plonk). *Let Π_{had} be the (semi) dynamic SNARK for the Hadamard relation from Section 5.1, Π_{gates} the (semi) dynamic SNARK for the gates relation from Section 5.2 and Π_{lin} the (semi) dynamic zkSNARK for the linear relation from Section 4.3. The protocol from Figure 7 is a (semi) dynamic zkSNARK for $\mathcal{R}^{\text{Plonk}}$ in the AGM and ROM under the (n, n) -PDL assumption, with the complexities listed for $\mathcal{R}^{\text{Plonk}}$ in Table 2.*

Let us discuss the claimed costs. The proof size and the complexities of **Derive**, **BuildData**, **Prove** and **Verify** follow naturally from each of the building blocks, so we will focus on the **Update** algorithm. In the semi-dynamic setting, the cost is dominated by **Had.Update** (of $O(k \log^2 k)$) and **Lin.Update** (of $O(k + \log n)$),

which results in a total of $O(\log n + k \log^2 k)^{14}$. In the fully-dynamic setting, observe that the update for the Hadamard relation yields a cost of $O(k\sqrt{n \log n} + k \log^2 k)$ (see Theorem 6), the one for the gates relation a cost of $O(k)$ (see Theorem 7), and the one for the linear relation a cost of $O(k\sqrt{n \log n})$ (see Corollary 3). Adding the involved costs, we obtain updates with complexity $O(k\sqrt{n \log n} + k \log^2 k)$.

We omit the proof of completeness, which follows directly from the completeness of Π_{had} , Π_{gates} and Π_{lin} . Throughout we write $i_{\text{had}} = (\mathbb{F}, n)$, $i_{\text{lin}} = (\mathbb{F}, n, \sigma)$ and $i_{\text{gates}} = (\mathbb{F}, n, n_0, \mathbf{q}_L, \mathbf{q}_R, \mathbf{q}_M, \mathbf{q}_O, \mathbf{q}_C)$ for the three sub-indices derived from i .

Table 2. Asymptotic costs and proof size of our (semi) dynamic zkSNARKs.

	Derive	BuildData	Prove	Update (semi-dyn.)	Update (fully-dyn.)	Verify	$ \pi $
$\mathcal{R}^{\text{Plonk}}$	$n \log n$	$n \log n$	$n \log n$	$\log n + k \log^2 k$	$k\sqrt{n \log n} + k \log^2 k$	$ \mathbf{x} + \log n$ 23 $\mathbb{P}$ (13 batched)	19 $\mathbb{G}_1, \mathbb{G}_2$ 2 $\mathbb{F}$
$\mathcal{R}^{V\text{-R1CS-lite}}$	$nV \log n$	$n \log n$	$n \log n$	$\log n + kV \log^2(kV)$	$kV\sqrt{n \log n} + kV \log^2(kV)$	$ \mathbf{x} + \log n + V$ 19 $\mathbb{P}$ (10 batched)	18 $\mathbb{G}_1, \mathbb{G}_2$ 2 $\mathbb{F}$

Note: O notation; k is the Hamming distance between $(\mathbf{x}, \mathbf{w})$ and $(\mathbf{x}', \mathbf{w}')$, V the fan-out, $\mathbb{P}$ a pairing. Batched counts from Section D.

Proposition 8 (Knowledge soundness). *The protocol from Figure 7 is knowledge-sound in the AGM and ROM under the (n, n) -PDL assumption.*

Proof. Let $\mathcal{A}$ be an algebraic PPT adversary that on input (i, srs) outputs $(\mathbf{x} = \mathbf{x}, \pi)$ with $\text{Verify}(\text{vk}_i, \mathbf{x}, \pi) = 1$. Parse

$$\pi = ([w(\tau), f_{\omega}^{(1)}(\tau), \varphi(\tau), f_{\omega}^{(2)}(\tau)]_1, [f_{\omega}^{(2)}(\tau)]_2, \pi_{\text{had}}, \pi_{\text{gates}}, \pi_{\text{lin}})$$

and denote $\mathbf{x}_{\text{had}} = ([w(\tau), \varphi(\tau)]_1, [f_{\omega}^{(2)}(\tau)]_2)$, $\mathbf{x}_{\text{gates}} = (\mathbf{x}, [w(\tau), f_{\omega}^{(1)}(\tau), \varphi(\tau), f_{\omega}^{(2)}(\tau)]_1)$ and $\mathbf{x}_{\text{lin}} = [w(\tau), f_{\omega}^{(1)}(\tau), f_{\omega}^{(2)}(\tau)]_1$, the three instances formed by the verifier. As $\mathcal{A}$ is algebraic, it must also output polynomials $f_{\omega}^{(1)}(X)$ and $f_{\omega}^{(2)}(X)$ of degree at most n that explain the group elements $[f_{\omega}^{(1)}(\tau)]_1$ and $[f_{\omega}^{(2)}(\tau)]_2$.

Now, as the verifier outputs acceptance, in particular it must hold that $e([f_{\omega}^{(1)}(\tau)]_1, [1]_2) = e([1]_1, [f_{\omega}^{(2)}(\tau)]_2)$, which directly implies $f_{\omega}^{(1)}(X) = f_{\omega}^{(2)}(X) =: f_{\omega}(X)$. Now, define the following PPT adversaries:

- $\mathcal{B}_{\text{had}}$ against Had, it runs $\mathcal{A}$ and outputs i_{had} and $(\mathbf{x}_{\text{had}}, \pi_{\text{had}})$.
- $\mathcal{B}_{\text{gates}}$ against Gates, it runs $\mathcal{A}$ and outputs i_{gates} and $(\mathbf{x}_{\text{gates}}, \pi_{\text{gates}})$.
- $\mathcal{B}_{\text{lin}}$ against Lin, it runs $\mathcal{A}$ and outputs i_{lin} and $(\mathbf{x}_{\text{lin}}, \pi_{\text{lin}})$.

Since Verify accepts only if the checks Had.Verify, Gates.Verify and Lin.Verify pass, whenever $\mathcal{A}$ wins so do $\mathcal{B}_{\text{had}}$, $\mathcal{B}_{\text{gates}}$ and $\mathcal{B}_{\text{lin}}$. Let Ext_{had} , $\text{Ext}_{\text{gates}}$ and Ext_{lin} be the extractors generated by the knowledge-soundness of Had for $\mathcal{B}_{\text{had}}$, Gates for $\mathcal{B}_{\text{gates}}$ and Lin for $\mathcal{B}_{\text{lin}}$, respectively. Let Ext run the 3 of them on the transcript of $\mathcal{A}$, and output $\mathbf{w}$, where $\mathbf{w}$ is such that $\mathbf{w}_{\text{had}} = (\mathbf{w}, \mathbf{f}_{\omega}, \varphi, \rho_{\mathbf{w}}, \rho_{\mathbf{f}_{\omega}}, \rho_{\varphi}) \leftarrow \text{Ext}_{\text{had}}$, $\mathbf{w}_{\text{gates}} = (\mathbf{w}, \mathbf{f}_{\omega}, \varphi, \mathbf{f}_{\omega}^{(2)}, \rho_{\mathbf{w}}, \rho_{\mathbf{f}_{\omega}}, \rho_{\varphi}, \rho_{\mathbf{f}_{\omega}^{(2)}}) \leftarrow \text{Ext}_{\text{gates}}$ and $\mathbf{w}_{\text{lin}} = (\mathbf{w}, \mathbf{f}_{\omega}, \mathbf{f}_{\omega}^{(2)}, \rho_{\mathbf{w}}, \rho_{\mathbf{f}_{\omega}}, \rho_{\mathbf{f}_{\omega}^{(2)}}) \leftarrow \text{Ext}_{\text{lin}}$. Given the binding property of the commitment scheme¹⁵, all of the extracted vectors and blinders are automatically consistent with each other. Except with negligible probability, we have $(i_{\text{had}}, \mathbf{x}_{\text{had}}, \mathbf{w}_{\text{had}}) \in \mathcal{R}_{\text{ck}}^{\text{had}}$, $(i_{\text{gates}}, \mathbf{x}_{\text{gates}}, \mathbf{w}_{\text{gates}}) \in \mathcal{R}_{\text{ck}}^{\text{gates}}$ and $(i_{\text{lin}}, \mathbf{x}_{\text{lin}}, \mathbf{w}_{\text{lin}}) \in \mathcal{R}_{\text{ck}}^{\text{Plonk, lin}}$. The last membership gives the copy constraints $w_{\ell} = w_{\sigma(\ell)}$ together with $\mathbf{f}_{\omega} = (w_2, \dots, w_n, w_1)$ and $\mathbf{f}_{\omega}^{(2)} = (w_3, \dots, w_1, w_2)$, and the first gives $\varphi = \mathbf{w} \circ \mathbf{f}_{\omega}$; substituting $f_{\omega, 3i+1} = w_{3i+2}$, $f_{\omega^2, 3i+1} = w_{3i+3}$ and $\varphi_{3i+1} = w_{3i+1}w_{3i+2}$ into (26) turns the second membership into the gate constraints of Definition 12. Note that $\mathbf{x}$ is not extracted: it is part of $\mathbf{x}_{\text{gates}}$, hence the public input on which $\mathcal{A}$ was challenged. This guarantees that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}^{\text{Plonk}}$, so Ext extracts $\mathbf{w}$ and we are done. A union bound over the extraction failures concludes the proof. $\square$

¹⁴ In Step 3, the updater can compute $[v'(\tau)]_1 = [v(\tau)]_1 + \sum_{\ell \in \text{Supp}(\mathbf{v}^*)} v_{\ell}^* [\lambda(\tau)]_1 + (\rho'_v - \rho_v) [\nu_{\mathbb{H}}(\tau)]_1$, where $\mathbf{v}^* = \mathbf{v}' - \mathbf{v}$.

This can be done in time $O(k)$, as we assume that the updater already holds the updated coordinates.

¹⁵ We could alternatively argue using the fact that the adversaries are also algebraic.

