Accordion: Efficient Batch Proving via Algebraic Folding

David Balbás[1], Anca Nitulescu[2], and Maxime Plançon[3]

[1] ETH Zürich, Switzerland
[2] Input Output, France
[3] Input Output, Switzerland

Abstract. We present Accordion, a new proof system that works in two steps:

- *Accumulate:* The first building block is a multi-folding scheme that efficiently accumulates k different instances of a degree-d polynomial relation into a single accumulated instance, without using recursion. Our scheme builds on the Protostar [Bünz & Chen, ASIACRYPT '23] and Protogalaxy [Eagen & Gabizon, ePrint 2023/1106] line of work, improving further the verifier efficiency. It supports the same expressive class of relations, arbitrary-degree polynomial constraints, enabling efficient folding for CCS, Plonkish constraints, and even custom gates.
- *Decide:* The second block is a SNARK decider for the folding scheme, which efficiently proves the validity of the resulting accumulated instance using off-the-shelf optimized building blocks. We give a concrete instantiation consisting of a Plonk-based proof plus an additional KZG polynomial commitment opening.

This design is especially attractive in batch-aggregation settings where a prover collects many independent witnesses, folds them into a single accumulator, and then runs one succinct decider to certify the entire batch. Our scheme avoids costly recursion overheads.

We implement Accordion and evaluate it against a highly optimized Plonk proving pipeline (`midnight-zk`). For $k = 64$ instances, Accordion is up to 2.5× faster than computing k Plonk proofs for SHA-256 preimages, with a 3× faster verifier and 4× smaller proofs. Its prover is also an order of magnitude faster than using IVC to aggregate the k proofs.

1 Introduction

Folding schemes [9,21] are cryptographic protocols that aggregate multiple instances of a constraint satisfaction relation (such as R1CS, Plonkish, or CCS) into a single *accumulated* instance. Folding schemes are run between a prover, which knows k valid instance-witness pairs $(\mathbf{t}_1, \mathbf{w}_1), \ldots, (\mathbf{t}_k, \mathbf{w}_k)$, and a verifier, which has only the instances $(\mathbf{t}_1, \ldots, \mathbf{t}_k)$. At the end of the protocol, the prover ends up with an accumulated instance $(\mathbf{t}^*, \mathbf{w}^*)$ and the verifier with $\mathbf{t}^*$. Folding schemes are formally captured by the *reductions of knowledge* framework [19]: given an accepting proof, if $(\mathbf{t}^*, \mathbf{w}^*)$ is a valid accumulated instance, then one can

extract a set of valid initial witnesses. Essentially, a valid folding proof reduces the verification of many claims to the verification of one.

The main gain of folding is to defer proving work to, instead, accumulate multiple instances into one of the same size. Later, one can prove the validity of the final accumulated instance by running what we call a *decider*. This idea can be carried out in practice in two different settings:

- *Iterative folding.* After applying many folding steps iteratively, a single succinct decider can be used to check the accumulated witness. This technique, which dates back to recursive SNARKs [7], enables efficient incrementally verifiable computation (IVC) [31], or more generally Proof-Carrying Data (PCD) [11].
- *One-shot folding.* k different instances of the same relation are combined into an accumulator that cannot be folded further with fresh instances, but for which an efficient decider is available. The advantage of this technique is that *folding* k instances requires much less work than *proving* k instances (and possibly aggregate the proofs later, such as in SnarkPack [15]), being a promising approach for building efficient proof systems for batch relation. Nevertheless, only two such constructions exist, both of them for R1CS: Symphony, from lattices [10], and FLIP-and-Prove, from pairing groups [27].

Taking a closer look at the literature, the main bottleneck towards instantiating folding-based batch proving systems seems to be precisely the *lack of efficient deciders* for the accumulator relation. While many recent works focus on optimizing the folding itself, they construct folding schemes whose accumulators live in non-standard relations for which a succinct decider is not built. General-purpose SNARKs, such as Plonk [14] (for algebraic folding [8,12]) and Spartan [29] (for sumcheck-based folding [22,20,17]) can serve as generic deciders, but they need to be tailored to these accumulator relations, making them expensive in practice.

1.1 Contributions

In this work, we aim to address this shortcoming by building Accordion, an efficient batch proving pipeline composed of a one-shot folding scheme and an efficient decider. Accordion supports very generic polynomial relations that seamlessly and efficiently capture well-established constraint systems such as R1CS, CCS or Plonkish, including lookups and custom gates.

Step 1: Accumulate via Algebraic Folding. The first step is to compute the accumulator using a multi-folding scheme. Our folding step is built for a flexible relation, defined by some polynomial map $f : \mathbb{F}^M \to \mathbb{F}^n$ of some arbitrary degree d in which the witnesses to be folded $\{\mathbf{w}_i\}_{i\in[k]}$ satisfy $f(\mathbf{w}_i) = \mathbf{0}_n$. More concretely, our techniques support folding transcripts of special-sound protocols à la Protostar [8].

Our folding step is a variant of Protogalaxy [12] that we optimize for one-shot folding of k instances. In particular, we modify Protogalaxy to yield a more structured algebraic relation in the accumulator. Thanks to this modification, we shave off an additive $\mathcal{O}(\log n)$ factor from the proof size and the verifier time of Protogalaxy, as well as shorten the protocol by one round (in practice removing one random oracle call).

Step 2: Prove the Accumulator via a Decider. The second step is to build an efficient decider for the final accumulated witness whose verifier runs in constant time. We provide a general framework for building such deciders from any off-the-shelf SNARK and polynomial commitment scheme. Given a concrete SNARK, Accordion can fold the extended witnesses in the arithmetization system the SNARK is using—such as CCS, R1CS, or Plonkish—using the one-shot folding scheme tailored to the polynomial commitment scheme (PCS) the SNARK uses. In Step 2, we use the SNARK itself as a decider, together with an extra polynomial commitment opening.

We show a concrete instantiation that works over a Plonk [14]/Halo2 [7] proof system with lookups and custom gates. In this context, the decider is simply a full Plonk proof and a KZG [18] PCS opening. For this instantiation, we also introduce an optimization that enables us to reduce the degree of the relation that is folded in the first step (which impacts prover performance and proof size) with respect to folding vanilla Plonk arithmetizations.

Implementation: Halo2-compatible, benchmarking using `midnight-zk`. We implement Accordion as an extension to the `midnight-zk` [24] library, a highly optimized Plonk/Halo2 library running in production. Thus, we test our implementation on Plonkish constraints (including high-degree gates and lookups), and use a KZG-based optimized ZK-proof available in the library as a decider. Our implementation works seamlessly on all `midnight-zk` supported circuits and examples, and we argue that, while still being a research prototype, is not too far from production-ready code. We benchmark Accordion for different number of folding instances $k = 2, \ldots, 64$ against two baselines: k parallel ZK-proofs for same instances, and k iterations of the IVC scheme available in `midnight-zk`. We find that Accordion is several orders of magnitude faster than IVC in prover time and about 30-40% faster than separate proving. On the verification side, IVC naturally remains faster and more compact (not by far), but the Accordion proof size and verifier runtime is 3x smaller than the separate zk-proofs baseline.

The code can be found at [3].

Extension: an algebraic PCD-friendly folding scheme for accumulators. As a side result, in Section C we present a technique that extends the algebraic folding approach in Accordion to support the folding of multiple accumulators at once. This extension enables another application for Accordion: a tool for an efficient folding-based proof-carrying data (PCD) [11] scheme, where Accordion (that has minimal prover overhead) is used to fold instances in a first layer of the PCD graph, after which these accumulators are re-folded with the extension scheme.

Because our construction is compatible with both Accordion and Protogalaxy accumulators, it applies even when the accumulators originate from different computation graphs. This makes it a generic PCD building block for a wide range of applications.

Folding accumulators is a result which we find of independent interest, especially since there exist several optimized implementations of Protogalaxy that could be easily extended to support PCD.[4]

Zero-Knowledge. As Accordion is implemented over `midnight-zk`, all commitments are hiding and all polynomials are masked in the actual implementation. A future work direction is therefore analyzing and formalizing zero-knowledge.

1.2 Applications and Related Work

In applications such as distributed ledgers and ZK-rollups, a common goal is to either aggregate multiple proofs, or to produce compact batch proofs from the start. Existing solutions for proof aggregation include schemes such as SnarkPack [15], SnarkBlock [28], or aPlonk [1]. These are batching k proofs into an aggregated form requiring $O(\log k)$ verification costs, reducing on-chain computation (known as gas consumption) by orders of magnitude. However, the tradeoff is in the total proving time, that includes the full cost of generating k individual proofs, plus the aggregation step. Also, while logarithmic, the proof size of such schemes inherits the large size of target group elements.

To reduce prover costs, there is a growing interest in building folding schemes for expressive high-degree relations with these applications in mind. In this context, the bottleneck is not only the lack of efficient deciders, but also having constructions that attain end-to-end compatibility with existing proving schemes and verification tools available on the systems deployed today. Accordion does satisfy the compatibility requirements for Plonk/Halo2, as many blockchains and Layer 2 networks employ proof stacks in the Plonk family, including Zcash with the first implementation of Halo2 [32], Ethereum, Scroll, Cardano, Midnight, zkSync, and others[5].

Regarding folding schemes, several of them such as HyperNova [22], NeutronNova [20], Protostar [8], Protogalaxy [12] or lattice-based schemes [4,5,6,16,25] can naturally accumulate general relations, employing either algebraic folding or sumcheck linearization, but their accumulated relations do not come with a concrete efficient decider.

SnarkFold [23] or Mangrove [26] propose folding-based IVC/PCD to achieve constant-size proofs, independent of the k iterations. This strategy has different goals, focusing on optimizing proof sizes and saving on memory footprint. However, they inevitably require the verifier algorithms to be arithmetized within the recursive circuit, thereby introducing substantial prover overheads.

[4] Note that Protogalaxy [12, Section 5] already gives a solution for folding many accumulators using sumcheck, but it does not extend to the algebraic construction.

[5] https://z.cash, https://ethereum.org, https://docs.scroll.io, https://cardano.org, https://docs.midnight.network, https://zksync.io

2 Preliminaries

2.1 Notation

We denote by $\mathbb{N}$ the set of natural numbers $\geqslant 0$. We denote the security parameter by $\lambda \in \mathbb{N}$. We call a function ϵ *negligible*, denoted $\epsilon(\lambda) = \mathsf{negl}(\lambda)$, if $\epsilon(\lambda) = O(\lambda^{-c})$ for every constant $c > 0$, and call a function $p(\lambda)$ *polynomial*, denoted $\mathsf{poly}(\lambda)$, if $p(\lambda) = O(\lambda^c)$ for some constant $c > 0$. We say that an algorithm is *probabilistic polynomial time* (PPT) if it consumes randomness and its running time is bounded by some $p(\lambda) = \mathsf{poly}(\lambda)$. For a finite set S, $x \leftarrow\!\!\$\ S$ denotes sampling x uniformly at random in S. For $n \in \mathbb{N}, [n]$ is the set $[1, n]$. We denote vectors using bold fonts. For a vector $\boldsymbol{x}$, x_i denotes its i-th component. We use the notation $\mathcal{R}$ to denote an efficiently decidable binary relation. For pairs $(t, w) \in \mathcal{R}$ we call t the statement and w the witness. We write $\mathcal{R} = \{(t; w) : p(t, w)\}$ to describe an NP relation.

2.2 Special Sound Protocols

We refer to special-soundness and non-interactive arguments as defined by [2].

Definition 1 (Public-coin interactive proof). *An interactive proof $\Pi = (\mathsf{P}, \mathsf{V})$ for relation $\mathcal{R}$ is an interactive protocol between two probabilistic machines, a prover P, and a polynomial time verifier V. Both P and V take as public input a statement $\boldsymbol{x}$ and, additionally, P takes as private input a witness $\boldsymbol{w}$ such that $(\boldsymbol{x}, \boldsymbol{w}) \in \mathcal{R}$. The verifier V outputs 1 if it accepts and 0 otherwise. Its output is denoted by $(\mathsf{P}(\boldsymbol{x}, \boldsymbol{w}), \mathsf{V}(\boldsymbol{x}))$. The protocol is public coin if the verifier randomness is public. Π is a $(2\mu+1)$-move protocol if there are $\mu+1$ prover messages a_i and μ verifier challenges c_i. We refer to the transcripts as $\mathsf{tr} = (a_0, c_1, a_1, \ldots, c_\mu, a_\mu)$.*

We define special-sound public-coin interactive proofs in Definition 11, in the appendix. In this work, we will often refer to transcripts of special-sound protocols as *witnesses*, and simply denote them by $\mathbf{w}$. Our folding schemes will actually fold these transcripts. For now on, we follow the conventions from Protostar [8] for committed transcripts and for committed relations. We introduce below the notions of committed transcripts and hash consistency with respect to the Fiat-Shamir transformation used in [8,12].

Definition 2 (Committed Transcripts). *For special-sound protocols with a transcript $\mathbf{w} = (a_0, c_1, a_1, \ldots, c_\mu, a_\mu)$, we denote their commited transcript by*

$$\mathbf{t} = \mathsf{cm}(\mathbf{w}) = (\mathsf{Com}(a_0), c_1, \mathsf{Com}(a_1), \ldots, c_\mu, \mathsf{Com}(a_\mu))$$

where cm is a shorthand for the operation of obtaining a committed transcript from a standard transcript.

Finally, we say that a transcript is hash-consistent if it has been obtained by a correct application of the Fiat-Shamir transform, as follows:

Definition 3 (Hash consistency). *Let* $\mathsf{H}(\cdot)$ *be a hash function mapping to the challenge space and modeled as a random oracle. We say that a* μ*-round commited transcript* $\mathbf{t}$ *is* $\mathcal{H}$*-consistent if:*

- *Let* $\mathbf{t}_i = (\mathsf{Com}(a_0), c_1, \mathsf{Com}(a_1), \ldots, \mathsf{Com}(a_{i-1}))$ *be the partial (committed) transcript of* $\mathbf{t}$ *up to the* i*-th round.*
- *For every* $i \in [\mu]$*, it holds that* $c_i = \mathsf{H}(\mathbf{t}_i)$*.*

For simplicity, we simply write $\mathcal{H}(\boldsymbol{t}) = 1$ *if* $\mathbf{t}$ *is* $\mathcal{H}$ *consistent for* H*.*

2.3 Reductions of Knowledge and Folding Schemes

We now review the *reductions of knowledge* framework, as introduced by Kothapalli and Parno [19].

Definition 4 (Reduction of Knowledge [19]). *A reduction of knowledge from* $\mathcal{R}_1 \to \mathcal{R}_2$ *is a tuple of three PPT algorithms* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *defined as follows:*

- $\mathcal{G}(1^\lambda, 1^\ell) \to \mathsf{p}$*: Takes as input a security parameter* 1^λ *and a size bound* 1^ℓ *and outputs public parameters* p*, which we assume are input to all algorithms.*
- $\mathcal{P}(t_1, w_1) \to (t_2, w_2)$*: Takes as input an instance-witness pair* (t_1, w_1) *and interactively reduces the task of checking* $(t_1, w_1) \in \mathcal{R}_1$ *to the task of checking* $(t_2, w_2) \in \mathcal{R}_2$*.*
- $\mathcal{V}(\mathsf{p}, t_1) \to t_2$*: Takes as input an instance* t_1 *in* $\mathcal{R}_1$ *and interactively reduces the task of checking the instance* t_1 *to the task of checking a new instance* t_2 *in* $\mathcal{R}_2$*.*

Let $\langle \mathcal{P}, \mathcal{V} \rangle(t_1, w_1) \to (t_2, w_2)$ *denote the interaction between* $\mathcal{P}$ *and* $\mathcal{V}$*. The reduction must satisfy the following properties:*

Completeness. *For any admissible set of parameters* $\lambda, \ell \in \mathbb{N}$*, given* $\mathsf{p} \leftarrow \mathcal{G}(1^\lambda, 1^\ell)$ *and* $(t_1, w_1) \leftarrow \mathcal{A}(\mathsf{p})$ *such that* $(t_1, w_1) \in \mathcal{R}_1$*, the prover's output instance must be equal to the verifier's output instance* t_2*, and* $\Pr[\langle \mathcal{P}, \mathcal{V} \rangle(t_1, w_1) \in \mathcal{R}_2] = 1$*.*

Knowledge soundness. *For any expected* PPT *adversary* $\mathcal{A}$*, there exists an expected poly-time extractor* $\mathcal{E}$ *with rewinding access to* $\mathcal{A}$ *such that if the success probability of the adversary*

$$\epsilon(\mathcal{A}) := \Pr\left[\langle \mathcal{A}, \mathcal{V} \rangle(t_1) \in \mathcal{R}_2 \;\middle|\; \begin{array}{l} \mathsf{p} \leftarrow \mathcal{G}(1^\lambda, 1^\ell) \\ t_1 \leftarrow \mathcal{A}(\mathsf{p}) \end{array}\right] \geqslant \frac{1}{\mathsf{poly}(\lambda)},$$

then

$$\Pr\left[(t_1, w_1) \in \mathcal{R}_1 \;\middle|\; \begin{array}{l} \mathsf{p} \leftarrow \mathcal{G}(1^\lambda, 1^\ell) \\ t_1 \leftarrow \mathcal{A}(\mathsf{p}) \\ w_1 \leftarrow \mathcal{E}(\mathsf{p}, t_1) \end{array}\right] \geqslant \epsilon(\mathcal{A}) - \mathsf{negl}(\lambda).$$

Public coin. *All verifier messages are bounded-length random messages, and the verifier does not use any other random coins beyond those used for sampling these messages.*

3 Algebraic Folding Techniques and Relations

We construct a folding scheme for special-sound protocols where the verification checks can be expressed as some polynomial relation. This captures a large family of relations, including R1CS, Plonkish or CCS (as described in Protostar [8]), and even custom gates and lookup gates. Our departing point is Protogalaxy [12] folding scheme that we improve in multiple aspects for one-shot executions.