• $\text{Derive}(\text{srs}, i = (\mathbb{F}, n, n_0, \mathbf{q}_L, \mathbf{q}_R, \mathbf{q}_M, \mathbf{q}_O, \mathbf{q}_C, \sigma)) \rightarrow (\text{pk}_i, \text{vk}_i)$ 1. Let $i_{\text{had}} = (\mathbb{F}, n)$, $i_{\text{gates}} = (\mathbb{F}, n, n_0, \mathbf{q}_L, \mathbf{q}_R, \mathbf{q}_M, \mathbf{q}_O, \mathbf{q}_C)$ and $i_{\text{lin}} = (\mathbb{F}, n, \sigma)$, and compute $(\text{pk}_i^P, \text{vk}_i^P) \leftarrow P.\text{Derive}(\text{srs}, i_P)$ for $P \in \{\text{had}, \text{gates}, \text{lin}\}$. 2. Output $\text{pk}_i = (\{\text{pk}_i^P\}_P, [\{\lambda_\ell(\tau)\}_{\ell=1}^n, \nu_{\mathbb{H}}(\tau)]_2)$ and $\text{vk}_i = \{\text{vk}_i^P\}_P$.
• $\text{Prove}(\text{pk}_i, x = x, w = w) \rightarrow (\pi, \text{rand})$ 1. Let $\mathbf{f}_w = (w_2, \dots, w_n, w_1)$, $\mathbf{f}_{w^2} = (w_3, \dots, w_n, w_1, w_2)$ and $\varphi = w \circ \mathbf{f}_w$. 2. For $v \in \{w, f_w, \varphi, f_{w^2}\}$, sample $\rho_v \leftarrow \mathbb{F}$ and compute $[v(\tau)]_1$ for $v(X) = \sum_{\ell=1}^n v_\ell \lambda_\ell(X) + \rho_v \nu_{\mathbb{H}}(X)$. Compute also $[f_w(\tau)]_2$. 3. Let $x_{\text{had}} = ([w(\tau), \varphi(\tau)]_1, [f_w(\tau)]_2)$ and $w_{\text{had}} = (w, \mathbf{f}_w, \varphi, \rho_w, \rho_{f_w}, \rho_\varphi)$, and compute $(\pi_{\text{had}}, \perp) \leftarrow \text{Had.Prove}(\text{pk}_i, x_{\text{had}}, w_{\text{had}})$. 4. Let $x_{\text{gates}} = (x, [w(\tau), f_w(\tau), \varphi(\tau), f_{w^2}(\tau)]_1)$ and $w_{\text{gates}} = (w, \mathbf{f}_w, \varphi, \mathbf{f}_{w^2}, \rho_w, \rho_{f_w}, \rho_\varphi, \rho_{f_{w^2}})$, and compute $(\pi_{\text{gates}}, \perp) \leftarrow \text{Gates.Prove}(\text{pk}_i, x_{\text{gates}}, w_{\text{gates}})$. 5. Let $x_{\text{lin}} = [w(\tau), f_w(\tau), f_{w^2}(\tau)]_1$ and $w_{\text{lin}} = (w, \mathbf{f}_w, \mathbf{f}_{w^2}, \rho_w, \rho_{f_w}, \rho_{f_{w^2}})$, and compute $(\pi_{\text{lin}}, \text{rand}_{\text{lin}}) \leftarrow \text{Lin.Prove}(\text{pk}_i, x_{\text{lin}}, w_{\text{lin}})$. 6. Output $\pi = ([w(\tau), f_w(\tau), \varphi(\tau), f_{w^2}(\tau)]_1, [f_w(\tau)]_2, \pi_{\text{had}}, \pi_{\text{gates}}, \pi_{\text{lin}})$ and $\text{rand} = (\rho_w, \rho_{f_w}, \rho_\varphi, \rho_{f_{w^2}}, \text{rand}_{\text{lin}})$. • $\text{BuildData}(\text{pk}_i, x, w, \text{rand}) \rightarrow \text{data}$ 1. Parse $\text{rand} = (\rho_w, \rho_{f_w}, \rho_\varphi, \rho_{f_{w^2}}, \text{rand}_{\text{lin}})$ and recompute $x_{\text{had}}, w_{\text{had}}, x_{\text{lin}}, w_{\text{lin}}$ as in Prove. 2. Compute $\text{data}_{\text{had}} \leftarrow \text{Had.BuildData}(\text{pk}_i, x_{\text{had}}, w_{\text{had}}, \perp)$ and $\text{data}_{\text{lin}} \leftarrow \text{Lin.BuildData}(\text{pk}_i, x_{\text{lin}}, w_{\text{lin}}, \text{rand}_{\text{lin}})$. 3. Output $\text{data} = (\text{rand}, \text{data}_{\text{had}}, \text{data}_{\text{lin}})$. • $\text{Update}(\text{pk}_i, x, w, x' = x', w' = w', \pi, \text{data}) \rightarrow \pi'$ 1. Parse $\pi = ([w(\tau), f_w(\tau), \varphi(\tau), f_{w^2}(\tau)]_1, [f_w(\tau)]_2, \pi_{\text{had}}, \pi_{\text{gates}}, \pi_{\text{lin}})$, $\text{data} = (\text{rand}, \text{data}_{\text{had}}, \text{data}_{\text{lin}})$ and $\text{rand} = (\rho_w, \rho_{f_w}, \rho_\varphi, \rho_{f_{w^2}}, \text{rand}_{\text{lin}})$. Recompute $x_{\text{had}}, w_{\text{had}}, x_{\text{gates}}, w_{\text{gates}}, x_{\text{lin}}, w_{\text{lin}}$ as in Prove, reading the commitments off π and the openings off rand. 2. Let $\mathbf{f}'_w = (w'_2, \dots, w'_n, w'_1)$, $\mathbf{f}'_{w^2} = (w'_3, \dots, w'_n, w'_1, w'_2)$ and $\varphi' = w' \circ \mathbf{f}'_w$. 3. For $v \in \{w, f_w, \varphi, f_{w^2}\}$, sample $\rho'_v \leftarrow \mathbb{F}$ and compute $[v'(\tau)]_1 = \sum_{\ell=1}^n v'_\ell [\lambda_\ell(\tau)]_1 + \rho'_v [\nu_{\mathbb{H}}(\tau)]_1$. Compute also $[f'_w(\tau)]_2$. 4. Let $x'_{\text{had}} = ([w'(\tau), \varphi'(\tau)]_1, [f'_w(\tau)]_2)$, $w'_{\text{had}} = (w', \mathbf{f}'_w, \varphi', \rho'_w, \rho'_{f_w}, \rho'_\varphi)$ and compute $\pi'_{\text{had}} \leftarrow \text{Had.Update}(\text{pk}_i, x_{\text{had}}, w_{\text{had}}, x'_{\text{had}}, w'_{\text{had}}, \pi_{\text{had}}, \text{data}_{\text{had}})$. 5. Let $x'_{\text{gates}} = (x', [w'(\tau), f'_w(\tau), \varphi'(\tau), f'_{w^2}(\tau)]_1)$, $w'_{\text{gates}} = (w', \mathbf{f}'_w, \varphi', \mathbf{f}'_{w^2}, \rho'_w, \rho'_{f_w}, \rho'_\varphi, \rho'_{f_{w^2}})$ and compute $\pi'_{\text{gates}} \leftarrow \text{Gates.Update}(\text{pk}_i, x_{\text{gates}}, w_{\text{gates}}, x'_{\text{gates}}, w'_{\text{gates}}, \pi_{\text{gates}}, \perp)$. 6. Let $x'_{\text{lin}} = [w'(\tau), f'_w(\tau), f'_{w^2}(\tau)]_1$, $w'_{\text{lin}} = (w', \mathbf{f}'_w, \mathbf{f}'_{w^2}, \rho'_w, \rho'_{f_w}, \rho'_{f_{w^2}})$ and compute $\pi'_{\text{lin}} \leftarrow \text{Lin.Update}(\text{pk}_i, x_{\text{lin}}, w_{\text{lin}}, x'_{\text{lin}}, w'_{\text{lin}}, \pi_{\text{lin}}, \text{data}_{\text{lin}})$. 7. Output $\pi' = ([w'(\tau), f'_w(\tau), \varphi'(\tau), f'_{w^2}(\tau)]_1, [f'_w(\tau)]_2, \pi'_{\text{had}}, \pi'_{\text{gates}}, \pi'_{\text{lin}})$. • $\text{Verify}(\text{vk}_i, x = x, \pi) \rightarrow 0/1$ 1. Parse $\pi = ([w(\tau), f_w(\tau), \varphi(\tau), f_{w^2}(\tau)]_1, [f_w(\tau)]_2, \pi_{\text{had}}, \pi_{\text{gates}}, \pi_{\text{lin}})$ and let $x_{\text{had}}, x_{\text{gates}}, x_{\text{lin}}$ be as in Prove. 2. Accept if and only if the following four checks pass: $e([f_w(\tau)]_1, [1]_2) \stackrel{?}{=} e([1]_1, [f_w(\tau)]_2), \quad \text{Had.Verify}(\text{vk}_i, x_{\text{had}}, \pi_{\text{had}}) \stackrel{?}{=} 1,$ $\text{Gates.Verify}(\text{vk}_i, x_{\text{gates}}, \pi_{\text{gates}}) \stackrel{?}{=} 1, \quad \text{Lin.Verify}(\text{vk}_i, x_{\text{lin}}, \pi_{\text{lin}}) \stackrel{?}{=} 1.$

Fig. 7. (Semi) dynamic zkSNARK for $\mathcal{R}^{\text{Plonk}}$. $\text{KeyGen}(1^\lambda)$ outputs $\text{srs} = \text{ck}$.

Proposition 9 (Zero-knowledge). *The protocol from Figure 7 is zero-knowledge.*

Proof. The simulator is depicted in Figure 8. We condition on $\nu_{\mathbb{H}}(\tau) \neq 0$, which fails with probability at most $n/|\mathbb{F}|$ and which also gives $\nu_{\mathbb{K}}(\tau) \neq 0$, since $\nu_{\mathbb{K}}(X)$ divides $\nu_{\mathbb{H}}(X)$. In the real protocol, a proof is of the form

$$\pi = ([w(\tau), f_{\omega}(\tau), \varphi(\tau), f_{\omega^2}(\tau)]_1, [f_{\omega}(\tau)]_2, \pi_{\text{had}}, \pi_{\text{gates}}, \pi_{\text{lin}}).$$

The commitments $[v(\tau)]_1$ have been blinded with independent $\rho_v \leftarrow \mathbb{F}$ and, as $\nu_{\mathbb{H}}(\tau) \neq 0$, are jointly uniformly random in the real protocol, so they can be simulated by sampling random $\rho_v \leftarrow \mathbb{F}$ and defining $\rho_v[\nu_{\mathbb{H}}(\tau)]_1$ for $v \in \{w, f_{\omega}, \varphi, f_{\omega^2}\}$. Same for $[f_{\omega}(\tau)]_2$, which is determined by the field element $f_{\omega}(\tau)$. This is exactly how the simulator from Figure 8 defines them. Recall that $\pi_{\text{had}} = [q(\tau)]_1$ and $\pi_{\text{gates}} = [Q(\tau)]_1$, where $q(X) = \frac{w(X)f_{\omega}(X) - \varphi(X)}{\nu_{\mathbb{H}}(X)}$ and

$$Q(X) = \frac{q_L(X)w(X) + q_R(X)f_{\omega}(X) + q_M(X)\varphi(X) + q_O(X)f_{\omega^2}(X) + q_C(X) - x(X)}{\nu_{\mathbb{K}}(X)}$$

Conditioned on the four commitments, $q(\tau)$ and $Q(\tau)$ are the *unique* field elements satisfying the verification equations of Π_{had} and Π_{gates} , precisely because $\nu_{\mathbb{H}}(\tau) \neq 0$ and $\nu_{\mathbb{K}}(\tau) \neq 0$. Both proofs are therefore deterministic functions of already-blinded commitments and of the public x , which is why the fact that Π_{had} and Π_{gates} are not zero-knowledge on their own is harmless here (cf. Remark 2). The simulator, using the trapdoor τ , can compute those commitments exactly as in Figure 8 to make the verifier accept, and they are thus distributed as in a real execution. Finally, it is possible to simulate the proof π_{lin} thanks to the zero-knowledge property of Π_{lin} , replacing Lin.Prove by $\text{Lin.Sim}(\tau, i_{\text{lin}}, x_{\text{lin}})$ in a single hybrid step.

Indistinguishability of fresh proofs and updated proofs follows from the same property of Π_{lin} and the discussion above: step 3 of **Update** samples fresh blinders $\rho'_v \leftarrow \mathbb{F}$, so the updated commitments are again jointly uniform, π'_{had} and π'_{gates} are again the unique accepting values determined by them, and π'_{lin} is handled by the second zero-knowledge condition of Π_{lin} in Definition 3. The same argument gives the fully-dynamic statement: **Update** re-randomizes every commitment, so π'_{had} and π'_{gates} are again the unique accepting values, and it suffices to invoke the corresponding condition of the fully-dynamic notion of Section A for Π_{lin} , which it satisfies by Theorem 1. $\square$

Fig. 8. Simulator for the protocol of Figure 7

$\text{Sim}(\tau, i = (\mathbb{F}, n, n_0, q_L, q_R, q_M, q_O, q_C, \sigma), x = x) \rightarrow \pi$

1. For $v \in \{w, f_{\omega}, \varphi, f_{\omega^2}\}$, sample $\rho_v \leftarrow \mathbb{F}$ and compute $v(\tau) = \rho_v \nu_{\mathbb{H}}(\tau)$, then $[w(\tau), f_{\omega}(\tau), \varphi(\tau), f_{\omega^2}(\tau)]_1$ and $[f_{\omega}(\tau)]_2$
2. Compute $q(\tau) = \frac{w(\tau)f_{\omega}(\tau) - \varphi(\tau)}{\nu_{\mathbb{H}}(\tau)}$. Set $\pi_{\text{had}} = [q(\tau)]_1$
3. Compute $x(\tau)$ for $x(X) = \sum_{i=1}^{n_0} x_i \mu_i(X)$ and
$$Q(\tau) = \frac{q_L(\tau)w(\tau) + q_R(\tau)f_{\omega}(\tau) + q_M(\tau)\varphi(\tau) + q_O(\tau)f_{\omega^2}(\tau) + q_C(\tau) - x(\tau)}{\nu_{\mathbb{K}}(\tau)}.$$

Set $\pi_{\text{gates}} = [Q(\tau)]_1$.

4. Let $x_{\text{lin}} = [w(\tau), f_{\omega}(\tau), f_{\omega^2}(\tau)]_1$. Compute $\pi_{\text{lin}} \leftarrow \text{Lin.Sim}(\tau, i_{\text{lin}}, x_{\text{lin}})$
5. Output $\pi = ([w(\tau), f_{\omega}(\tau), \varphi(\tau), f_{\omega^2}(\tau)]_1, [f_{\omega}(\tau)]_2, \pi_{\text{had}}, \pi_{\text{gates}}, \pi_{\text{lin}})$

Remark 4 (Batching Pairings). Our compiler and commit-and-prove zkSNARKs [CFQ19] lead to a modular design of our final dynamic zkSNARKs. This has the advantage that each building block can be analyzed and understood independently. However, we can optimize further by no longer treating them as black boxes. Concretely, we show in Section D that it is possible to reduce the amount of pairing operations the verifier performs by using standard techniques [GWC19, CHM⁺20].

6.2 Dynamic zkSNARK for V-R1CS-lite

We begin with the definition of the constraint system that we are interested in.

Definition 13 (V-R1CS-lite). *The V-R1CS-lite universal relation $\mathcal{R}^{V\text{-R1CS-lite}}$ is defined as*

$$\mathcal{R}^{V\text{-R1CS-lite}} = \left\{ (i, \mathbf{x}, \mathbf{w}) \left| \begin{array}{l} i = (\mathbb{F}, n, n_0, V, A, B) \\ \mathbf{x} = \mathbf{x} \in \mathbb{F}^{n_0}, \mathbf{w} = \mathbf{w} \in \mathbb{F}^{n-n_0} \\ \mathbf{z} = (\mathbf{x}, \mathbf{w}) \in \mathbb{F}^n \\ \begin{pmatrix} A \\ B \end{pmatrix} \in \mathcal{M}_{2n,n}(V) \\ A\mathbf{z} \circ B\mathbf{z} = \mathbf{z} \end{array} \right. \right\}.$$

We have performed one crucial modification to the standard definition of R1CS-lite [CFF⁺21]. Normally, it is assumed that the matrices A and B have at most $O(n)$ non-zero elements, without any further assumption on how they are distributed. In our case, we have assumed that those non-zero terms are distributed in a particular way, namely, that $\begin{pmatrix} A \\ B \end{pmatrix} \in \mathbb{F}^{2n \times n}$ has at most V non-zero entries in each column. As explained in [RZ21], this constraint system naturally captures arithmetic circuits of fan-out V . We remark that circuits of small fan-out are a natural setting of dynamic zkSNARKs. Intuitively, if an updated gate has a large fan-out, the modification will spread to many gates of the circuit, and the updated witness will be far in Hamming distance from the original one, defeating the purpose of *any* dynamic zkSNARK.

Overview of the dynamic zkSNARK. It is clear that building a dynamic zkSNARK for $\mathcal{R}^{V\text{-R1CS-lite}}$ reduces to: (i) a Hadamard product $\mathbf{z}_A \circ \mathbf{z}_B = \mathbf{z}$ and (ii) the linear checks $A\mathbf{z} = \mathbf{z}_A$ and $B\mathbf{z} = \mathbf{z}_B$.

Therefore, the prover starts by committing to $\mathbf{z} = (\mathbf{x}, \mathbf{w})$ and to $\mathbf{z}_A = A\mathbf{z}$ and $\mathbf{z}_B = B\mathbf{z}$. It then computes π_{had} using `Had.Prove`, which shows $\mathbf{z}_A \circ \mathbf{z}_B = \mathbf{z}$, and π_{lin} with `Lin.Prove`, which attests to $\mathbf{z}_A = A\mathbf{z}$ and $\mathbf{z}_B = B\mathbf{z}$. Both proofs can be updated with their respective `Update` algorithm, and both updates are cheap because sparsity is preserved up to a factor V : if $\mathbf{z}$ and $\mathbf{z}'$ are at Hamming distance k , then $\mathbf{z}'_M$ and $\mathbf{z}_M$ are at Hamming distance kV for $M \in \{A, B\}$. As in the Plonk case, $\mathbf{z}_B$ must be committed in both groups and the verifier checks $e([z_B(\tau)]_1, [1]_2) = e([1]_1, [z_B(\tau)]_2)$.

Binding the public instance. Neither $\mathbf{z}_A \circ \mathbf{z}_B = \mathbf{z}$ nor the linchecks force the prover to place $\mathbf{x}$ in the public positions of $\mathbf{z}$. Binding the instance is done with standard techniques (a commitment to a shifted witness, in the style of [CHM⁺20, CFF⁺21]) and leaves all update costs unchanged; it adds two pairings to verification, which batching techniques bring down to one. We detail this next.

The prover commits, besides $\mathbf{z}$, to the *shifted witness*

$$\hat{w}(X) = \frac{z(X) - x(X)}{\nu_0(X)}, \quad \nu_0(X) = \prod_{\ell=1}^{n_0} (X - \mathbf{h}_\ell),$$

where $x(X) = \sum_{\ell=1}^{n_0} x_\ell \lambda_\ell(X)$ is the low-degree extension of the instance. This is a polynomial precisely when $z(\mathbf{h}_\ell) = x_\ell$ for every $\ell \leq n_0$. The verifier recomputes $[x(\tau)]_1$ from $\mathbf{x}$ and checks

$$e([z(\tau)]_1 - [x(\tau)]_1, [1]_2) = e([\hat{w}(\tau)]_1, [\nu_0(\tau)]_2), \quad (31)$$

with $[\nu_0(\tau)]_2$ part of $\mathbf{vk}_i$. In the AGM, (31) forces the identity $z(X) - x(X) = \nu_0(X)\hat{w}(X)$, and hence $z_\ell = x_\ell$ for every $\ell \leq n_0$. Zero-knowledge is unaffected: $\hat{w}(\tau)$ is a deterministic function of $z(\tau)$ and the public $\mathbf{x}$, so $[\hat{w}(\tau)]_1$ reveals nothing beyond $[z(\tau)]_1$, which ρ_z already hides.

We still need to clarify how to compute the commitment $[\hat{w}(\tau)]_1$ in the dynamic setting. First, write $\lambda_\ell(X) = \frac{\mathbf{h}_\ell}{n} \frac{\nu_{\mathbb{H}}(X)}{X - \mathbf{h}_\ell}$ and $\hat{\nu}(X) = \nu_{\mathbb{H}}(X)/\nu_0(X) = \prod_{\ell > n_0} (X - \mathbf{h}_\ell)$, and define the polynomials $\hat{\lambda}_\ell(X) = \lambda_\ell(X)/\nu_0(X) = \frac{\mathbf{h}_\ell}{n} \frac{\hat{\nu}(X)}{X - \mathbf{h}_\ell}$. The `Derive` can compute $\{[\hat{\lambda}_\ell(\tau)]_1\}_{\ell=n_0+1}^n$ in $O(n \log n)$ group operations, which changes neither the asymptotic cost of `Derive` nor the order of $|\mathbf{pk}_i|$. Second, in this basis the shifted witness

$$\hat{w}(X) = \sum_{\ell=n_0+1}^n z_\ell \hat{\lambda}_\ell(X) + \rho_z \hat{\nu}(X)$$

is as sparse as $\mathbf{w}$ itself, so $[\hat{w}(\tau)]_1$ is updated in $O(k)$ scalar multiplications exactly as the other commitments are. The complete protocol is given in Figure 9.

Theorem 9 (Dynamic zkSNARK for V-R1CS-lite). *Assume $V = O(\log n)$. Let Π_{had} be the (semi) dynamic SNARK for the Hadamard relation from Section 5.1, and Π_{lin} the (semi) dynamic zkSNARK for the linear relation from Section 4.4. The protocol from Figure 9 is a (semi) dynamic zkSNARK for $\mathcal{R}^{V\text{-R1CS-lite}}$ in the AGM and ROM under the (n, n) -PDL assumption, with the complexities listed for $\mathcal{R}^{V\text{-R1CS-lite}}$ in Table 2.*

The proof is completely analogous to the Plonk case and therefore omitted. Likewise, we can reduce the number of pairings as explained in Remark 4.

Acknowledgments. The authors used AI tools to help with grammar, proofreading, and formatting of the document. These tools were also used as a computational aid in Section D to reduce the number of pairings. The last three authors were supported by the AEI-PID2021-128521OB-I00 grant of the Spanish Ministry of Science and Innovation.

• $\text{KeyGen}(1^\lambda) \rightarrow \text{srs}$: output $\text{srs} = \text{ck}$. // $\text{ck} = ([1, \tau, \dots, \tau^n]_1, [1, \tau, \dots, \tau^n]_2)$ • $\text{Derive}(\text{srs}, i = (\mathbb{F}, n, n_0, V, A, B)) \rightarrow (\text{pk}_i, \text{vk}_i)$ 1. Let $i_{\text{had}} = (\mathbb{F}, n)$ and $i_{\text{lin}} = (\mathbb{F}, n, V, A, B)$, and compute $(\text{pk}_i^P, \text{vk}_i^P) \leftarrow \text{P.Derive}(\text{srs}, i_P)$ for $P \in \{\text{had}, \text{lin}\}$. 2. Let $\nu_0(X) = \prod_{\ell=1}^{n_0} (X - h_\ell)$ and $\hat{\nu}(X) = \nu_{\mathbb{H}}(X)/\nu_0(X) = \prod_{\ell=n_0+1}^n (X - h_\ell)$, and compute $[\hat{\nu}(\tau)]_1$ and $[\hat{\lambda}_\ell(\tau)]_1 = \frac{h_\ell}{n} [\hat{\nu}(\tau)]_1$ for $\hat{\nu}(X) = \hat{\nu}(X)/(X - h_\ell)$ and $\ell \in \{n_0 + 1, \dots, n\}$. 3. Output $\text{pk}_i = (\{\text{pk}_i^P\}_P, [\{\hat{\lambda}_\ell(\tau)\}_{\ell=n_0+1}^n, \hat{\nu}(\tau)]_1, [\{\lambda_\ell(\tau)\}_{\ell=1}^n, \nu_{\mathbb{H}}(\tau)]_2)$ and $\text{vk}_i = (\{\text{vk}_i^P\}_P, [\{\lambda_\ell(\tau)\}_{\ell=1}^{n_0}]_1, [\nu_0(\tau)]_2)$.	
• $\text{Prove}(\text{pk}_i, x = \mathbf{x}, w = \mathbf{w}) \rightarrow (\pi, \text{rand})$ 1. Let $\mathbf{z} = (\mathbf{x}, \mathbf{w})$, $\mathbf{z}_A = A\mathbf{z}$ and $\mathbf{z}_B = B\mathbf{z}$. 2. For $v \in \{z, z_A, z_B\}$, sample $\rho_v \leftarrow \mathbb{F}$ and compute $[v(\tau)]_1$ for $v(X) = \sum_{\ell=1}^n v_\ell \lambda_\ell(X) + \rho_v \nu_{\mathbb{H}}(X)$. Compute also $[z_B(\tau)]_2$. 3. Compute $[\hat{w}(\tau)]_1 = \sum_{\ell=n_0+1}^n z_\ell [\hat{\lambda}_\ell(\tau)]_1 + \rho_z [\hat{\nu}(\tau)]_1$. 4. Let $\mathbf{x}_{\text{had}} = ([z_A(\tau), z(\tau)]_1, [z_B(\tau)]_2)$ and $\mathbf{w}_{\text{had}} = (\mathbf{z}_A, \mathbf{z}_B, \mathbf{z}, \rho_{z_A}, \rho_{z_B}, \rho_z)$, and compute $(\pi_{\text{had}}, \perp) \leftarrow \text{Had.Prove}(\text{pk}_i, \mathbf{x}_{\text{had}}, \mathbf{w}_{\text{had}})$. 5. Let $\mathbf{x}_{\text{lin}} = [z(\tau), z_A(\tau), z_B(\tau)]_1$ and $\mathbf{w}_{\text{lin}} = (\mathbf{z}, \mathbf{z}_A, \mathbf{z}_B, \rho_z, \rho_{z_A}, \rho_{z_B})$, and compute $(\pi_{\text{lin}}, \text{rand}_{\text{lin}}) \leftarrow \text{Lin.Prove}(\text{pk}_i, \mathbf{x}_{\text{lin}}, \mathbf{w}_{\text{lin}})$. 6. Output $\pi = ([z(\tau), \hat{w}(\tau), z_A(\tau), z_B(\tau)]_1, [z_B(\tau)]_2, \pi_{\text{had}}, \pi_{\text{lin}})$ and $\text{rand} = (\rho_z, \rho_{z_A}, \rho_{z_B}, \text{rand}_{\text{lin}})$. • $\text{BuildData}(\text{pk}_i, x, w, \text{rand}) \rightarrow \text{data}$ 1. Parse $\text{rand} = (\rho_z, \rho_{z_A}, \rho_{z_B}, \text{rand}_{\text{lin}})$ and recompute $\mathbf{x}_{\text{had}}, \mathbf{w}_{\text{had}}, \mathbf{x}_{\text{lin}}, \mathbf{w}_{\text{lin}}$ as in Prove. 2. Compute $\text{data}_{\text{had}} \leftarrow \text{Had.BuildData}(\text{pk}_i, \mathbf{x}_{\text{had}}, \mathbf{w}_{\text{had}}, \perp)$ and $\text{data}_{\text{lin}} \leftarrow \text{Lin.BuildData}(\text{pk}_i, \mathbf{x}_{\text{lin}}, \mathbf{w}_{\text{lin}}, \text{rand}_{\text{lin}})$. 3. Output $\text{data} = (\text{rand}, \text{data}_{\text{had}}, \text{data}_{\text{lin}})$. • $\text{Update}(\text{pk}_i, x, w, x' = \mathbf{x}', w' = \mathbf{w}', \pi, \text{data}) \rightarrow \pi'$ 1. Parse $\pi = ([z(\tau), \hat{w}(\tau), z_A(\tau), z_B(\tau)]_1, [z_B(\tau)]_2, \pi_{\text{had}}, \pi_{\text{lin}})$ and $\text{data} = ((\rho_z, \rho_{z_A}, \rho_{z_B}, \text{rand}_{\text{lin}}), \text{data}_{\text{had}}, \text{data}_{\text{lin}})$. 2. Let $\mathbf{z}' = (\mathbf{x}', \mathbf{w}')$, $\mathbf{z}'_A = A\mathbf{z}'$ and $\mathbf{z}'_B = B\mathbf{z}'$. 3. For $v \in \{z, z_A, z_B\}$, sample $\rho'_v \leftarrow \mathbb{F}$, let $\mathbf{v}^* = \mathbf{v}' - \mathbf{v}$ and $\rho_v^* = \rho'_v - \rho_v$, and compute $[v'(\tau)]_1 = [v(\tau)]_1 + \sum_{\ell \in \text{Supp}(\mathbf{v}^*)} v_\ell^* [\lambda_\ell(\tau)]_1 + \rho_v^* [\nu_{\mathbb{H}}(\tau)]_1,$ and $[z'_B(\tau)]_2$ in the same way from $[z_B(\tau)]_2$. 4. Compute $[\hat{w}'(\tau)]_1 = [\hat{w}(\tau)]_1 + \sum_{\ell \in \text{Supp}(\mathbf{z}^*), \ell > n_0} z_\ell^* [\hat{\lambda}_\ell(\tau)]_1 + \rho_z^* [\hat{\nu}(\tau)]_1$. 5. Let $\mathbf{x}'_{\text{had}} = ([z'_A(\tau), z'(\tau)]_1, [z'_B(\tau)]_2)$, $\mathbf{w}'_{\text{had}} = (\mathbf{z}'_A, \mathbf{z}'_B, \mathbf{z}', \rho'_{z_A}, \rho'_{z_B}, \rho'_z)$ and compute $\pi'_{\text{had}} \leftarrow \text{Had.Update}(\text{pk}_i, \mathbf{x}_{\text{had}}, \mathbf{w}_{\text{had}}, \mathbf{x}'_{\text{had}}, \mathbf{w}'_{\text{had}}, \pi_{\text{had}}, \text{data}_{\text{had}})$. 6. Let $\mathbf{x}'_{\text{lin}} = [z'(\tau), z'_A(\tau), z'_B(\tau)]_1$, $\mathbf{w}'_{\text{lin}} = (\mathbf{z}', \mathbf{z}'_A, \mathbf{z}'_B, \rho'_z, \rho'_{z_A}, \rho'_{z_B})$ and compute $\pi'_{\text{lin}} \leftarrow \text{Lin.Update}(\text{pk}_i, \mathbf{x}_{\text{lin}}, \mathbf{w}_{\text{lin}}, \mathbf{x}'_{\text{lin}}, \mathbf{w}'_{\text{lin}}, \pi_{\text{lin}}, \text{data}_{\text{lin}})$. 7. Output $\pi' = ([z'(\tau), \hat{w}'(\tau), z'_A(\tau), z'_B(\tau)]_1, [z'_B(\tau)]_2, \pi'_{\text{had}}, \pi'_{\text{lin}})$. • $\text{Verify}(\text{vk}_i, x = \mathbf{x}, \pi) \rightarrow 0/1$ 1. Parse $\pi = ([z(\tau), \hat{w}(\tau), z_A(\tau), z_B(\tau)]_1, [z_B(\tau)]_2, \pi_{\text{had}}, \pi_{\text{lin}})$, compute $[x(\tau)]_1 = \sum_{\ell=1}^{n_0} x_\ell [\lambda_\ell(\tau)]_1$, and let $\mathbf{x}_{\text{had}}, \mathbf{x}_{\text{lin}}$ be as in Prove. 2. Accept if and only if the following four checks pass: $e([z(\tau)]_1 - [x(\tau)]_1, [1]_2) \stackrel{?}{=} e([\hat{w}(\tau)]_1, [\nu_0(\tau)]_2),$ $e([z_B(\tau)]_1, [1]_2) \stackrel{?}{=} e([1]_1, [z_B(\tau)]_2),$ $\text{Had.Verify}(\text{vk}_i, \mathbf{x}_{\text{had}}, \pi_{\text{had}}) \stackrel{?}{=} 1, \quad \text{Lin.Verify}(\text{vk}_i, \mathbf{x}_{\text{lin}}, \pi_{\text{lin}}) \stackrel{?}{=} 1.$	

Fig. 9. (Semi) dynamic zkSNARK for $\mathcal{R}^{V\text{-R1CS-lite}}$.