3.1 Main Intuition for Algebraic Folding

Protogalaxy folds a set of k witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_k$ that satisfy $f(\mathbf{w}) = 0^n$ for some polynomial map $f : \mathbb{F}^M \to \mathbb{F}^n$ of a bounded degree d. Such witnesses are combined with an accumulator $\mathbf{w}_0$, which is a witness that satisfies a relaxed relation $\mathcal{R}_{\mathsf{rand}}$, into a fresh accumulator $\mathbf{w}^*$. The first key technique that makes Protogalaxy efficient is that witnesses are aggregated using the algebraic properties of the Lagrange polynomials. Given $\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_k$, the folding scheme works by defining a polynomial

$$W(X) = L_0(X)\mathbf{w}_0 + \sum_{i \in [k]} L_i(X)\mathbf{w}_i$$

where $L_0(X), \ldots, L_k(X)$ are a basis of Lagrange polynomials over some domain $\mathbb{H}$ of size $k+1$. Then, given a random challenge $\gamma \in \mathbb{F}$ sent by the verifier, the prover sets $\mathbf{w}^* = W(\gamma)$ as the new accumulator. This strategy minimizes the number of rounds of interaction between prover and verifier.

The second key insight in Protogalaxy is the way "rows" are compressed. Accumulators $\mathbf{w}^* \in \mathbb{F}^M$ are large vectors, and so are their evaluations $f(\boldsymbol{w}^*) \in \mathbb{F}^n$. These evaluations cannot be sent to the verifier in the clear since they are too large, so the protocol compresses them into a single error term $e = \sum_{i \in [n]} b_i f_i(\boldsymbol{w}^*)$ using a vector of randomizers $\boldsymbol{b} \in \mathbb{F}^n$. This is not sufficient, however, as now the verifier needs to know the vector $\boldsymbol{b}$, which is also large. To resolve this, $\boldsymbol{b}$ is *structured*: it is expanded from a smaller seed $\boldsymbol{\beta} \in \mathbb{F}^t$, where $t = \lceil \log n \rceil$, using a tensor product structure $\boldsymbol{b} = \bigotimes_{i=1}^{t}(1, \beta_i)$. Thus, $\boldsymbol{\beta}$ is the only vector that appears on the instance, keeping the work of the verifier logarithmic in the witness size.

We observe that we can further reduce the verifier's costs from logarithmic in n to constant by modifying the structure of the randomizers. We can use any family of linear-independent polynomials to expand the seed. Leveraging this observation, we pick univariate polynomials $\{P_i(X)\}_{i \in [n]} \subset \mathbb{F}[X]$ to expand a *constant* seed $\beta \in \mathbb{F}$ into a n-size vector. We have two natural candidate families for polynomials P_i:

- The monomial basis $P_i(X) = X^{i-1}$. In this case, the vector of randomizers becomes the powers of β : $\boldsymbol{b} = (1, \beta, \beta^2, \ldots, \beta^{n-1})$
- The Lagrange basis over a domain $\mathbb{D}$ of dimension n : $\{\lambda_i(X)\}_{i \in [n]}$ evaluated on β, yielding $\boldsymbol{b} = (\lambda_1(\beta), \ldots, \lambda_n(\beta))$.[6]

[6] To prevent confusion, we denote the Lagrange polynomials over the domain $\mathbb{H}$ (of size k) by L_i, and those over $\mathbb{D}$ (of size n) by λ_i.

Both of these families are linearly independent over $\mathbb{F}[X]$ and yield valid randomizers. The drawback with respect to the initial tensor product structure is that the accumulator relation becomes harder to fold. The advantages are that it removes one round of interaction (with respect to vanilla Protogalaxy), and especially, that it enables very efficient deciders.

3.2 Relations and Polynomials

Next, we formally describe the relations that will be used throughout this work. We let $\mathbb{F}$ be a finite field, and $f := (f_1, \ldots, f_n) : \mathbb{F}^M \to \mathbb{F}^n$ be a degree-d polynomial map. We recall that whenever we write $\mathbf{t} = \mathsf{cm}(\mathbf{w})$, we refer to the committed relations introduced in Definition 2 and the hash consistency check $\mathcal{H}(\mathbf{t}) = 1$ is as stated in Definition 3. We refer to the space of committed relations simply by $\mathcal{CR}$.

Definition 5 (Base polynomial relation). *The base relation $\mathcal{R}$ is the subset $\mathcal{R} \subseteq \mathcal{CR} \times \mathbb{F}^M$ defined by*

$$\mathcal{R} := \left\{ (\mathbf{t}, \mathbf{w}) \;\middle|\; \mathbf{t} = \mathsf{cm}(\mathbf{w}) \;\wedge\; f(\mathbf{w}) = \mathbf{0}_n \;\wedge\; \mathcal{H}(\mathbf{t}) = 1 \right\}.$$

Definition 6 (Generalized randomized relation). *The generalized randomized relation $\mathcal{R}_{\mathsf{gen}}$ is the subset $\mathcal{R}_{\mathsf{gen}} \subseteq (\mathcal{CR} \times \mathbb{F}^n \times \mathbb{F}) \times \mathbb{F}^M$ defined by*

$$\mathcal{R}_{\mathsf{gen}} := \left\{ ((\mathbf{t}, \boldsymbol{b}, e), \mathbf{w}) \;\middle|\; \mathbf{t} = \mathsf{cm}(\mathbf{w}) \;\wedge\; \sum_{i=1}^{n} b_i f_i(\mathbf{w}) = e \right\}.$$

As we mentioned above, this is an inefficient relation to work with, as the vector $\mathbf{b}$ is of size n. To address this, we introduce $\mathcal{R}_{\mathsf{P}}$, which is parametrized by a family of polynomials $\{P_i(X)\}_{i\in[n]}$.

Definition 7 (Relation $\mathcal{R}_{\mathsf{P}}$). *The relation $\mathcal{R}_{\mathsf{P}}$ is parameterized by a family of polynomials $\{P_i(X)\}_{i\in[n]}$ and consists of the pairs $((\mathbf{t}, \beta, e), \mathbf{w})$ such that:*

1. $\mathbf{t} = \mathsf{cm}(\mathbf{w})$.
2. $\beta, e \in \mathbb{F}$, *and*

$$\sum_{i\in[n]} P_i(\beta) f_i(\mathbf{w}) = e,$$

In our batch proving construction, Accordion, we will use (univariate) Lagrange polynomials as a basis, i.e. $P_i(X) := \lambda_i(X) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{X - x_j}{x_i - x_j}$.

4 Accordion: Proving Scheme for Batch Relations

In this section, we present Accordion, an efficient proof system for batch relations. Accordion is built in two steps. The first step is a reduction of knowledge from

k instances of $\mathcal{R}$ into a single instance of $\mathcal{R}_{\mathsf{P}}$. The second step is to build an efficient decider for $\mathcal{R}_{\mathsf{P}}$, both for the polynomials P_i being the (univariate) power basis and the Lagrange basis (that is, for $\mathcal{R}_\lambda$). We show how this can be done efficiently for a broad class of SNARKs that support whichever arithmetization is used during folding. Then, we specify a concrete instantiation of our decider using a Plonk/Halo2 proof system, which we implement and benchmark.

The main idea behind Accordion is that folding randomizers only need linear independence, as we saw in the previous section. Thus, we can choose them to match the basis used in a polynomial commitment scheme of our choice, which enables the efficient decider we are looking for.

4.1 Step 1: The One-Shot Folding Scheme

The first step of Accordion is an optimized Protogalaxy-based reduction of knowledge from k instances of the base relation to a single instance of $\mathcal{R}_{\mathsf{P}}$ for be a degree-d polynomial maps f. As before, fix positive integers d, $k < n$. Let $\mathbb{H} \subset \mathbb{F}$ be a domain of size k, and let $L_1(X), \ldots, L_k(X)$ be its Lagrange basis. Let $Z_{\mathbb{H}}(X) := \prod_{a \in \mathbb{H}}(X - a)$ be the vanishing polynomial of $\mathbb{H}$.

Finally, let $\{P_i\}_{i \in [n]}$ be a family of polynomials as in Definition 7. We present the protocol in Figure 1.

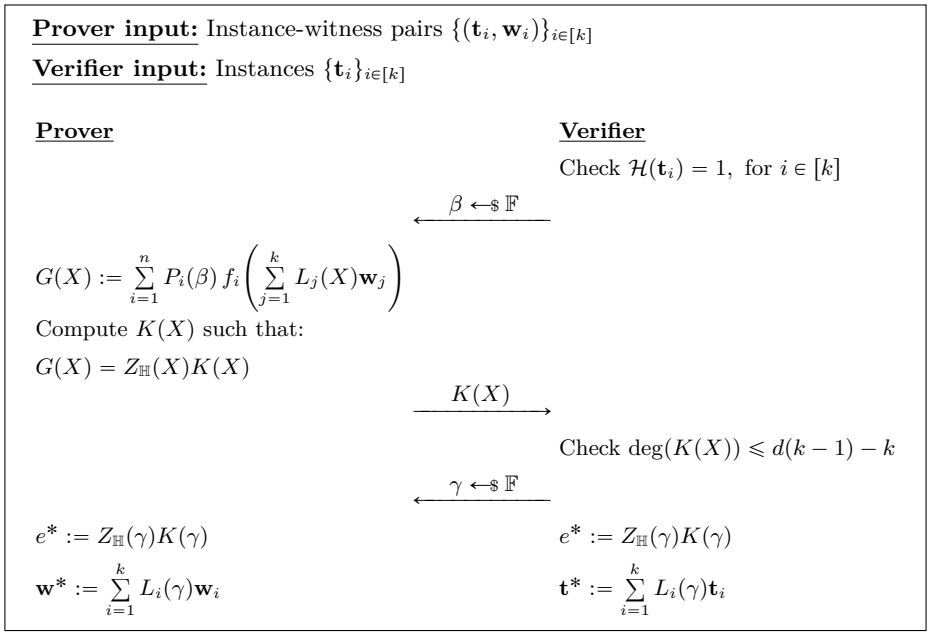

Fig. 1. Accordion reduction of knowledge $\mathcal{R}^k \to \mathcal{R}_{\mathsf{P}}$

Theorem 1. *Let* $\mathsf{cm} : \mathbb{F}^M \to \mathbb{G}$ *be an additively homomorphic binding commitment scheme. Then, the protocol in Figure 1 is a reduction of knowledge* $\mathcal{R}^k \to \mathcal{R}_{\mathsf{P}}$.

Both correctness and security follow similarly as for Protogalaxy. For completeness, we provide a detailed security proof in Section D. We remark that Figure 1 is a one-shot protocol by construction; an output instance of $\mathcal{R}_{\mathsf{P}}$ cannot be refolded as β is freshly sampled at every fold and the k inputs are assumed to have $e_i = 0$.

Prover Efficiency. The prover's efficiency of the RoK in Figure 1 is essentially that of Protogalaxy for k instead of $k+1$ instances, except that the computation of the polynomial F is not carried out here, saving also one random oracle evaluation. Hence, the cost that dominates is the computation of $G(X)$, which requires $dk(M\log(dk) + Cn)$ field operations via [12, Claim 4.5].[7] This holds as long as computing $P_i(\beta)$ $\forall i \in [n]$ can be done in $\mathcal{O}(n)$ time. For power monomials, this holds trivially. For $P_i(X) = \lambda_i(X)$ being the Lagrange basis, we rely on the following fact:

Lemma 1. *Let* $\mathbb{D} \subset \mathbb{F}$ *be a multiplicative subgroup of size* n *and let* $\{\lambda_i(X)\}_{i\in[n]}$ *be the Lagrange polynomials defined on the domain* $\mathbb{D}$*. Then, for any* $\beta \in \mathbb{F}$ *and* $i \in [n]$*, it holds that*

$$\lambda_i(\beta) = \frac{\beta^n - 1}{n} \cdot \frac{\omega^i}{\beta - \omega^i}$$

where ω *is a generator of* $\mathbb{D}$*, that is, a* n*-th root of unity.*

As the factor $(\beta^n - 1)/n$ *is independent of* i *and the other factor can be computed for all* $i \in [n]$ *in time* $\mathcal{O}(n)$ *if the* ω^i*'s are precomputed, calculating* $\lambda_i(\beta)$ *for every* $i \in [n]$ *can be done in* $\mathcal{O}(n)$ *time.*

Verifier Efficiency. The size of the one-shot folding proof is exactly $d(k-1) - k = \mathcal{O}(dk)$ field elements. The verifier needs to check the validity of the instances (which may include a few hash function evaluations, depending on the underlying arithmetization), compute two random oracle evaluations over $\mathbb{F}$, and to evaluate a polynomial of degree $\mathcal{O}(dk)$. Most importantly, the verifier needs to aggregate the commitments $\{\mathbf{t}_i\}_{i\in[k]}$ to obtain $\mathbf{t}^*$, which is the cost that dominates since it involves group operations. All costs are independent from the witness size M and the arithmetization size n. Compared to Protogalaxy, we remove the first round, shaving off a $\mathcal{O}(\log n)$ additive term from the proof size and the verifier time.

[7] Here, C is the number of multiplications needed to compute a single $f_i(\mathbf{w})$ for a given $\mathbf{w} \in \mathbb{F}^M$, and d is the folding degree.

4.2 Step 2: The Efficient Decider

Next, we introduce how to build an efficient SNARK for the output relation $\mathcal{R}_{\mathsf{P}}$ of the previous RoK, or in other words, an efficient decider. For this, we expand the underlying proof system for the base relations that is used to compute the committed transcripts that we fold. In our implementation, we use an optimized Plonk-Halo2 scheme, but the approach is valid for any underlying polynomial IOP-like proof system.

We need to prove knowledge of a (trace) witness $\mathbf{w} \in \mathbb{F}^M$ such that $\mathbf{t} = \mathsf{cm}(\mathbf{w})$ and $\sum_{i\in[n]} P_i(\beta) f_i(\mathbf{w}) = e$. Let $s_i = f_i(\mathbf{w})$ for every $i \in [n]$, and define the extended witness $\bar{\mathbf{w}} = (\mathbf{w}, \boldsymbol{s})$. Also define the extended functions $\bar{f}_i(\bar{\mathbf{w}}) = f_i(\mathbf{w}) - s_i$. Clearly, $\bar{f}_i(\bar{\mathbf{w}}) = 0$ if and only if $f_i(\mathbf{w}) = s_i$. Then, the proof boils down to proving that:

1. $\bar{f}_i(\bar{\mathbf{w}}) = 0$ for every $i \in [n]$, i.e., the accumulator $\bar{\mathbf{w}}$ is a *valid witness* for the original relation $\mathcal{R}$ when applied on the extended function $\bar{f}$, and
2. $\sum_{i=1}^{n} s_i P_i(\beta) = e$.

To prove (1), one can use the proof system used to generate the traces with an additional input, that is $\boldsymbol{s}$, and the additional constraints to define the extended functions $\bar{f}_i$. Naturally, one needs to add a commitment cm_s to $\boldsymbol{s}$ as an additional input to the verifier. Proving (2) is more challenging as the vector $\boldsymbol{b} = (P_1(\beta), \ldots, P_n(\beta))$ cannot be added as a constraint into $\bar{f}$ easily. Indeed, if $\boldsymbol{s}$ is committed, one needs to prove that the relation $\sum_{i=1}^{n} s_i P_i(\beta) = e$ holds given that the verifier knows β and e. This is essentially an inner product argument (IPA) relation $\mathcal{R}_{\mathsf{ipa}}$ between a vector $\boldsymbol{b}$ of size n that is known to the verifier and a vector of committed values $\boldsymbol{s}$:

$$\mathcal{R}_{\mathsf{ipa}} = \{(c_s, \boldsymbol{b}, e; \boldsymbol{s}) : c_s = \mathsf{cm}(\boldsymbol{s}) \ \wedge \ \langle \boldsymbol{s}, \boldsymbol{b} \rangle = e\}$$

In general, this requires the verifier to run in linear time, and also to compute $\boldsymbol{b}$ beforehand. *Importantly, the commitment c_s in (1) needs to be the same as the one used in (2).*

To avoid linear-time verification in step (2), we rely on additional structure of the P_i's for power monomials and for the Lagrange basis.

Power basis: $\boldsymbol{P_i(X) = X^{i-1}}$. Note that $b_i = P_i(\beta) = \beta^{i-1}$, and so $\mathcal{R}_{\mathsf{ipa}}$ can be expressed as follows:

$$\mathcal{R}_{\mathsf{ipa}} = \left\{ (c_s, \beta, e; \boldsymbol{s}) : c_s = \mathsf{cm}(\boldsymbol{s}) \ \wedge \ \sum_{i\in[n]} s_i \beta^{i-1} = e \right\}$$

If c_s is computed as a polynomial commitment to a polynomial $S(X)$ such that $S(X) = \sum_{i\in[n]} s_i X^{i-1}$, such as with KZG [18], the prover simply needs to prove that opening S on β yields e. We note that an explicit degree check on $S(X)$ is necessary for soundness. Naturally, the polynomial commitment must be

compatible with the proof system for step (1). This is not the case for Plonk, since polynomials are committed in the evaluation domain. Hence, we also introduce an efficient decider in the Lagrange basis.

Lagrange basis: $\boldsymbol{P_i(X) = \lambda_i(X)}$. For building a decider of relation $\mathcal{R}_\lambda$, we have that

$$\mathcal{R}_{\mathsf{ipa}} = \left\{ (c_s, \beta, e; \boldsymbol{s}) : c_s = \mathsf{cm}(\boldsymbol{s}) \ \wedge \ \sum_{i \in [n]} s_i \lambda_i(\beta) = e \right\}$$

If $\boldsymbol{s}$ is committed using a polynomial commitment in the Lagrange basis (as in KZG for Plonk) such that $S(X) = \sum_{i \in [n]} s_i \lambda_i(X)$, the prover simply needs to open $S(\beta) = e$. With this observation, the efficiency of the decider (prover time, verifier time, and proof size) is exactly that of the underlying proof system plus one polynomial commitment evaluation.