References

- ACFY25. Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. WHIR: Reed-solomon proximity testing with super-fast verification. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part IV*, volume 15604 of *LNCS*, pages 214–243. Springer, Cham, May 2025. doi:10.1007/978-3-031-91134-7_8.
- BBCT22. Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, and Nicola Tuveri. OpenSSLNTRU: Faster post-quantum TLS key exchange. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 845–862. USENIX Association, August 2022. URL: <https://www.usenix.org/conference/usenixsecurity22/presentation/bernstein>.
- BCR⁺19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Cham, May 2019. doi:10.1007/978-3-030-17653-2_4.
- CBBZ23. Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 499–530. Springer, Cham, April 2023. doi:10.1007/978-3-031-30617-4_17.
- CFF⁺21. Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 3–33. Springer, Cham, December 2021. doi:10.1007/978-3-030-92078-4_1.
- CFF⁺24. Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. Lookup arguments: Improvements, extensions and applications to zero-knowledge decision trees. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14602 of *LNCS*, pages 337–369. Springer, Cham, April 2024. doi:10.1007/978-3-031-57722-2_11.
- CFQ19. Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019. doi:10.1145/3319535.3339820.
- CHM⁺20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Cham, May 2020. doi:10.1007/978-3-030-45721-1_26.
- FK23. Dankrad Feist and Dmitry Khovratovich. Fast amortized KZG proofs. Cryptology ePrint Archive, Report 2023/033, 2023. URL: <https://eprint.iacr.org/2023/033>.
- FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018. doi:10.1007/978-3-319-96881-0_2.
- GLS⁺23. Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 193–226. Springer, Cham, August 2023. doi:10.1007/978-3-031-38545-2_7.
- Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016. doi:10.1007/978-3-662-49896-5_11.
- GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. URL: <https://eprint.iacr.org/2019/953>.
- KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Berlin, Heidelberg, December 2010. doi:10.1007/978-3-642-17373-8_11.
- Mon87. Peter L. Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.
- NS24. Ngoc Khanh Nguyen and Gregor Seiler. Greyhound: Fast polynomial commitments from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 243–275. Springer, Cham, August 2024. doi:10.1007/978-3-031-68403-6_8.
- RZ21. Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 774–804, Virtual Event, August 2021. Springer, Cham. doi:10.1007/978-3-030-84242-0_27.

- Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Cham, August 2020. doi:10.1007/978-3-030-56877-1_25.
- STW23. Srinath Setty, Justin Thaler, and Riad Wahby. Customizable constraint systems for succinct arguments. Cryptology ePrint Archive, Report 2023/552, 2023. URL: <https://eprint.iacr.org/2023/552>.
- WPSP24. Weijie Wang, Charalampos Papamanthou, Shravan Srinivasan, and Dimitrios Papadopoulos. Dynamic zk-SNARKs. Cryptology ePrint Archive, Report 2024/1566, 2024. URL: <https://eprint.iacr.org/2024/1566>.
- WPSP26a. Weijie Wang, Charalampos Papamanthou, Shravan Srinivasan, and Dimitrios Papadopoulos. Designated-verifier dynamic zk-SNARKs with applications to dynamic proofs of index. Cryptology ePrint Archive, Paper 2026/144, 2026. URL: <https://eprint.iacr.org/2026/144>.
- WPSP26b. Weijie Wang, Charalampos Papamanthou, Shravan Srinivasan, and Dimitrios Papadopoulos. Dynamic zk-SNARKs (with applications to sparse zk-SNARKs and IVC). In Joan Daemen and Emmanuel Thomé, editors, *EUROCRYPT 2026, Part VII*, volume 16547 of *LNCS*, pages 329–358. Springer, Cham, May 2026. doi:10.1007/978-3-032-25336-1_12.
- ZOZ26. Tianyu Zhang, Yupeng Ouyang, and Yupeng Zhang. Dynark: Making Groth16 dynamic. In Joan Daemen and Emmanuel Thomé, editors, *EUROCRYPT 2026, Part VII*, volume 16547 of *LNCS*, pages 178–208. Springer, Cham, May 2026. doi:10.1007/978-3-032-25336-1_7.

A Definition of Universal Fully Dynamic zkSNARKs

In this section we provide the definition of universal fully-dynamic zkSNARKs. The fundamental difference with respect to the semi-dynamic definition is that now, the proofs output by the **Update** algorithm can be updated too, and not just the ones given by **Prove**.

Definition 14. A universal fully-dynamic zero-knowledge succinct non-interactive argument of knowledge (universal fully-dynamic zk-SNARK) for an indexed relation $\mathcal{R}$ consists of the following PPT algorithms $\Pi = (\text{KeyGen}, \text{Derive}, \text{BuildData}, \text{Prove}, \text{Update}, \text{Verify})$:

- $\text{KeyGen}(1^\lambda) \rightarrow (\text{srs} = (\text{pk}, \text{vk}), \text{trap})$: On input the security parameter 1^λ , outputs the universal structured reference string srs containing the universal prover key pk , a verifier key vk and a trapdoor trap needed for simulation.
- $\text{Derive}(i, \text{srs}) \rightarrow \text{srs}_i = (\text{pk}_i, \text{vk}_i)$: On input an index i and the universal structured reference string srs , outputs specialized structured reference string srs_i containing the specialized prover key pk_i and verifier key vk_i .
- $\text{Prove}(\text{pk}_i, x, w) \rightarrow (\pi, \text{rand})$: On input the specialized prover key pk_i , valid extended witness $(i, x, w) \in \mathcal{R}$, outputs the proof π and randomness rand .
- $\text{BuildData}(\text{pk}_i, x, w, \text{rand}) \rightarrow \text{data}$: On input specialized prover key pk_i , instance x , witness w and randomness rand , outputs a data structure data used for updates.
- $\text{Update}(\text{pk}_i, x, w, x', w', \pi, \text{data}, \text{rand}) \rightarrow (\pi', \text{data}', \text{rand}')$: On input the specialized prover key pk_i , valid extended witnesses $(i, x, w), (i, x', w') \in \mathcal{R}$, proof π , preprocessed data data , and randomness rand , outputs an updated proof π' , updated preprocessed data data' and updated randomness rand' .
- $\text{Verify}(\text{vk}_i, x, \pi) \rightarrow 0/1$: On input the specialized verifier key vk_i , instance x and proof π output by **Prove** or **Update**, outputs accept or reject.

The following properties are required:

- **Completeness**: It is statistically complete if for any index i , any sequence $(i, x_0, w_0), \dots, (i, x_k, w_k) \in \mathcal{R}$, the following probability is at least $1 - \text{negl}(\lambda)$:

$$\Pr \left[\begin{array}{l} \text{Verify}(\text{vk}_i, x_0, \pi_0) = 1 \\ \text{Verify}(\text{vk}_i, x_k, \pi_k) = 1 \end{array} \middle| \begin{array}{l} (\text{srs}, \text{trap}) \leftarrow \text{KeyGen}(1^\lambda) \\ (\text{pk}_i, \text{vk}_i) \leftarrow \text{Derive}(i, \text{srs}) \\ (\pi_0, \text{rand}_0) \leftarrow \text{Prove}(\text{pk}_i, x_0, w_0) \\ \text{data}_0 \leftarrow \text{BuildData}(\text{pk}_i, x_0, w_0, \text{rand}_0) \\ \text{for } j \in \{1, \dots, k\}, \\ (\pi_j, \text{data}_j, \text{rand}_j) \leftarrow \text{Update}(\text{pk}_i, x_{j-1}, w_{j-1}, x_j, w_j, \pi_{j-1}, \text{data}_{j-1}, \text{rand}_{j-1}) \end{array} \right]$$

- **Knowledge Soundness:** It is knowledge sound if for any PPT adversary $\mathcal{A}$ there exists an extractor Ext such that:

$$\Pr \left[\begin{array}{l} \text{Verify}(\text{vk}_i, \mathbf{x}, \pi) = 1 \\ (\mathbf{i}, \mathbf{x}, \mathbf{w}) \notin \mathcal{R} \end{array} \middle| \begin{array}{l} (\text{srs}, \text{trap}) \leftarrow \text{KeyGen}(1^\lambda) \\ \mathbf{i} \leftarrow \mathcal{A}(\text{srs}) \\ (\text{pk}_i, \text{vk}_i) \leftarrow \text{Derive}(\mathbf{i}, \text{srs}) \\ (\mathbf{x}, \pi) \leftarrow \mathcal{A}(\mathbf{i}, \text{srs}) \\ \mathbf{w} \leftarrow \text{Ext}(\mathbf{i}, \text{srs}, \mathbf{x}, \pi) \end{array} \right] \leq \text{negl}(\lambda)$$

- **Zero-knowledge:** It is zero-knowledge if there exists a simulator Sim such that for all PPT adversaries $\mathcal{A}$, every index $\mathbf{i}$, every $k = \text{poly}(\lambda)$ and every sequence $(\mathbf{i}, \mathbf{x}_0, \mathbf{w}_0), \dots, (\mathbf{i}, \mathbf{x}_k, \mathbf{w}_k) \in \mathcal{R}$ we have:

$$\begin{aligned} \Pr \left[\mathcal{A}(\text{srs}, \text{trap}, \mathbf{i}, \mathbf{x}_0, \mathbf{w}_0, \mathbf{x}_k, \mathbf{w}_k, \pi_k) = 1 \middle| \begin{array}{l} (\text{srs}, \text{trap}) \leftarrow \text{KeyGen}(1^\lambda) \\ (\text{pk}_i, \text{vk}_i) \leftarrow \text{Derive}(\mathbf{i}, \text{srs}) \\ (\pi_0, \text{rand}_0) \leftarrow \text{Prove}(\text{pk}_i, \mathbf{x}_0, \mathbf{w}_0) \end{array} \right] \\ \approx \Pr \left[\mathcal{A}(\text{srs}, \text{trap}, \mathbf{i}, \mathbf{x}_0, \mathbf{w}_0, \pi'_0) = 1 \middle| \begin{array}{l} (\text{srs}, \text{trap}) \leftarrow \text{KeyGen}(1^\lambda) \\ (\text{pk}_i, \text{vk}_i) \leftarrow \text{Derive}(\mathbf{i}, \text{srs}) \\ \pi'_0 \leftarrow \text{Sim}(\text{trap}, \mathbf{i}, \mathbf{x}_0) \end{array} \right] \end{aligned}$$

and

$$\begin{aligned} \Pr \left[\mathcal{A}(\text{srs}, \text{trap}, \mathbf{i}, \mathbf{x}_0, \mathbf{w}_0, \mathbf{x}_k, \mathbf{w}_k, \pi_k) = 1 \middle| \begin{array}{l} (\text{srs}, \text{trap}) \leftarrow \text{KeyGen}(1^\lambda) \\ (\text{pk}_i, \text{vk}_i) \leftarrow \text{Derive}(\mathbf{i}, \text{srs}) \\ (\pi_k, \text{aux}_k) \leftarrow \text{Prove}(\text{pk}_i, \mathbf{x}_k, \mathbf{w}_k) \end{array} \right] \approx \\ \Pr \left[\mathcal{A}(\text{srs}, \text{trap}, \mathbf{i}, \mathbf{x}_0, \mathbf{w}_0, \mathbf{x}_k, \mathbf{w}_k, \pi_k) = 1 \middle| \begin{array}{l} (\text{srs}, \text{trap}) \leftarrow \text{KeyGen}(1^\lambda) \\ (\text{pk}_i, \text{vk}_i) \leftarrow \text{Derive}(\mathbf{i}, \text{srs}) \\ (\pi_0, \text{rand}_0) \leftarrow \text{Prove}(\text{pk}_i, \mathbf{x}_0, \mathbf{w}_0) \\ \text{data}_0 \leftarrow \text{BuildData}(\text{pk}_i, \mathbf{x}_0, \mathbf{w}_0, \text{rand}_0) \\ \text{for } j \in \{1, \dots, k\}, \\ (\pi_j, \text{data}_j, \text{rand}_j) \leftarrow \text{Update}(\text{pk}_i, \mathbf{x}_{j-1}, \mathbf{w}_{j-1}, \mathbf{x}_j, \mathbf{w}_j, \pi_{j-1}, \\ \text{data}_{j-1}, \text{rand}_{j-1}) \end{array} \right] \end{aligned}$$

- **Succinctness:** The proof size $|\pi|$ is polynomial in $(\lambda + \log |\mathbf{w}|)$ and the running time of Verify is polynomial in $(\lambda + |\mathbf{x}| + \log |\mathbf{w}|)$.
- **Updatability:** Let k be the Hamming distance between $(\mathbf{x}, \mathbf{w})$ and $(\mathbf{x}', \mathbf{w}')$. If $k = o(|\mathbf{x}| + |\mathbf{w}|)$, then the running time of Update is sublinear in the running time of Prove .