4.3 An Optimized Plonk/Halo2 Instantiation

We instantiate Accordion over an optimized Plonk/Halo2-family proof system supporting high-degree custom gates, a multi-set permutation argument, and a LogUp lookup argument [30]. We follow the notation from the original Plonk [14]. In this section, we describe how the folding process and the final decider look like in the context of Plonk. This is a simplified view, and we refer to the reference implementation [3] for the missing technical details.

Traces and the decision function. A trace is the transcript of the Plonk polynomial-IOP up to, but excluding, the quotient round. Concretely, $\mathbf{w} \in \mathbb{F}^M$ consists of one length-n block per polynomial (in evaluation form),

$$\big(\underbrace{a_1, \ldots, a_{n_a}}_{\text{advice}}, \ \underbrace{p_1, \ldots, p_{n_p}}_{\text{public}}, \ \underbrace{z_1, \ldots, z_{n_z}}_{\text{permutation}}, \ \underbrace{m, h_1, \ldots, h_{n_h}, z_{\mathsf{lu}}}_{\text{LogUp}} \big),$$

followed by the verifier challenges of those rounds, $(\boldsymbol{\theta}, \beta_{\mathsf{plonk}}, \gamma_{\mathsf{plonk}}, \tau, \boldsymbol{y})$, where $\boldsymbol{\theta}$ compresses lookup columns, $(\beta_{\mathsf{plonk}}, \gamma_{\mathsf{plonk}})$ are the permutation challenges, and $\boldsymbol{y}$ aggregates constraints. In particular, the fixed columns and selectors are *not* part of the trace, since all k folded instances share the same circuit and the same preprocessing key. The instance $\mathbf{t}$ is the corresponding vector of commitments together with the challenges in the clear.

Towards defining the arithmetization-dependent decision function f, let $\mathcal{C}_1$, $\ldots, \mathcal{C}_C$ be the constraint polynomials of the arithmetization (gates, permutation, LogUp), and let

$$\nu_{\mathbf{w}}(X) \ = \ \sum_{c=1}^{C} y_c \cdot \mathcal{C}_c(\mathbf{w})(X)$$

be the aggregated *numerator*, i.e. the quantity that a Plonk prover divides by $Z_{\mathbb{H}}$ to obtain the quotient. We define the decision function row-wise,

$$f_j(\mathbf{w}) \;:=\; \nu_{\mathbf{w}}(\omega^j), \qquad j \in [n].$$

Then $\mathbf{w}$ satisfies the Plonk relation iff $f_j(\mathbf{w}) = 0$ for all $j \in [n]$, which is equivalent to checking the statement $Z_{\mathbb{H}} \mid \nu_{\mathbf{w}}$. Note that the quotient itself is *not* part of the trace and is never folded. Folding therefore only requires evaluating ν over the domain $\mathbb{H}$.

Folding degree. The f_j must be low-degree d polynomials in all trace entries, including the challenges, since the challenges are also folded. The folding degree of Figure 1 is $d = \max_c \deg_{\mathbf{w}}\big(y_c\,\mathcal{C}_c\big)$, where fixed columns count as degree 0 and each advice/public/derived column and each challenge counts as degree 1. In particular, d may become larger than the arithmetization degree D of the constraint system due to challenges being considered as variables.

To reduce the folding degree, we apply an optimization to vanilla Plonk. We compute all Plonk challenges from separate random oracle calls instead of as powers of a single challenge.[8] This costs $C + |\boldsymbol{\theta}|$ extra random-oracle queries and does not increase the proof size. Besides, we apply a further optimization available in `midnight-zk` to express the quotient polynomial using a degree $D-1$ polynomial.

As an example, with these optimizations the Plonkish arithmetization of degree $D = 5$ of SHA-256 that we use for benchmarking yields the same folding degree $d = 5$. Accordion can be run without these optimizations, but the folding degree increases to $d = 7$ or $d = 8$, depending on the exact Plonk implementation, increasing the prover cost.

We remark that since $\deg G = d(k-1)$, the domain used to compute and interpolate G must have size greater than $d(k-1)$. Since, for efficiency, we take these domains to be FFT-friendly (and hence of power-of-two size), the performance of the scheme is best when $d(k-1)$ is close to a power of two.

The decider. Applying Section 4.2 for $\mathcal{R}_\lambda$, i.e., with $P_j = L_j$, the accumulator is $(\mathbf{t}^*, \beta, e^*; \mathbf{w}^*)$ with $\sum_j \lambda_j(\beta) f_j(\mathbf{w}^*) = e^*$. T he prover sets $s_j := f_j(\mathbf{w}^*) = \nu_{\mathbf{w}*}(\omega^j)$ and commits to the *slack polynomial*

$$S(X) \;=\; \sum_{j\in[n]} s_j \lambda_j(X),$$

i.e. to the vector $\boldsymbol{s}$ in the Lagrange basis. *Crucially, this is the same basis in which the Plonk prover already commits to its witness columns.* The two proofs described in Section 4.2 become:

[8] Vanilla Plonk batches constraints as $\sum_c \alpha^c \mathcal{C}_c$ and compresses lookup columns with powers θ^i. Viewed as a polynomial in the trace entry α, the former has degree C.

	Folding	Decider
Prover time	$\tilde{\mathcal{O}}(dkn)$ $\mathbb{F}$-ops + $\mathcal{O}(k\ell)$ $\mathbb{G}$-ops	$\tilde{\mathcal{O}}(dn)$ $\mathbb{F}$-ops + $\mathcal{O}(1)$ $\mathbb{G}$-ops
Verifier time	$\mathcal{O}(dk)$ $\mathbb{F}$-ops + k $\mathbb{G}$-ops	ℓ $\mathbb{G}$-ops + 2 pairings
Proof size	dk $\lvert\mathbb{F}\rvert$	$\mathcal{O}(1) \cdot \lvert\mathbb{G}\rvert$

Table 1. Asymptotic cost of end-to-end Accordion for a batch of k instances of a degree-d relation $f : \mathbb{F}^M \to \mathbb{F}^n$. $\tilde{\mathcal{O}}$ hides logarithmic factors in n and/or in k. ℓ is the number of columns in the Plonk arithmetization. The reported proof size does not include the commitments to the witnesses $\mathcal{O}(k\ell)$ $\mathbb{G}$-elements.

1. $\bar{f}_j(\bar{\mathbf{w}}) = 0$ for all j: the prover runs the Plonk prover on $\mathbf{w}^*$ with numerator $\nu_{\mathbf{w}^*} - S$, i.e. it commits to $h = (\nu_{\mathbf{w}^*} - S)/Z_{\mathbb{H}}$. This is a polynomial iff $s_j = f_j(\mathbf{w}^*)$ for every j.
2. $\langle \boldsymbol{s}, \boldsymbol{b} \rangle = e^*$: this is a simple KZG opening proof $S(\beta) = e^*$.

Both openings of S (at x and at β) are batched into the multi-opening argument already present in Plonk. Overall, the overhead of the decider over a plain Plonk proof is one commitment (to $S(X)$) and one extra opening point.

In Table 1, we describe the asymptotic performance of the instantiation of Accordion that we just introduced.

5 Implementation and Performance Evaluation

We implemented Accordion in Rust, starting from a fork of the main branch of `midnight-zk` done in June 2026. We compare it with the production code from the `midnight-zk` library, a higly optimized library building on Halo2 proof system from the Privacy Scaling Explorations (PSE) team, itself originally derived from the Zcash Sapling proving system. Our code is a proof of concept and has not targeted all possible optimisations, but it inherits some efficient building blocks from its dependencies on `midnight-zk` library. We note that the `midnight-zk` library is actively being maintained and new versions of IVC are being developed, so the benchmarked IVC performance may improve significantly in the near future.

In Figure 2, we benchmark end-to-end Accordion for batch proving k different statements for the relation *"I know x such that $y = \mathsf{SHA256}(x)$"*, for k ranging from 2 to 64. We also provide the concrete numbers in Table 2. The arithmetisation of the circuit is done in Plonkish constraints, includes lookup custom gates, and has size 2^{12}. In Figure 3, we show benchmarks for a significantly more complex circuit: a batch verification of 15 ECDSA signatures, with 2^{17} gates (meaning $15 \cdot k$ verified signatures in total). We compare Accordion against two other approaches:

– *Separate ($k\times$Plonk)*: generate and verify k independent Plonk-family proofs separately, with parallelization, using the production code from the `midnight-zk` library (no batching for the verifier, no IVC).