B Cost analysis of Dynaverse and Full Comparison

In this section we will discuss the costs of Dynaverse [WPSP26b]. The number of pairings performed in Verify and the proof size will be given for the non-universal and non-zero-knowledge construction based on symmetric pairings, but the complexities will be given for the universal version. As a reference for this discussion, we follow Figure 4 from the full version [WPSP24].

The semi-dynamic Update algorithm has a term $O(k \log^2 k)$ that comes from the computation of an update for a Hadamard relation, and a term $O(k \log n)$ that arises from the universal variant of their consistency checks (see Appendix D of [WPSP24] and Theorem 11 therein). Their fully-dynamic construction is given in the amortized setting, in which the prover has to recompute the anchor proof whenever the witness gets b -far from the preceding one, for $b = \sqrt{n \log n}$ (see the discussion in Section 4 of [WPSP24]). Under the assumption that consecutive witnesses are at Hamming distance $\leq k$, the fully-dynamic Update algorithm yields a cost of $O(k \sqrt{n \log n} + \sqrt{n \log n} \log^2 n)$.

For the proof size, observe that a proof in Dynaverse is split into π^{ini} and π^{upd} . The former consists of 6 consistency proofs (each one consisting on 7 $\mathbb{G}_1$, see Figure 8 from [WPSP24] but removing the first zero check, since it is not needed in the Dynaverse construction), a permutation check (consisting on 5 $\mathbb{G}_1$, see Figure 3 from [WPSP24]) and 10 further polynomial commitments in $\mathbb{G}_1$. This gives a size for π^{ini} of 57 $\mathbb{G}_1$. The latter consists of 6 consistency checks, 1 permutation check and 27 further commitments in $\mathbb{G}_1$, which gives a proof size for π^{upd} of 74 $\mathbb{G}_1$. In total, we get a proof size of 131 $\mathbb{G}_1$. The verification consists of 30

initial pairings, which consist mostly on gate constraint checks, followed by 2 permutation verifications (each consisting on 9 pairings, see again Figure 3 from [WPSP24]) and 12 consistency checks (each consisting on 11 pairings, see again Figure 8 from [WPSP24], but removing the first pairing check). This gives a total verifier computation of 180 pairings.

Table 3. Comparison with existing dynamic zkSNARKs.

Work	PROPERTIES			COMPLEXITY				
	Constraint system	Univ.	PV	$ \pi $	Prove	Update semi-dyn.	Update fully-dyn.	Verify
Dynark [ZOZ26]	R1CS	✗	✓	$2\mathbb{G}_1 + \mathbb{G}_2$	$n \log n$	$k \log^2 k$	$k\sqrt{n \log n} + k \log^2 k$	$4\mathbb{P}$
Delphus [WPSP26a]	R1CS	✗	✗	$2\mathbb{G}_1 + \mathbb{G}_2 + \mathbb{G}_T$	$n \log n$	—	$k \log n$	$4\mathbb{P}$
Dynalog [WPSP26b]	Plonk variant	✓	✓	$\log^3 n$	$n \log^2 n$	—	$k \log^3 n$	$\log^3 n$
Dynaverse [WPSP26b]	Plonk variant	✓	✓	$131 \mathbb{G}_1$	$n \log n$	$k \log n + k \log^2 k$	$k\sqrt{n \log n} + \sqrt{n \log n} \log^2 n$	$\log n$ $180 \mathbb{P}$
<i>This work</i>								
Section 6.1	Plonk	✓	✓	$19 \mathbb{G}_1 + \mathbb{G}_2 + 2\mathbb{F}$	$n \log n$	$\log n + k \log^2 k$	$k\sqrt{n \log n} + k \log^2 k$	$\log n$ $23 \mathbb{P}$ (13 batched)
Section 6.2	V-R1CS-lite	✓	✓	$18 \mathbb{G}_1 + \mathbb{G}_2 + 2\mathbb{F}$	$n \log n$	$\log n + kV \log^2(kV)$	$kV\sqrt{n \log n} + kV \log^2(kV)$	$\log n + V$ $19 \mathbb{P}$ (10 batched)

Note: All costs are given in O notation. k denotes the Hamming distance between the updated witness and the original one, V denotes the fan-out of the circuit, $\mathbb{P}$ denotes a pairing, and $|\pi|$ denotes the size of an updated proof. PV stands for publicly verifiable. Verify costs are given on top of reading the instance, and the batched pairing counts for our schemes are those of Section D.

C De-amortization of the Plonk Linear zkSNARK

In this section we describe how to compile the semi-dynamic zkSNARK of Section 4.3 into a fully-dynamic protocol in the de-amortized setting; an analogous argument applies to the protocol of Section 4.4 and, more generally, to any instantiation of the general protocol Figure 3. By Section 2.1 it suffices to show that **BuildData** admits a step-wise evaluation, which then allows us to invoke Theorem 1.

Throughout this section we reserve k for the Hamming distance between two consecutive instance-witness pairs—the quantity that appears in the update cost—and write k_0 for the sparsity of the *base* witness on which **BuildData** is run. For a dense witness $k_0 = n$, which is the regime of interest here.

Recall that the semi-dynamic zkSNARK of Section 4.3 is obtained by first designing a sparse zkSNARK and then compiling it via Theorem 2. Inspecting Figure 1, the **BuildData** algorithm consists of the re-randomization $(x^{\text{ini}}, w^{\text{ini}}) = (x, w) - Z(\text{rand})$ followed by a single call to **Sparse.Prove**. The first part is trivially step-wise: because the witness vectors of a null pair are $\mathbf{0}$, it leaves w, f_w, f_{w^2} untouched and only shifts the $l + 1$ commitments and openings, at a cost of $O(l) = O(1)$ group operations. Our task is therefore to show that the prover of Figure 3, specialized to the rational encoding RE^{Plonk} of Lemma 9, can be executed step-wise.

Overview of the step-wise computation. Thanks to the sparsity analysis of Section 4.2, the **Prove** algorithm of Figure 3 consists essentially of multi-scalar multiplications against precomputed group elements. Since MSMs admit step-wise evaluation by construction, the only real obstacle is the sequential nature of the Random Oracle challenges. We overcome this by splitting the prover’s work into large, challenge-independent

MSMs that are accumulated incrementally, and cheap $O(1)$ challenge-dependent combinations performed at the very first and the very last step.

More concretely, the challenges $r, \delta \leftarrow \text{RO}$ are derived from $(\mathbf{vk}_i, \mathbf{x}^{\text{ini}}, [S(\tau)]_1)$, all of which are available at the outset: the instance and the verification key are inputs to **BuildData**, and $[S(\tau)]_1 = \rho_{S,1}[\tau]_1 + \rho_{S,2}[\nu_{\mathbb{H}}(\tau)]_1$ is computed in $O(1)$ operations. Hence r and δ can be obtained in the very first step, after which the prover may start accumulating the commitments to $p(X)$ and $\tilde{p}(X)$.

The same strategy does not apply directly to $Q(X)$ and $G(X)$, since both depend on the challenge $\eta \leftarrow \text{RO}$, which can only be derived once the commitments to $p(X)$ and $\tilde{p}(X)$ are complete—and these cannot be computed in constant time. Fortunately, Lemma 6 gives

$$[G(\tau)]_1 = \sum_{\ell=1}^n (p(\mathbf{h}_\ell) - \tilde{p}(\mathbf{h}_\ell)) [G_\ell(\tau)]_1 + \eta \rho_{S,1}[\tau^2 - 1]_1,$$

and the summation is entirely independent of η . The prover may therefore accumulate $\sum_\ell (p(\mathbf{h}_\ell) - \tilde{p}(\mathbf{h}_\ell)) [G_\ell(\tau)]_1$ in parallel with $p(X)$ and $\tilde{p}(X)$, and add the single remaining term $\eta \rho_{S,1}[\tau^2 - 1]_1$ in the final step, in $O(1)$ operations. The same holds for $Q(X) = q(X) + \eta \tilde{q}(X)$: by Lemmas 7 and 8 both $q(X)$ and $\tilde{q}(X)$ are independent of η , so $[q(\tau)]_1$ and $[\tilde{q}(\tau)]_1$ are accumulated alongside the other commitments and combined into $[Q(\tau)]_1 = [q(\tau)]_1 + \eta [\tilde{q}(\tau)]_1$ in constant time at the end.

Fixing notation. We provide the concrete shape of the prover's scalars and group elements of Figure 3 under the rational encoding RE^{Plonk} , which will help us next.

It will be convenient to abbreviate the coordinates over the Lagrange basis of $p(X)$ from Step 3 as

$$p(\mathbf{h}_\ell) = P_\ell \lambda_\ell(r) \quad P_\ell := w(\mathbf{h}_\ell) + \delta f_\omega(\mathbf{h}_\ell) + \delta^2 f_{\omega^2}(\mathbf{h}_\ell).$$

Also, since $\omega^n = 1$ we have $\lambda_\ell(\omega^i r) = \frac{r^n - 1}{n} \cdot \frac{\mathbf{h}_\ell}{\omega^i r - \mathbf{h}_\ell}$ for $i \in \{1, 2\}$, so (15) and Lemma 5 give the coordinates of $\tilde{p}(X)$ from Step 4

$$\tilde{p}(\mathbf{h}_\ell) = w(\mathbf{h}_\ell) \tilde{C}_\ell, \quad \tilde{C}_\ell := \lambda_{\sigma^{-1}(\ell)}(r) + \delta \lambda_\ell(\omega r) + \delta^2 \lambda_\ell(\omega^2 r).$$

Moreover, applying Remark 1 to $\mathcal{X} = (1, X, X^2)$ —which yields $[\lambda_\ell(X)/\nu_{\mathbb{H}}(X)] = 0$, $[\lambda_\ell(X)X/\nu_{\mathbb{H}}(X)] = \frac{\mathbf{h}_\ell}{n}$ and $[\lambda_\ell(X)X^2/\nu_{\mathbb{H}}(X)] = \frac{\mathbf{h}_\ell}{n}(X + \mathbf{h}_\ell)$ —the quotients $[\lambda_\ell(X)\hat{I}_r(X)/\nu_{\mathbb{H}}(X)]$ and $[\lambda_\ell(X)\hat{R}_{\delta,r}(X)/\nu_{\mathbb{H}}(X)]$ are committed as

$$\begin{aligned} [\text{den}_\ell]_1 &= -[q_\ell^{(3)}(\tau)]_1 + (\omega + \omega^2)r[q_\ell^{(2)}(\tau)]_1 - \omega^3 r^2[q_\ell^{(1)}(\tau)]_1 + r \frac{\mathbf{h}_\ell}{n}([\tau]_1 + \mathbf{h}_\ell[1]_1) - (\omega + \omega^2)r^2 \frac{\mathbf{h}_\ell}{n}[1]_1, \\ [\text{num}_\ell]_1 &= (1 + \delta + \delta^2)[q_\ell^{(3)}(\tau)]_1 - ((\omega + \omega^2) + \delta\omega^2 + \delta^2\omega)r[q_\ell^{(2)}(\tau)]_1 + \omega^3 r^2[q_\ell^{(1)}(\tau)]_1 - (\delta + \delta^2)r \frac{\mathbf{h}_\ell}{n}([\tau]_1 + \mathbf{h}_\ell[1]_1) \\ &\quad + (\delta\omega^2 + \delta^2\omega)r^2 \frac{\mathbf{h}_\ell}{n}[1]_1. \end{aligned} \tag{32}$$

Each is formed with $O(1)$ group operations from $\{[q_\ell^{(s)}(\tau)]_1\}_{s=1}^3, [\tau]_1, [1]_1$ given in $\mathbf{pk}_i$ (see Equation (18)), so Lemma 8 costs $O(k + \log n)$ in total.

The step-wise computation. Write $\text{Supp}(w, \mathbf{f}_\omega, \mathbf{f}_{\omega^2}) := \text{Supp}(w) \cup \text{Supp}(\mathbf{f}_\omega) \cup \text{Supp}(\mathbf{f}_{\omega^2})$ and split both this set and $\text{Supp}(w)$ into b chunks of asymptotically equal size,

$$\text{Supp}(w) = \bigsqcup_{j=1}^b \text{Supp}(w, j), \quad \text{Supp}(w, \mathbf{f}_\omega, \mathbf{f}_{\omega^2}) = \bigsqcup_{j=1}^b \text{Supp}((w, \mathbf{f}_\omega, \mathbf{f}_{\omega^2}), j),$$

so that both $|\text{Supp}(w, j)|$ and $|\text{Supp}((w, \mathbf{f}_\omega, \mathbf{f}_{\omega^2}), j)|$ are $O(\frac{k_0}{b})$ for every $j \in [b]$.¹⁶

The prover proceeds as follows.

- **Step 1.** Sample $\rho_{S,1}, \rho_{S,2} \leftarrow \mathbb{F}$, compute $[S(\tau)]_1 = \rho_{S,1}[\tau]_1 + \rho_{S,2}[\nu_{\mathbb{H}}(\tau)]_1$ and obtain $r, \delta \leftarrow \text{RO}$. Compute r^n with $O(\log n)$ operations and initialize $[p_0]_1 = [\tilde{p}_0]_1 = [q_0]_1 = [\tilde{q}_0]_1 = [g_0]_1 = [0]_1$. Then proceed as in the generic step below with $j = 1$.

¹⁶ For the lincheck of Section 4.4 the corresponding bounds are $O(\frac{k_0}{b})$ and $O(\frac{k_0 V}{b})$, respectively.

– **Step** $j = 1, \dots, b$. Compute

$$\begin{aligned}
[p_j]_1 &= [p_{j-1}]_1 + \sum_{\ell \in \text{Supp}((\mathbf{w}, \mathbf{f}_\omega, \mathbf{f}_{\omega^2}), j)} P_\ell \lambda_\ell(r) [\lambda_\ell(\tau)]_1 \\
[\tilde{p}_j]_1 &= [\tilde{p}_{j-1}]_1 + \sum_{\ell \in \text{Supp}(\mathbf{w}, j)} w(\mathbf{h}_\ell) \tilde{C}_\ell [\lambda_\ell(\tau)]_1 \\
[q_j]_1 &= [q_{j-1}]_1 - \sum_{\ell \in \text{Supp}((\mathbf{w}, \mathbf{f}_\omega, \mathbf{f}_{\omega^2}), j)} P_\ell \frac{\mathbf{h}_\ell}{n} (n \lambda_\ell(r) + (r^n - 1)) [1]_1 \\
[\tilde{q}_j]_1 &= [\tilde{q}_{j-1}]_1 + \sum_{\ell \in \text{Supp}(\mathbf{w}, j)} w(\mathbf{h}_\ell) \left(n \tilde{C}_\ell [\text{den}_\ell]_1 - (r^n - 1) [\text{num}_\ell]_1 \right) \\
[g_j]_1 &= [g_{j-1}]_1 + \sum_{\ell \in \text{Supp}((\mathbf{w}, \mathbf{f}_\omega, \mathbf{f}_{\omega^2}), j)} P_\ell \lambda_\ell(r) [G_\ell(\tau)]_1 - \sum_{\ell \in \text{Supp}(\mathbf{w}, j)} w(\mathbf{h}_\ell) \tilde{C}_\ell [G_\ell(\tau)]_1.
\end{aligned}$$

Once r^n is available, every scalar above is obtained in $O(1)$ field operations as in Lemmas 4, 5, 7 and 8, so the cost of a step is $O(\frac{k_0}{b})$, plus the one-off $O(\log n)$ paid in step 1.

– **Final step** ($j = b$), **continued**. Sample $\rho_p, \tilde{\rho}_p \leftarrow \mathbb{F}$ and compute

$$\begin{aligned}
[p(\tau)]_1 &= [p_b]_1 + \rho_p [\nu_{\mathbb{H}}(\tau)]_1, & [\tilde{p}(\tau)]_1 &= [\tilde{p}_b]_1 + \tilde{\rho}_p [\nu_{\mathbb{H}}(\tau)]_1, \\
[q(\tau)]_1 &= [q_b]_1 + n \rho_p [r - \tau]_1 - (\rho_w + \delta \rho_{f_\omega} + \delta^2 \rho_{f_{\omega^2}}) (r^n - 1) [\tau]_1, \\
[\tilde{q}(\tau)]_1 &= [\tilde{q}_b]_1 + n \tilde{\rho}_p [\hat{I}_r(\tau)]_1 - (r^n - 1) \rho_w [\hat{R}_{\delta, r}(\tau)]_1,
\end{aligned}$$

where, by (17),

$$\begin{aligned}
[\hat{I}_r(\tau)]_1 &= -[e_3(\tau)]_1 + (\omega + \omega^2) r [e_2(\tau)]_1 - \omega^3 r^2 [e_1(\tau)]_1 + r [\tau^2]_1 - (\omega + \omega^2) r^2 [\tau]_1 + \omega^3 r^3 [1]_1, \\
[\hat{R}_{\delta, r}(\tau)]_1 &= (1 + \delta + \delta^2) [e_3(\tau)]_1 - ((\omega + \omega^2) + \delta \omega^2 + \delta^2 \omega) r [e_2(\tau)]_1 \\
&\quad + \omega^3 r^2 [e_1(\tau)]_1 - (\delta + \delta^2) r [\tau^2]_1 + (\delta \omega^2 + \delta^2 \omega) r^2 [\tau]_1.
\end{aligned}$$

All of this takes $O(1)$ operations. Finally obtain $\eta \leftarrow \text{RO}$ and compute

$$[G(\tau)]_1 = [g_b]_1 + \eta \rho_{S,1} [\tau^2 - 1]_1, \quad [Q(\tau)]_1 = [q(\tau)]_1 + \eta [\tilde{q}(\tau)]_1, \quad h = \rho_p - \tilde{\rho}_p + \eta \rho_{S,2},$$

again in $O(1)$ operations. Output $\pi^{\text{ini}} = ([S(\tau), p(\tau), \tilde{p}(\tau), G(\tau), Q(\tau)]_1, h)$.

Observe that the only state carried between steps consists of the five accumulators $[p_j, \tilde{p}_j, q_j, \tilde{q}_j, g_j]_1$ together with $r, \delta, \rho_{S,1}, \rho_{S,2}$ and r^n ; that is, $O(1)$ group and field elements on top of the witness itself. We summarize the discussion in the following proposition.

Proposition 10. *Let BuildData be the algorithm of Figure 1 instantiated with the sparse zkSNARK of Corollary 1, run on a base witness of sparsity k_0 . For every $b \leq k_0$, BuildData can be evaluated in b steps, the j -th of which performs $O(\frac{k_0}{b})$ operations for $j \geq 2$ and $O(\frac{k_0}{b} + \log n)$ for $j = 1$, and carries $O(1)$ group and field elements of state between consecutive steps. The same holds for the sparse zkSNARK of Corollary 4, with $O(\frac{k_0 V}{b})$ operations per step.*

Choice of the parameter b . With Proposition 10 in hand we may invoke Theorem 1, which yields a de-amortized update cost of

$$O\left(k \frac{\mathsf{T}(\text{BuildData})}{b} + \mathsf{T}(\text{Update}_{\text{semi}, b})\right).$$

Recall the shape of the schedule: the sequence of instance-witness pairs is divided into blocks spanning Hamming distance b , and the updates inside a block are performed against the data associated to the *first pair of the previous block*, while the data for the current block is being assembled one step at a time. The distance between a pair and the base pair against which it is updated is therefore at most $2b$ rather than b ,

which does not affect the asymptotics. Each block contains b/k updates and **BuildData** is split into b steps, so each update runs k of them.

For the semi-dynamic lincheck of $\mathcal{R}_{\text{ck}}^{\text{Plonk}, \text{lin}}$ we have $\mathsf{T}(\text{BuildData}) = O(k_0 + \log n)$, which is $O(n)$ for a dense base witness, while Theorem 2 together with Corollary 1 gives $\mathsf{T}(\text{Update}_{\text{semi}, b}) = O(b + \log n)$. The update cost is thus

$$O\left(\frac{kn}{b} + b + \log n\right),$$

which is minimized at $b = \sqrt{kn}$, giving $O(\sqrt{kn} + \log n)$. However, we instead choose

$$b = \sqrt{n \log n},$$

the threshold required by the Hadamard product protocol of [ZOZ26], so that a single schedule governs both protocols; note that this is also what makes the $O(\log n)$ of the first step harmless, since $\frac{k_0}{b} = \sqrt{n/\log n} = \Omega(\log n)$ for every $n \geq \log^3 n$, so all b steps have the same asymptotic cost, as Theorem 1 requires. Substituting, we obtain the following.

Theorem 10. *There exists a fully-dynamic zkSNARK for $\mathcal{R}_{\text{ck}}^{\text{Plonk}, \text{lin}}$ in the AGM and the ROM under the (n, n) -PDL assumption whose update time in the de-amortized setting, with $b = \sqrt{n \log n}$, is*

$$O(k\sqrt{n/\log n} + \sqrt{n \log n}) = O(k\sqrt{n \log n}),$$

where k is the Hamming distance between consecutive instance-witness pairs.

An identical discussion applies to the protocol of Section 4.4, giving us the following.

Theorem 11. *There exists a fully-dynamic zkSNARK for $\mathcal{R}_{\text{ck}}^{\text{lin}, V\text{-R1CS-lite}}$ in the AGM and the ROM under the (n, n) -PDL assumption whose update time in the de-amortized setting, with $b = \sqrt{n \log n}$, is*

$$O(kV\sqrt{n/\log n} + V\sqrt{n \log n}) = O(kV\sqrt{n \log n}).$$

Theorems 10 and 11 are the linear-relation components of the fully-dynamic update costs reported in Theorem 8 and Theorem 9. In both cases they are dominated by the cost of the fully-dynamic Hadamard protocol of Theorem 6, which is precisely why aligning the two thresholds at $b = \sqrt{n \log n}$ costs nothing overall.

D Optimizing the Verifier Checks

In this appendix we justify the batched pairing counts claimed in our tables. Every check performed by our verifiers is a pairing equation, and the only source of savings is bilinearity in the first argument: several pairings sharing a *common* $\mathbb{G}_2$ point collapse into one, at the price of a few operations in $\mathbb{G}_1$. The whole appendix is therefore a rewriting of the verification equations so that as few distinct $\mathbb{G}_2$ points as possible occur.

Conventions. We write $\mathbb{G}_T$ additively, with neutral element $[0]_T$, and we use bilinearity in the two forms

$$e(x[A]_1, y[B]_2) = xy e([A]_1, [B]_2), \quad \sum_i e([A_i]_1, [B]_2) = e\left(\sum_i [A_i]_1, [B]_2\right), \quad (34)$$

for $x, y \in \mathbb{F}$. A check $\text{LHS} \stackrel{?}{=} \text{RHS}$ is always brought to the homogeneous form $\mathcal{P} = [0]_T$ with $\mathcal{P} = \text{LHS} - \text{RHS}$, and then to the *grouped form*

$$\mathcal{P} = \sum_{[Y]_2} e(A_Y, [Y]_2),$$

where $[Y]_2$ ranges over the distinct $\mathbb{G}_2$ points appearing in $\mathcal{P}$ and $A_Y \in \mathbb{G}_1$ collects, via the second identity of (34), every first argument paired with $[Y]_2$. The number of pairings is then the number of summands of the grouped form. We abbreviate $[v^\bullet]_1 := [v^\bullet(\tau)]_1$ for the witness-dependent commitments of a run, keeping τ explicit for the fixed points such as $[\nu_{\mathbb{H}}(\tau)]_2$. Finally, note that e_1, e_2, e_3 denote the Plonk index polynomials of (16), not the pairing $e(\cdot, \cdot)$.

D.1 Batching the Plonk verifier

A 13-pairing verifier. The verifier of Figure 7 runs four checks:

- (i) the cross-group consistency of $[f_\omega(\tau)]_1$ and $[f_\omega(\tau)]_2$;
- (ii) **Had.Verify** on the instance $x_{\text{had}} = ([w(\tau), \varphi(\tau)]_1, [f_\omega(\tau)]_2)$;
- (iii) **Gates.Verify** on x_{gates} ;
- (iv) **Lin.Verify**, which by Figure 1 consists of two runs of **Sparse.Verify**, one on (x^*, π^*) and one on $(x^{\text{ini}}, \pi^{\text{ini}})$.

We start with (iv), which dominates the count.

A single run of **Sparse.Verify**. Fix $\bullet \in \{*, \text{ini}\}$. The run has instance $x^\bullet = [w^\bullet, f_\omega^\bullet, f_{\omega^2}^\bullet]_1$ and proof

$$\pi^\bullet = ([S^\bullet, p^\bullet, \tilde{p}^\bullet, G^\bullet, Q^\bullet]_1, h^\bullet).$$

Write $(r^\bullet, \delta^\bullet, \eta^\bullet)$ for the challenges that Figure 3 derives from the random oracle on that run's transcript and $\xi^\bullet \leftarrow \mathbb{F}$ for the randomness sampled by the verifier. Specialising (7) to RE^{Plonk} , that is, to $l = 3$, $\mathbf{a} = \mathbf{w}$ and $(\mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \mathbf{b}^{(3)}) = (\mathbf{w}, \mathbf{f}_\omega, \mathbf{f}_{\omega^2})$, and moving every term to the left-hand side, the check reads $\mathcal{P}^\bullet = [0]_T$ for

$$\begin{aligned} \mathcal{P}^\bullet = & e(n[p^\bullet]_1, [r^\bullet - \tau]_2) + \eta^\bullet e(n[\tilde{p}^\bullet]_1, [\hat{I}_{r^\bullet}(\tau)]_2) \\ & - \eta^\bullet e(((r^\bullet)^n - 1)[w^\bullet]_1, [\hat{R}_{\delta^\bullet, r^\bullet}(\tau)]_2) \\ & + \xi^\bullet e(T^\bullet, [\tau^2 - 1]_2) - e([Q^\bullet]_1 + \xi^\bullet h^\bullet[\tau^2 - 1]_1, [\nu_{\mathbb{H}}(\tau)]_2) \\ & - e(((r^\bullet)^n - 1)B^\bullet + \xi^\bullet[G^\bullet]_1, [\tau]_2), \end{aligned} \quad (35)$$

where $T^\bullet = [p^\bullet]_1 - [\tilde{p}^\bullet]_1 + \eta^\bullet[S^\bullet]_1$ and $B^\bullet = [w^\bullet]_1 + \delta^\bullet[f_\omega^\bullet]_1 + (\delta^\bullet)^2[f_{\omega^2}^\bullet]_1$, the latter being the Plonk form of $\sum_{j=1}^3 \delta^{j-1}[b^{(j)}(\tau)]_1$ recorded after Definition 8.