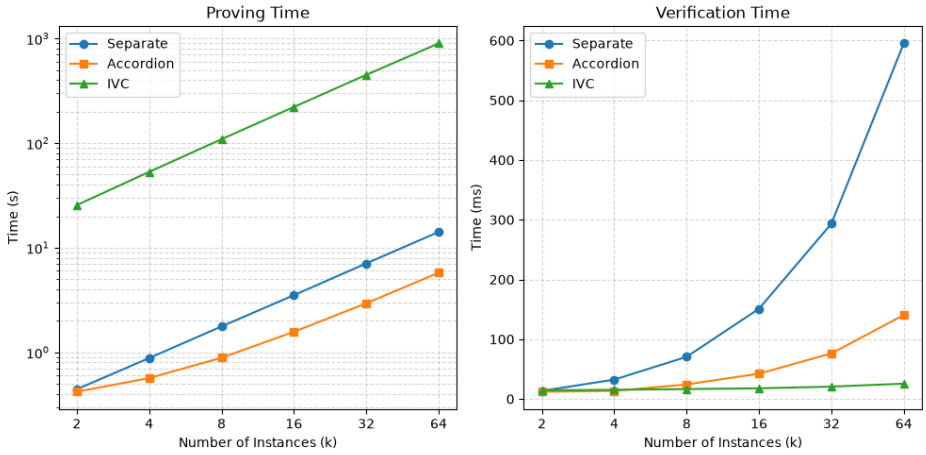

Fig. 2. Benchmarks for k instances for a circuit of 2^{12} gates proving SHA-256 preimages.

k	Proving Time (s)			Verification Time (ms)			Proof Size (kB)		
	Separate	**IVC**	**Accordion**	**Separate**	**IVC**	**Accordion**	**Separate**	**IVC**	**Accordion**
2	0.4	25.5	0.4	13.5	14.8	12.0	6.82	5.05	4.4
4	0.9	53.1	0.6	32.4	15.3	13.5	13.6	5.05	6.0
8	1.8	109	0.9	70.5	16.5	24.3	27.3	5.05	9.3
16	3.5	222	1.6	151	17.8	42.3	54.5	5.05	15.8
32	7.1	450	2.9	293	20.6	76.2	109	5.05	28.9
64	14.2	899	5.8	595	25.6	140	218	5.05	55.0

Table 2. Performance comparison for a batch of k instances of a SHA-256 preimage

– *IVC*: Recurses k times over the k instances using the existing IVC pipeline in `midnight-zk`. This is a self-recursive SNARK scheme designed to support Plonkish relations and custom gates.

Hardware. All benchmarks were run on Intel® Core™ i9-14900HX (32 threads) with 64GB RAM, with no restriction on the number of cores used. Parallelization was used in all benchmarks, with no cores being idle.

Proving time. Both figures show that Accordion trails k-Plonk closely for small k, but for large k's, generating Plonk proofs becomes significantly more expensive. Concretely, for $k = 64$ Accordion is about $2.5\times$ faster than k-Plonk for the SHA-256 circuit. This is expected, as the folding step removes a significant amount of computation otherwise done by the Plonk prover. The IVC approach is 1 to 2 orders of magnitude more expensive: the per-step in-circuit verification contributes significantly to the total proving cost.

Verification time. In comparison to the verification of k separate Plonk proofs (without using batching techniques), Accordion is up to 3× faster. The Accordion verifier is linear in k, the cost that dominates being the cost of aggregating k commitments. Naturally, the IVC approach scales better as it does not check any accumulator until the very end.

Trade-offs and total costs. Accordion offers a great trade-off between IVC and separate proving: the prover efficiency is best out of the three approaches, and while verification time and proof size are larger than for IVC-based aggregation, the costs are in any case very small. Notably, Accordion can verify $15 \cdot 64 = 960$ ECDSA signatures in only 200 ms. To have a full pipeline comparison between Accordion and IVC, we can consider the total time of proving and verifying: For $k = 32$ SHA-256 instances, IVC requires 450 s, while Accordion runs in only **2.98 s**.

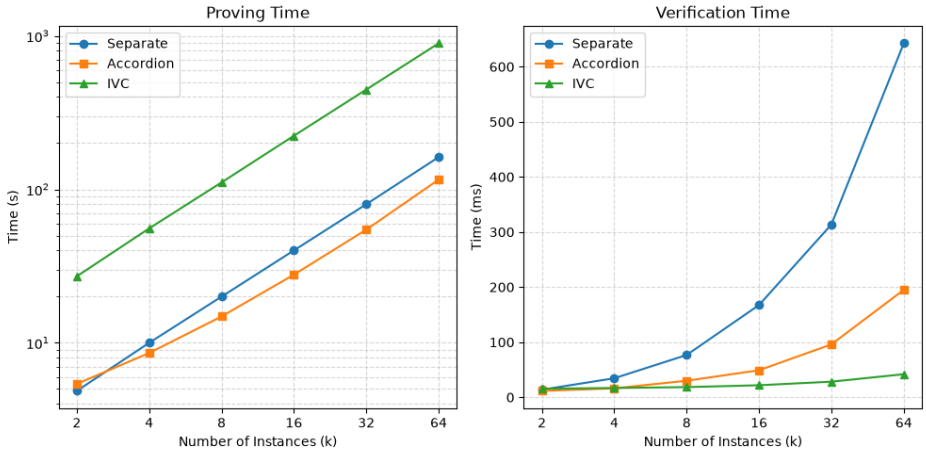

Fig. 3. Benchmarks over a 15×ECDSA verification circuit with 2^{17} gates.

6 Conclusions

We built Accordion, a proof system for batch relations that combines the efficiency of folding schemes with the practicality of off-the-shelf SNARKs, achieving competitive proving time, proof size and verification time. We designed Accordion having Blockchain applications in mind, but we left many other application venues unexplored. Our folding step aggregates computations that can be expressed as a polynomial map $f : \mathbb{F}^M \to \mathbb{F}^n$ of some degree d with a verifier complexity independent of n. This feature can be possibly exploited for optimal proving of parallel executions of multiple instructions on the same data, by crafting the target computation as polynomial maps with large n.

Acknowledgements

We are incredibly thankful to Íñigo Querejeta Azurmendi for his work on the implementation and evaluation of our scheme. We also want to thank Raphael Toledo for the effort of running benchmarks and the help with understanding the Midnight-zk rust library, and Miguel Ambrona for initial discussions.

The research of David Balbás is supported by Zurich Information Security and Privacy Center (ZISC). Part of this work was done while David was at Input Output.

References

[1] Ambrona, M., Beunardeau, M., Schmitt, A.L., Toledo, R.R.: aplonk: Aggregated plonk from multi-polynomial commitment schemes. p. 195–213. Springer-Verlag, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-3-031-41326-1_11, https://doi.org/10.1007/978-3-031-41326-1_11

[2] Attema, T., Fehr, S., Klooß, M.: Fiat-shamir transformation of multi-round interactive proofs. In: TCC 2022, Part I. pp. 113–142. LNCS (Nov 2022). https://doi.org/10.1007/978-3-031-22318-1_5

[3] Balbás, D., Nitulescu, A., Plançon, M.: Implementation of accordion over midnight-zk. https://github.com/davidbalbas/midnight-zk/tree/accordion (2026)

[4] Balbás, D., Nitulescu, A., Plançon, M.: ProtogaLattice: Constant-round lattice-based folding for general polynomial relations. Cryptology ePrint Archive, Paper 2026/1317 (2026), https://eprint.iacr.org/2026/1317

[5] Boneh, D., Chen, B.: LatticeFold: A lattice-based folding scheme and its applications to succinct proof systems. In: ASIACRYPT 2025, Part III. pp. 330–362. LNCS (Dec 7–11, 2025). https://doi.org/10.1007/978-981-95-5099-9_11

[6] Boneh, D., Chen, B.: LatticeFold+: Faster, simpler, shorter lattice-based folding for succinct proof systems. pp. 327–361. LNCS, Santa Barbara, CA, USA (Aug 2025). https://doi.org/10.1007/978-3-032-01907-3_11

[7] Bowe, S., Grigg, J., Hopwood, D.: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Paper 2019/1021 (2019), https://eprint.iacr.org/2019/1021

[8] Bünz, B., Chen, B.: Protostar: Generic efficient accumulation/folding for special-sound protocols. In: ASIACRYPT 2023, Part II. pp. 77–110. LNCS (Dec 7–11, 2023). https://doi.org/10.1007/978-981-99-8724-5_3

[9] Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Recursive proof composition from accumulation schemes. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part II. LNCS, vol. 12551, pp. 1–18. Durham, NC, USA (Nov 16–19, 2020). https://doi.org/10.1007/978-3-030-64378-2_1

[10] Chen, B.: Symphony: Scalable SNARKs in the random oracle model from lattice-based high-arity folding. Cryptology ePrint Archive, Paper 2025/1905 (2025), https://eprint.iacr.org/2025/1905

[11] Chiesa, A., Tromer, E.: Proof-carrying data and hearsay arguments from signature cards. In: Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings. pp. 310–331 (2010)

[12] Eagen, L., Gabizon, A.: ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances. Cryptology ePrint Archive, Report 2023/1106 (2023), https://eprint.iacr.org/2023/1106

[13] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Santa Barbara, CA, USA (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12

[14] Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019), https://eprint.iacr.org/2019/953

[15] Gailly, N., Maller, M., Nitulescu, A.: SnarkPack: Practical SNARK aggregation. pp. 203–229. LNCS (2022). https://doi.org/10.1007/978-3-031-18283-9_10
[16] Garreta, A., Lipmaa, H., Luhaäär, U., Osadnik, M.: Cyclo: Lightweight lattice-based folding via partial range checks. In: Advances in Cryptology – EUROCRYPT 2026. pp. 33–62. Springer Nature Switzerland, Cham (2026)
[17] Garreta, A., Manzur, I.: FLI: Folding lookup instances. pp. 402–435. LNCS (Dec 7–11, 2024). https://doi.org/10.1007/978-981-96-0935-2_13
[18] Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Singapore (Dec 5–9, 2010). https://doi.org/10.1007/978-3-642-17373-8_11
[19] Kothapalli, A., Parno, B.: Algebraic reductions of knowledge. In: CRYPTO 2023, Part IV. pp. 669–701. LNCS, Santa Barbara, CA, USA (Aug 2023). https://doi.org/10.1007/978-3-031-38551-3_21
[20] Kothapalli, A., Setty, S.: NeutronNova: Folding everything that reduces to zero-check. Cryptology ePrint Archive, Report 2024/1606 (2024), https://eprint.iacr.org/2024/1606
[21] Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive zero-knowledge arguments from folding schemes. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 359–388. Santa Barbara, CA, USA (Aug 15–18, 2022). https://doi.org/10.1007/978-3-031-15985-5_13
[22] Kothapalli, A., Setty, S.T.V.: HyperNova: Recursive arguments for customizable constraint systems. pp. 345–379. LNCS, Santa Barbara, CA, USA (Aug 2024). https://doi.org/10.1007/978-3-031-68403-6_11
[23] Liu, X., Gao, S., Zheng, T., Xiao, B.: SnarkFold: Efficient SNARK proof aggregation from split incrementally verifiable computation. Cryptology ePrint Archive, Report 2023/1946 (2023), https://eprint.iacr.org/2023/1946
[24] Midnight: midnight-zk. https://github.com/midnightntwrk/midnight-zk (2026)
[25] Nguyen, W., Setty, S.: Neo and superneo: Post-quantum folding with pay-per-bit costs over small fields. p. 492–523. Springer-Verlag, Berlin, Heidelberg (2026). https://doi.org/10.1007/978-3-032-35424-2_17, https://doi.org/10.1007/978-3-032-35424-2_17
[26] Nguyen, W.D., Datta, T., Chen, B., Tyagi, N., Boneh, D.: Mangrove: A scalable framework for folding-based SNARKs. pp. 308–344. LNCS, Santa Barbara, CA, USA (Aug 2024). https://doi.org/10.1007/978-3-031-68403-6_10
[27] Nitulescu, A., Paslis, N., Ràfols, C.: FLIP-and-prove r1CS. IACR Communications in Cryptology **2**(4) (2026). https://doi.org/10.62056/ayc3tx4e-
[28] Rosenberg, M., Maller, M., Miers, I.: Snarkblock: Federated anonymous blocklisting from hidden common input aggregate proofs. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 948–965 (2022). https://doi.org/10.1109/SP46214.2022.9833656
[29] Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 704–737. Santa Barbara, CA, USA (Aug 17–21, 2020). https://doi.org/10.1007/978-3-030-56877-1_25
[30] Soukhanov, L.: Logup*: faster, cheaper logup argument for small-table indexed lookups. Cryptology ePrint Archive, Paper 2025/946 (2025), https://eprint.iacr.org/2025/946
[31] Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp.

1–18. San Francisco, CA, USA (Mar 19–21, 2008). https://doi.org/10.1007/978-3-540-78524-8_1
[32] zcash: halo2. https://zcash.github.io/halo2/ (2022)

A Additional Preliminaries

A.1 Commitments

Definition 8. *A commitment scheme is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{VerCom})$ *that work as follows:*

- $\mathsf{Setup}\left(1^\lambda\right) \to \mathsf{ck}$ *takes as input the security parameter* λ *and returns the commitment key* ck*, and descriptions of the input space* $\mathcal{D}$*, commitment space* $\mathcal{C}$ *and opening space* $\mathcal{O}$*.*
- $\mathsf{Com}(\mathsf{ck}, v) \to (c, o)$ *takes the commitment key* ck *and a value* $v \in \mathcal{D}$*, and outputs a commitment* c *and an opening* o*.*
- $\mathsf{VerCom}(\mathsf{ck}, c, v, o) \to b$ *takes as input a commitment* c*, a value* v *and an opening* o*, and accepts* $(b = 1)$ *or rejects* $(b = 0)$*.*

A commitment scheme should satisfy the correctness and binding properties, and often they also satisfies a hiding property.
Correctness. For all $\lambda \in \mathbb{N}$ and any input $v \in \mathcal{D}$ we have:

$$\mathsf{Pr}\left[\mathsf{ck} \leftarrow \mathsf{Setup}\left(1^\lambda\right), (c, o) \leftarrow \mathsf{Com}(\mathsf{ck}, v) : \mathsf{VerCom}(\mathsf{ck}, c, v, o) = 1\right] = 1$$

Binding. For every polynomial-time adversary $\mathcal{A}$, the following probability is negligible in the security parameter

$$\mathsf{Pr}\left[\begin{array}{rl} & v \neq v' \ \wedge \ \mathsf{VerCom}(\mathsf{ck}, c, v, o) = 1 \\ \mathsf{ck} \leftarrow \mathsf{Setup}\left(1^\lambda\right), & (c, v, o, v', o') \leftarrow \mathcal{A}(\mathsf{ck}) \end{array} ; \mathsf{VerCom}\left(\mathsf{ck}, c, v', o'\right) = 1\right].$$

Hiding. For $\mathsf{ck} \leftarrow \mathsf{Setup}\left(1^\lambda\right)$ and $\forall v, v' \in \mathcal{D}$, the following two distributions are statistically close: $\mathsf{Com}(\mathsf{ck}, v) \approx \mathsf{Com}\left(\mathsf{ck}, v'\right)$.

Remark 1. For simplicity, we denote by $\mathsf{Com}(\mathbf{v}) = c$ a (computationally) binding commitment to a vector of values $\mathbf{v}$. All commitment schemes we use in this work are additively homomorphic.

A.2 Polynomial Commitments.

Polynomial commitments (PC) first introduced by [18] are commitments for the message space $\mathbb{F}^{\leq d}[X]$, the ring of polynomials in X with maximum degree $d \in \mathbb{N}$ and coefficients in the field $\mathbb{F} = \mathbb{Z}_p$, that support an interactive argument of knowledge $(\mathsf{KG}, \mathsf{Open}, \mathsf{Check})$ for proving the correct evaluation of a committed polynomial at a given point without revealing any other information about the committed polynomial.

A polynomial commitment scheme over a field family $\mathcal{F}$ consists in 4 algorithms $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Check})$ defined as follows:

$\mathsf{Setup}(1^\lambda, d) \to (\mathsf{ck}, \mathsf{vk})$**:** given a security parameter λ fixing a field $\mathcal{F}_\lambda$ family and a maximal degree d samples a group description gk containing a description of a field $\mathbb{F} \in \mathcal{F}_\lambda$, and commitment and verification keys $(\mathsf{ck}, \mathsf{vk})$. We implicitly assume ck and vk each contain gk.

$\mathsf{Com}(\mathsf{ck}; f(X)) \to C$**:** given ck and a polynomial $f(X) \in \mathbb{F}^{\leqslant d}[X]$ outputs a commitment C.

$\mathsf{Open}(\mathsf{ck}; C, x, y; f(X)) \to \pi$**:** given a commitment C, an evaluation point x, a value y and the polynomial $f(X) \in \mathbb{F}[X]$, it output a prove π for the relation:

$$\mathcal{R}_{\mathsf{kzg}} := \left\{ (\mathsf{ck}, C, x, y; f(X)) : \begin{array}{r} C = \mathsf{Com}\,(\mathsf{ck}; f(X)) \\ \wedge\ \deg(f(X)) \leqslant d \\ \wedge\ y = f(x) \end{array} \right\}$$

$\mathsf{Check}(\mathsf{vk}, C, x, y, \pi) \to 1/0$: Outputs 1 if the proof π verifies and 0 if π is not a valid proof for the opening (C, x, y).

A polynomial commitment satisfies evaluation binding:

Definition 9 (Computational Knowledge Binding). *For every* PPT *adversary* $\mathcal{A}$ *that produces a valid proof* π *for statement* C, x, y*, i.e. such that* $\mathsf{Check}(\mathsf{vk}, C, x, y, \pi) = 1$*, there is a* PPT *extractor* $\mathsf{Ext}_{\mathcal{A}}$ *that is able to output a pre-image polynomial* $f(X)$ *with overwhelming probability:*

$$\Pr\left[\begin{array}{l} \mathsf{Check}(\mathsf{vk}, C, x, y, \pi) = 1 \\ \wedge\ C = \mathsf{Com}(\mathsf{ck}; f(X)) \end{array} \middle| \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{KG}(1^\lambda, d) \\ (C, x, y, \pi; f(X)) \leftarrow (\mathcal{A} \| \mathsf{Ext}_{\mathcal{A}})(\mathsf{ck}) \end{array} \right] = 1 - \mathsf{negl}(\lambda).$$

A.3 KZG Polynomial Commitment

We describe the KZG Polynomial Commitment from [18] which allows to check correctness of evaluation openings.
We recall the scheme $\mathsf{KZG.PC} = (\mathsf{KZG.Setup}, \mathsf{KZG.Com}, \mathsf{KZG.Open}, \mathsf{KZG.Check})$ defined over bilinear groups $\mathsf{gk} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ with $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle h \rangle$:

$\mathsf{KZG.KG}(1^\lambda, n) \to (\mathsf{ck}, \mathsf{vk}_h)$**:** Set keys $\mathsf{ck}_g = \{g^{\alpha^i}\}_{i=0}^{n-1}, \mathsf{vk}_h = h^\alpha$.