Equation (35) has six pairings, but three of its $\mathbb{G}_2$ arguments—namely $[r^\bullet - \tau]_2$, $[\hat{I}_{r^\bullet}(\tau)]_2$ and $[\hat{R}_{\delta^\bullet, r^\bullet}(\tau)]_2$ —depend on the run because of the challenges, so nothing can be shared between the two runs as it stands. We remove this dependency with three substitutions.

- (1) $[r^\bullet - \tau]_2 = r^\bullet[1]_2 - [\tau]_2$, so that, by (34),

$$e(n[p^\bullet]_1, [r^\bullet - \tau]_2) = e(nr^\bullet[p^\bullet]_1, [1]_2) + e(-n[p^\bullet]_1, [\tau]_2).$$

- (2) $[\tau^2 - 1]_2 = [\tau^2]_2 - [1]_2$, so that

$$\xi^\bullet e(T^\bullet, [\tau^2 - 1]_2) = e(\xi^\bullet T^\bullet, [\tau^2]_2) + e(-\xi^\bullet T^\bullet, [1]_2).$$

- (3) By (17),

$$[\hat{I}_{r^\bullet}(\tau)]_2 = \sum_{s=1}^6 \alpha_s(r^\bullet)[f_s(\tau)]_2, \quad [\hat{R}_{\delta^\bullet, r^\bullet}(\tau)]_2 = \sum_{s=1}^6 \beta_s(r^\bullet, \delta^\bullet)[f_s(\tau)]_2.$$

Expanding the second and third pairings of (35) by bilinearity in the second argument gives twelve pairings, which regroup by index s into

$$\begin{aligned} & \sum_{s=1}^6 e(U_s^\bullet, [f_s(\tau)]_2), \quad \text{for} \\ U_s^\bullet := & \eta^\bullet \left(n\alpha_s(r^\bullet)[\tilde{p}^\bullet]_1 - ((r^\bullet)^n - 1)\beta_s(r^\bullet, \delta^\bullet)[w^\bullet]_1 \right). \end{aligned}$$

The split of the rational encoding into index polynomials and monomials is exactly what makes (3) profitable. Since $(f_1, \dots, f_6) = (e_1, e_2, e_3, 1, X, X^2)$ by Lemma 9, the last three points are $[f_4(\tau)]_2 = [1]_2$, $[f_5(\tau)]_2 = [\tau]_2$ and $[f_6(\tau)]_2 = [\tau^2]_2$, which (1) and (2) have already produced; only the $t = 3$ index polynomials contribute new $\mathbb{G}_2$ points, and those are independent of the run. Explicitly, reading α and β off Lemma 9,

$$\begin{aligned} U_4^\bullet &= \eta^\bullet n\omega^3(r^\bullet)^3[\tilde{p}^\bullet]_1, \\ U_5^\bullet &= -\eta^\bullet(r^\bullet)^2 \left(n(\omega + \omega^2)[\tilde{p}^\bullet]_1 + ((r^\bullet)^n - 1)(\delta^\bullet\omega^2 + (\delta^\bullet)^2\omega)[w^\bullet]_1 \right), \\ U_6^\bullet &= \eta^\bullet r^\bullet \left(n[\tilde{p}^\bullet]_1 + ((r^\bullet)^n - 1)(\delta^\bullet + (\delta^\bullet)^2)[w^\bullet]_1 \right), \end{aligned}$$

the absence of a $[w^\bullet]_1$ term in $U_4^\bullet$ being due to $\beta_4 = 0$.

Applying the substitutions (1)–(3) to (35) and collecting, for each of the seven $\mathbb{G}_2$ points $[1, \tau, \tau^2, \nu_{\mathbb{H}}(\tau), e_1(\tau), e_2(\tau), e_3(\tau)]_2$, the $\mathbb{G}_1$ elements paired with it, we obtain $\mathcal{P}^\bullet = [0]_T$ with

$$\begin{aligned} \mathcal{P}^\bullet = & e(nr^\bullet[p^\bullet]_1 - \xi^\bullet T^\bullet + U_4^\bullet, [1]_2) + e(-n[p^\bullet]_1 - ((r^\bullet)^n - 1)B^\bullet - \xi^\bullet[G^\bullet]_1 + U_5^\bullet, [\tau]_2) + e(\xi^\bullet T^\bullet + U_6^\bullet, [\tau^2]_2) \\ & + e(-[Q^\bullet]_1 - \xi^\bullet h^\bullet[\tau^2 - 1]_1, [\nu_{\mathbb{H}}(\tau)]_2) + \sum_{s=1}^3 e(U_s^\bullet, [e_s(\tau)]_2). \end{aligned} \quad (36)$$

Taken in isolation this is one pairing more than (35); the gain is that all seven $\mathbb{G}_2$ arguments are now fixed by vk_i , so the two runs share them and *both* runs together cost seven pairings instead of twelve. Forming the seven first arguments takes $O(1)$ operations in $\mathbb{G}_1$: each is a fixed linear combination of a constant number of elements of $\pi^\bullet$, $x^\bullet$ and vk_i , and each scalar is evaluated with $O(1)$ field operations once $(r^\bullet)^n$ is known, which costs $O(\log n)$.

The other three checks. In homogeneous form they read

$$\mathcal{P}_{\text{had}} = e([w(\tau)]_1, [f_\omega(\tau)]_2) - e([\varphi(\tau)]_1, [1]_2) - e(\pi_{\text{had}}, [\nu_{\mathbb{H}}(\tau)]_2), \quad (37)$$

$$\mathcal{P}_{\text{gates}} = \sum_{(J,v)} e([v(\tau)]_1, [q_J(\tau)]_2) + e([q_C(\tau)]_1 - [x(\tau)]_1, [1]_2) - e(\pi_{\text{gates}}, [\nu_{\mathbb{K}}(\tau)]_2), \quad (38)$$

$$\mathcal{P}_{\text{cg}} = e([f_\omega(\tau)]_1, [1]_2) - e([1]_1, [f_\omega(\tau)]_2), \quad (39)$$

where (J, v) ranges over the four pairs (L, w) , (R, f_ω) , (M, φ) , (O, f_{ω^2}) , and $[x(\tau)]_1 = \sum_{i=1}^{n_0} x_i [\mu_i(\tau)]_1$ is recomputed by the verifier. Note the sign in (38): the check of Figure 6 carries $[x(\tau)]_1 - [q_C(\tau)]_1$ on its right-hand side.

Merging. The verifier samples $\zeta_2, \dots, \zeta_5 \leftarrow \mathbb{F}$ and checks

$$\mathcal{P}^* + \zeta_2 \mathcal{P}^{\text{ini}} + \zeta_3 \mathcal{P}_{\text{had}} + \zeta_4 \mathcal{P}_{\text{gates}} + \zeta_5 \mathcal{P}_{\text{cg}} = [0]_T, \quad (40)$$

which is equivalent to the five original checks up to a soundness loss of $1/|\mathbb{F}|$. Folding each coin into the first arguments—writing $\zeta_* := 1$, $\zeta_{\text{ini}} := \zeta_2$ and letting $\sum_\bullet$ range over $\bullet \in \{*, \text{ini}\}$ —and grouping the result by $\mathbb{G}_2$ argument yields the following thirteen pairs $([Y]_2, A_Y)$:

$$\begin{aligned} [1]_2 : & \sum_\bullet \zeta_\bullet (nr^\bullet[p^\bullet]_1 - \xi^\bullet T^\bullet + U_4^\bullet) - \zeta_3 [\varphi(\tau)]_1 \\ & + \zeta_4 ([q_C(\tau)]_1 - [x(\tau)]_1) + \zeta_5 [f_\omega(\tau)]_1, \\ [\tau]_2 : & \sum_\bullet \zeta_\bullet (-n[p^\bullet]_1 - ((r^\bullet)^n - 1)B^\bullet - \xi^\bullet[G^\bullet]_1 + U_5^\bullet), \\ [\tau^2]_2 : & \sum_\bullet \zeta_\bullet (\xi^\bullet T^\bullet + U_6^\bullet), \\ [\nu_{\mathbb{H}}(\tau)]_2 : & \sum_\bullet \zeta_\bullet (-[Q^\bullet]_1 - \xi^\bullet h^\bullet[\tau^2 - 1]_1) - \zeta_3 \pi_{\text{had}}, \\ [e_s(\tau)]_2 : & \sum_\bullet \zeta_\bullet U_s^\bullet \quad (s = 1, 2, 3), \\ [f_\omega(\tau)]_2 : & \zeta_3 [w(\tau)]_1 - \zeta_5 [1]_1, \\ [q_J(\tau)]_2 : & \zeta_4 [v(\tau)]_1 \quad ((J, v) \text{ as in (38)}), \\ [\nu_{\mathbb{K}}(\tau)]_2 : & -\zeta_4 \pi_{\text{gates}}. \end{aligned}$$

Since $\mathcal{X} = (1, X, X^2)$, the first seven points are precisely those of (19); the eighth is the $\mathbb{G}_2$ commitment carried by π ; and the last five belong to vk_i by Theorem 7. Hence 13 pairings in total.

Written out in full, the single equation the verifier checks is

$$\begin{aligned}
& e\left(\sum_{\bullet} \zeta_{\bullet}(nr^{\bullet}[p^{\bullet}]_1 - \xi^{\bullet}T^{\bullet} + U_4^{\bullet}) - \zeta_3[\varphi(\tau)]_1 \right. \\
& \quad \left. + \zeta_4([q_C(\tau)]_1 - [x(\tau)]_1) + \zeta_5[f_{\omega}(\tau)]_1, [1]_2\right) \\
& + e\left(\sum_{\bullet} \zeta_{\bullet}(-n[p^{\bullet}]_1 - ((r^{\bullet})^n - 1)B^{\bullet} - \xi^{\bullet}[G^{\bullet}]_1 + U_5^{\bullet}), [\tau]_2\right) \\
& + e\left(\sum_{\bullet} \zeta_{\bullet}(\xi^{\bullet}T^{\bullet} + U_6^{\bullet}), [\tau^2]_2\right) \\
& + e\left(\sum_{\bullet} \zeta_{\bullet}(-[Q^{\bullet}]_1 - \xi^{\bullet}h^{\bullet}[\tau^2 - 1]_1) - \zeta_3\pi_{\text{had}}, [\nu_{\mathbb{H}}(\tau)]_2\right) \\
& + \sum_{s=1}^3 e\left(\sum_{\bullet} \zeta_{\bullet}U_s^{\bullet}, [e_s(\tau)]_2\right) + e(\zeta_3[w(\tau)]_1 - \zeta_5[1]_1, [f_{\omega}(\tau)]_2) \\
& + \sum_{(J,v)} e(\zeta_4[v(\tau)]_1, [q_J(\tau)]_2) - e(\zeta_4\pi_{\text{gates}}, [\nu_{\mathbb{K}}(\tau)]_2) \stackrel{?}{=} [0]_T,
\end{aligned} \tag{41}$$

with (J, v) as in (38). Every $\mathbb{G}_1$ argument is formed with $O(1)$ scalar multiplications and additions, and all thirteen $\mathbb{G}_2$ arguments are read directly off $\mathbf{vk}_i$ and π .

Remark 5 (The unbatched count). Performed independently, the four checks cost

$$\underbrace{2}_{\text{cross-group}} + \underbrace{3}_{\text{Had.Verify}} + \underbrace{6}_{\text{Gates.Verify}} + \underbrace{2 \cdot 6}_{\text{Lin.Verify}} = 23$$

pairings, spread over 16 distinct $\mathbb{G}_2$ points. Seven of them are redundant: the point $[1]_2$ is used three times, by the cross-group, Hadamard and gates checks; $[\nu_{\mathbb{H}}(\tau)]_2$ three times, by the Hadamard check and by both runs; and each of $[f_{\omega}(\tau)]_2$, $[\tau]_2$ and $[\tau^2 - 1]_2$ twice. Grouping alone thus gives 16. The substitutions (1)–(3) then eliminate seven further points—the six run-dependent arguments of (35), together with $[\tau^2 - 1]_2$ —in exchange for $[1]_2$ and $[\tau]_2$, which are already in use, and for the four new points $[\tau^2]_2$, $[e_1(\tau)]_2$, $[e_2(\tau)]_2$, $[e_3(\tau)]_2$; hence $16 - 7 + 4 = 13$.

D.2 Batching the V-R1CS-lite verifier

A 10-pairing verifier. The verifier of Figure 9 runs four checks: the public-input check (31), the cross-group consistency of $[z_B(\tau)]_1$ and $[z_B(\tau)]_2$, **Had.Verify** on $\mathbf{x}_{\text{had}} = ([z_A(\tau), z(\tau)]_1, [z_B(\tau)]_2)$, and **Lin.Verify**, consisting again of two runs of **Sparse.Verify**.

A single run of Sparse.Verify. Equation (35) holds verbatim, now with $\mathbf{a} = \mathbf{z}$, $l = 2$ and $(\mathbf{b}^{(1)}, \mathbf{b}^{(2)}) = (\mathbf{z}_A, \mathbf{z}_B)$, so that $B^{\bullet} = [z_A^{\bullet}]_1 + \delta^{\bullet}[z_B^{\bullet}]_1$. Substitutions (1) and (2) are unchanged. Substitution (3), however, is *not* applied here. Indeed, for $\text{RE}^{V\text{-R1CS-lite}}$ we have $u = 0$ and $t = 3V + 1$, so decomposing over (23) would introduce $3V + 1$ new $\mathbb{G}_2$ points; even though these are index polynomials and hence shared by the two runs, **Lin.Verify** would then cost $4 + (3V + 1) = 3V + 5$ pairings, against the $4 + 2 \cdot 2 = 8$ obtained by leaving $[\hat{I}_{r^{\bullet}}(\tau)]_2$ and $[\hat{R}_{\delta^{\bullet}, r^{\bullet}}(\tau)]_2$ intact and paying two run-specific pairings per run. The latter is smaller for every $V \geq 2$ and, unlike the former, does not grow with $V = O(\log n)$.

Accordingly, the verifier forms the four run-specific $\mathbb{G}_2$ points itself from (25), as in Corollary 4:

$$\begin{aligned}
[\hat{I}_{r^{\bullet}}(\tau)]_2 &= \sum_{j=0}^V (r^{\bullet})^j [v_j^{\hat{I}}(\tau)]_2, \\
[\hat{R}_{\delta^{\bullet}, r^{\bullet}}(\tau)]_2 &= \sum_{j=0}^{V-1} (r^{\bullet})^j \left([v_j^{\hat{R}, A}(\tau)]_2 + \delta^{\bullet} [v_j^{\hat{R}, B}(\tau)]_2 \right),
\end{aligned}$$

that is, two multi-scalar multiplications of total size $t = 3V + 1$ in $\mathbb{G}_2$ per run. Applying only substitutions (1) and (2) to (35) and grouping, the run reduces to

$$\begin{aligned} \mathcal{P}^\bullet = & e(nr^\bullet[p^\bullet]_1 - \xi^\bullet T^\bullet, [1]_2) + e(-n[p^\bullet]_1 - ((r^\bullet)^n - 1)B^\bullet - \xi^\bullet[G^\bullet]_1, [\tau]_2) \\ & + e(\xi^\bullet T^\bullet, [\tau^2]_2) + e(-[Q^\bullet]_1 - \xi^\bullet h^\bullet[\tau^2 - 1]_1, [\nu_{\mathbb{H}}(\tau)]_2) \\ & + e(\eta^\bullet n[\tilde{p}^\bullet]_1, [\hat{L}_{r^\bullet}(\tau)]_2) - e(\eta^\bullet((r^\bullet)^n - 1)[z^\bullet]_1, [\hat{R}_{\delta^\bullet, r^\bullet}(\tau)]_2), \end{aligned} \quad (42)$$

which is exactly (36) with $U_4^\bullet = U_5^\bullet = U_6^\bullet = 0$ and the sum over the index polynomials replaced by the last two pairings.

The other three checks. In homogeneous form,

$$\mathcal{P}_{\text{had}} = e([z_A(\tau)]_1, [z_B(\tau)]_2) - e([z(\tau)]_1, [1]_2) - e(\pi_{\text{had}}, [\nu_{\mathbb{H}}(\tau)]_2), \quad (43)$$

$$\mathcal{P}_{\text{pub}} = e([z(\tau)]_1 - [x(\tau)]_1, [1]_2) - e([\hat{w}(\tau)]_1, [\nu_0(\tau)]_2), \quad (44)$$

$$\mathcal{P}_{\text{cg}} = e([z_B(\tau)]_1, [1]_2) - e([1]_1, [z_B(\tau)]_2). \quad (45)$$

Merging. The verifier samples $\zeta_2, \dots, \zeta_5 \leftarrow \mathbb{F}$ and checks

$$\mathcal{P}^* + \zeta_2 \mathcal{P}^{\text{ini}} + \zeta_3 \mathcal{P}_{\text{had}} + \zeta_4 \mathcal{P}_{\text{pub}} + \zeta_5 \mathcal{P}_{\text{cg}} = [0]_T, \quad (46)$$

again equivalent to the five checks up to $1/|\mathbb{F}|$ soundness error. With the same conventions $\zeta_* = 1$, $\zeta_{\text{ini}} = \zeta_2$, grouping by $\mathbb{G}_2$ argument gives the following ten pairs, where $[\hat{L}_{r^\bullet}(\tau)]_2$ and $[\hat{R}_{\delta^\bullet, r^\bullet}(\tau)]_2$ each represent two arguments for $\bullet \in \{*, \text{ini}\}$:

$$\begin{aligned} [1]_2 : & \sum_{\bullet} \zeta_{\bullet} (nr^\bullet[p^\bullet]_1 - \xi^\bullet T^\bullet) + (\zeta_4 - \zeta_3)[z(\tau)]_1 \\ & - \zeta_4[x(\tau)]_1 + \zeta_5[z_B(\tau)]_1, \\ [\tau]_2 : & \sum_{\bullet} \zeta_{\bullet} (-n[p^\bullet]_1 - ((r^\bullet)^n - 1)B^\bullet - \xi^\bullet[G^\bullet]_1), \\ [\tau^2]_2 : & \sum_{\bullet} \zeta_{\bullet} \xi^\bullet T^\bullet, \\ [\nu_{\mathbb{H}}(\tau)]_2 : & \sum_{\bullet} \zeta_{\bullet} (-[Q^\bullet]_1 - \xi^\bullet h^\bullet[\tau^2 - 1]_1) - \zeta_3 \pi_{\text{had}}, \\ [\hat{L}_{r^\bullet}(\tau)]_2 : & \zeta_{\bullet} \eta^\bullet n[\tilde{p}^\bullet]_1, \\ [\hat{R}_{\delta^\bullet, r^\bullet}(\tau)]_2 : & -\zeta_{\bullet} \eta^\bullet((r^\bullet)^n - 1)[z^\bullet]_1, \\ [z_B(\tau)]_2 : & \zeta_3[z_A(\tau)]_1 - \zeta_5[1]_1, \\ [\nu_0(\tau)]_2 : & -\zeta_4[\hat{w}(\tau)]_1. \end{aligned}$$

The distinct $\mathbb{G}_2$ arguments are the four shared points $[1, \tau, \tau^2, \nu_{\mathbb{H}}(\tau)]_2$, the four run-specific ones, and $[z_B(\tau)]_2, [\nu_0(\tau)]_2$: 10 pairings.

Written out in full, the single equation the verifier checks is

$$\begin{aligned} & e\left(\sum_{\bullet} \zeta_{\bullet} (nr^\bullet[p^\bullet]_1 - \xi^\bullet T^\bullet) + (\zeta_4 - \zeta_3)[z(\tau)]_1 - \zeta_4[x(\tau)]_1 + \zeta_5[z_B(\tau)]_1, [1]_2\right) \\ & + e\left(\sum_{\bullet} \zeta_{\bullet} (-n[p^\bullet]_1 - ((r^\bullet)^n - 1)B^\bullet - \xi^\bullet[G^\bullet]_1), [\tau]_2\right) + e\left(\sum_{\bullet} \zeta_{\bullet} \xi^\bullet T^\bullet, [\tau^2]_2\right) \\ & + e\left(\sum_{\bullet} \zeta_{\bullet} (-[Q^\bullet]_1 - \xi^\bullet h^\bullet[\tau^2 - 1]_1) - \zeta_3 \pi_{\text{had}}, [\nu_{\mathbb{H}}(\tau)]_2\right) \\ & + \sum_{\bullet} \left(e(\zeta_{\bullet} \eta^\bullet n[\tilde{p}^\bullet]_1, [\hat{L}_{r^\bullet}(\tau)]_2) - e(\zeta_{\bullet} \eta^\bullet((r^\bullet)^n - 1)[z^\bullet]_1, [\hat{R}_{\delta^\bullet, r^\bullet}(\tau)]_2)\right) \\ & + e(\zeta_3[z_A(\tau)]_1 - \zeta_5[1]_1, [z_B(\tau)]_2) - e(\zeta_4[\hat{w}(\tau)]_1, [\nu_0(\tau)]_2) \stackrel{?}{=} [0]_T. \end{aligned} \quad (47)$$

Beyond the $O(1)$ scalar multiplications forming the $\mathbb{G}_1$ arguments, the only extra work is the $6V + 2$ scalar multiplications in $\mathbb{G}_2$ that produce the four run-specific points.

Remark 6 (The unbatched count). Performed independently, the four checks cost

$$\underbrace{2}_{\text{public input}} + \underbrace{2}_{\text{cross-group}} + \underbrace{3}_{\text{Had. Verify}} + \underbrace{2 \cdot 6}_{\text{Lin. Verify}} = 19$$

pairings, spread over 12 distinct $\mathbb{G}_2$ points: $[1]_2$ occurs in the public-input, cross-group and Hadamard checks, $[z_B(\tau)]_2$ in the cross-group and Hadamard checks, $[\nu_{\mathbb{H}}(\tau)]_2$ in the Hadamard check and in both runs, and $[\tau]_2$, $[\tau^2 - 1]_2$ in both runs. Grouping alone therefore gives 12, and substitutions (1)–(2) eliminate the three points $[r^* - \tau]_2$, $[r^{\text{ini}} - \tau]_2$ and $[\tau^2 - 1]_2$ in exchange for the single new point $[\tau^2]_2$; hence $12 - 3 + 1 = 10$.

E Extension to Higher Degree Gates

Thus far we have targeted constraint systems for circuits with only addition and multiplication gates of arity 2. In this section we give a high level overview of how the techniques developed in this paper can be generalized in a straightforward manner to cover more expressive constraint systems (for simplicity of presentation we omit details such as public input binding and cross-group checks). In particular, we are interested in the *customizable constraints systems* (CCS) of Setty, Thaler and Wahby [STW23] which covers circuits with higher degree gates. Its defining equation has the form

$$\sum_{i=1}^q c_i \left(M^{(j_{i,1})} \mathbf{z} \circ \dots \circ M^{(j_{i,d_i})} \mathbf{z} \right) = \mathbf{0}, \quad (48)$$

for scalar coefficients c_i and terms of arity $d_i \leq d$. As with the case of R1CS-lite we make the further, natural assumption that the circuit is of bounded fan-out V . As is evident from Equation (48), constructing a dynamic zkSNARK for CCS boils down to proving and updating (i) the linear equations $\mathbf{z}^{(j)} = M^{(j)} \mathbf{z}$, for $j \in [l]$ (where l is the number of distinct CCS matrices, as in Section 4; we reserve t for the number of index polynomials of Definition 7), and (ii) a multi-Hadamard equation $\sum_{i=1}^q c_i (\mathbf{z}^{(j_{i,1})} \circ \dots \circ \mathbf{z}^{(j_{i,d_i})}) = \mathbf{0}$, for committed vectors $\mathbf{z}^{(j)}$. To this end, we describe how our techniques from Section 3, Section 4, and Section 6 can be applied.

Matrices as a stack. The lincheck of Section 4.4 stacks A over B , bounds the amount of non-zero entries of the *stack* by V , and places both numerators of each column over one common denominator determined by the column’s non-zero rows. The same construction applies to the stack $(M^{(1)}; \dots; M^{(l)})$ of all CCS matrices: one common denominator of degree at most V per column, now carrying l numerators, so that the number t of index polynomials grows from $3V+1$ (the case $l = 2$ of Section 4.4) to $(l+1)V+1$. Theorem 3 then applies unchanged and, through the compiler of Section 3, yields a semi-dynamic zkSNARK proving all l linear relations $\mathbf{z}^{(j)} = M^{(j)} \mathbf{z}$ at once. As before, bounding the stack rather than each matrix keeps the denominator degree at V , retaining its interpretation as the fan-out of the circuit.

Iterating the Plonk trick. In Section 6.1 we replaced the triple product of the Plonk’s gate equation $q_M(X)w(X)w(\omega X)$ by one committed two-multiplication plus a relation that is linear in the remaining polynomials. We did so because a pairing verifies exactly one multiplication of committed values. When terms have degree $d_i > 2$, the same trick is applied repeatedly rather than once: for the i -th term $\mathbf{z}^{(j_{i,1})} \circ \dots \circ \mathbf{z}^{(j_{i,d_i})}$ the prover commits to the *running* partial products of the term, that is, to the vectors

$$\varphi^{(i,2)} = \mathbf{z}^{(j_{i,1})} \circ \mathbf{z}^{(j_{i,2})}, \quad \varphi^{(i,\ell)} = \varphi^{(i,\ell-1)} \circ \mathbf{z}^{(j_{i,\ell})} \quad \text{for } 3 \leq \ell \leq d_i,$$

so that $\varphi^{(i,d_i)} = \mathbf{z}^{(j_{i,1})} \circ \dots \circ \mathbf{z}^{(j_{i,d_i})}$ is the full term.

Costs. The higher multiplicative degree is paid for during the application of our Plonk trick. Whereas in Plonk this costs one extra commitment and one Hadamard equation, here each term costs $d_i - 1$ of each, for a total of $\sum_i (d_i - 1)$: proof size and verifier pairings grow additively by $O(q(d-1))$, and proving time by the cost of $\sum_i (d_i - 1)$ Hadamard proofs. Updates are affected twice. First, as for R1CS-lite, a k -entry change of $\mathbf{z}$ touches at most $k(1+V)$ entries across the $\mathbf{z}^{(j)}$ vectors. Second, changes *accumulate* along the Hadamard chain: the difference in a partial product is supported on the union of the differences of its two vectors, so the update support of a partial product grows with its level. Summed over the $d_i - 1$ levels of the chain, the total Hadamard update support is inflated by a factor of at most $d - 1$. The linear layer inherits

the costs of Theorem 3 with $t = (l+1)V+1$ index polynomials: key sizes, derivation, proving and updating grow linearly in lV , while the linear relation zkSNARK still contributes $O(1)$ proof elements and pairings. In short, all costs grow linearly in the number of matrices (through the stacked encoding) and linearly in the degree of the terms (through the repeated applications of the trick).