$\mathsf{KZG.Com}(\mathsf{ck}_g; f(X)) \to C_f$**:** For $f(X) = \sum_{i=0}^{n-1} f_i X^i$, computes

$$C_f = \prod_{i=0}^{n-1} g^{f_i \alpha^i} = g^{f(\alpha)}.$$

$\mathsf{KZG.Open}(\mathsf{ck}_g; C_f, x, y; f(X)) \to \pi$**:** For an evaluation point x, a value y, compute the quotient polynomial

$$q(X) = \frac{f(X) - y}{X - x}$$

and output proof

$$\pi := C_q = \mathsf{KZG.Com}(\mathsf{ck}_g; q(X)).$$

$\mathsf{KZG.Check}(\mathsf{vk}_h = h^\alpha, C_f, x, y, \pi) \to 1/0$**:** Check if

$$e(C_f \cdot g^{-y}, h) = e(C_q, \mathsf{vk}_h \cdot h^{-x}).$$

A.4 Special Sound Protocols

Definition 10 (Tree of transcripts). *Let $k \in \mathbb{N}$ and arities $(\alpha_1, \ldots, \alpha_\mu) \in \mathbb{N}$. An $\boldsymbol{\alpha}$-tree of transcripts for a $(2\mu+1)$-move protocol is a set of $\alpha_1 \cdot \ldots \cdot \alpha_\mu$ accepting transcripts arranged in a μ-depth tree of arity $\alpha_1, \ldots, \alpha_\mu$. Every transcript corresponds to one path from the root to a leaf node.*

Definition 11 (Special Soundness). *Let Π be a $(2\mu+1)$-move public-coin interactive proof Π for relation $\mathcal{R}$, where the verifier samples its challenges from a set of size N. We say that Π is $(\alpha_1, \ldots, \alpha_\mu)$-out-of-$N$ special sound if there exists a PPT extractor $\mathcal{E}$ such that, given any $(\alpha_1, \ldots, \alpha_\mu)$-tree of accepting transcripts for a statement $\boldsymbol{x}$, $\mathcal{E}$ outputs a witness $\boldsymbol{w}$ such that $(\boldsymbol{x}, \boldsymbol{w}) \in \mathcal{R}$ with overwhelming probability.*

Random-Oracle Non-Interactive Argument of Knowledge. We say that a $(2\mu+1)$-move interactive protocol is *special-sound*, if it is α^μ-out-of-N special-sound, for $\alpha = \mathsf{poly}(\lambda)$ and $N = \lambda^{\omega(1)}$. The results of [2] imply that the non-interactive version of the protocol, after applying Fiat-Shamir transformation [13], is knowledge-sound with error $\mathsf{negl}(\lambda)$ in the random oracle model.

Special Soundness for Committed Transcripts. We remark that if an interactive proof is special-sound, the committed version of the protocol (i.e. such that all prover messages are committed) is computationally special sound. The transformation holds by extending the definition of a valid extractor: for a committed special-sound protocol, given a valid tree of transcripts, an extractor should be able to extract *either a witness of the relation or a collision for the commitment scheme* with overwhelming probability, breaking commitment binding in the latter case. We refer to [12] for details and a formal exposition.

B Description of Protogalaxy

B.1 Protogalaxy Folding

We first introduce the randomized relation $\mathcal{R}_{\mathsf{rand}}$ for the accumulators of the Protogalaxy scheme.

Definition 12 (Relation $\mathcal{R}_{\mathsf{rand}}$). *The relation $\mathcal{R}_{\mathsf{rand}}$ consists of the pairs $((\mathbf{t}, \boldsymbol{\beta}, e), \mathbf{w})$ such that:*

1. $\mathbf{t} = \mathsf{cm}(\mathbf{w})$.
2. $\boldsymbol{\beta} \in \mathbb{F}^t, e \in \mathbb{F}$, *and*
$$\sum_{i=1}^{n} \mathsf{pow}_i(\beta_1, \ldots, \beta_t) f_i(\mathbf{w}) = e,$$
where the vector $\mathsf{pow}(X_1, \ldots, X_t) = (1, X_t) \otimes \cdots \otimes (1, X_1) \in \mathbb{F}[X_1, \ldots, X_t]^{2^t}$, for $t = \lceil \log n \rceil$ and $\mathsf{pow}_i(X_1, \ldots, X_t)$ is its i-th coordinate.

In Figure 4, we describe the Protogalaxy folding scheme.

The protocol is a reduction of knowledge $\mathcal{R}_{\mathsf{rand}} \times \mathcal{R}^k \to \mathcal{R}_{\mathsf{rand}}$. We fix positive integers d, $k < n$, and denote $t := \log n$. We also let $L_0(X), L_1(X), \ldots, L_k(X)$ be the Lagrange basis over a domain $\mathbb{H} \subset \mathbb{F}$ of size $k+1$, and $Z_{\mathbb{H}}(X) := \prod_{a \in \mathbb{H}}(X - a)$ be the corresponding vanishing polynomial.

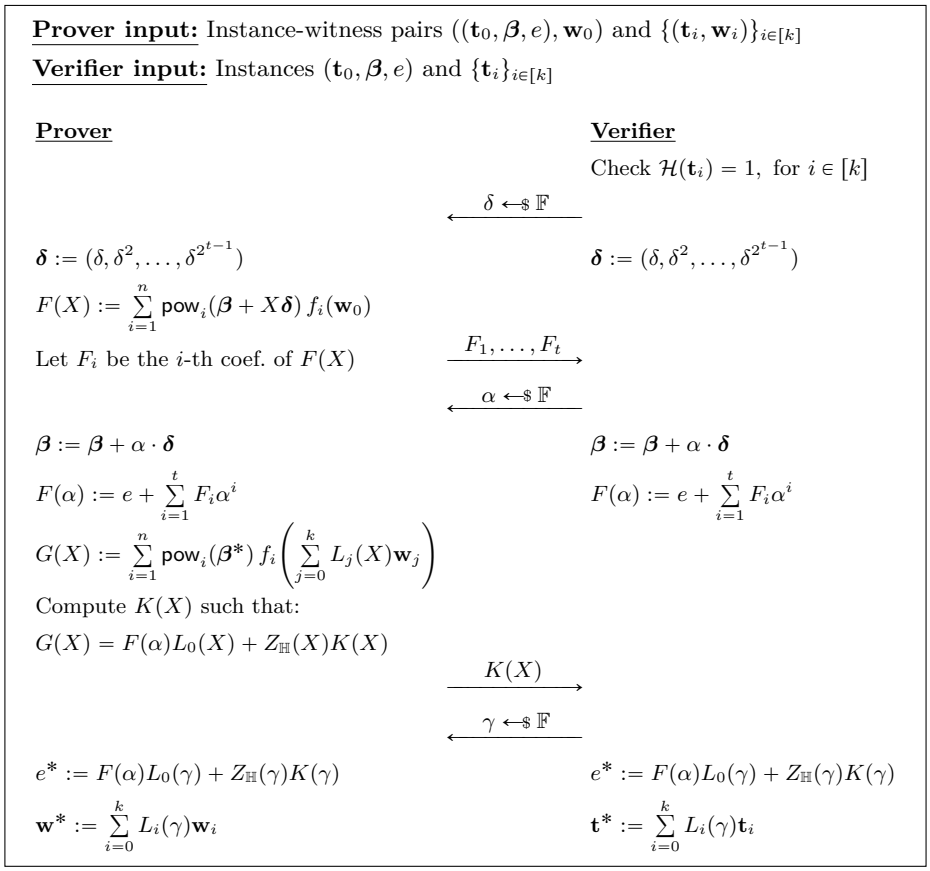

Fig. 4. Protogalaxy folding protocol $\mathcal{R}_{\mathsf{rand}} \times \mathcal{R}^k \to \mathcal{R}_{\mathsf{rand}}$

Both the correctness and the security of the protocol (as a reduction of knowledge) crucially rely on the following result about Lagrange polynomials, which we adapt from [12, Lemma 4.2].

Lemma 2. *Let $f : \mathbb{F}^M \to \mathbb{F}$ be a polynomial map and $\mathbf{w}_1, \ldots, \mathbf{w}_k \in \mathbb{F}^m$ vectors. Let $(L_i)_{i \in [k]}$ be the Lagrange basis over a domain $\mathbb{H} \subset \mathbb{F}$ of size k and $Z_{\mathbb{H}}(X)$*

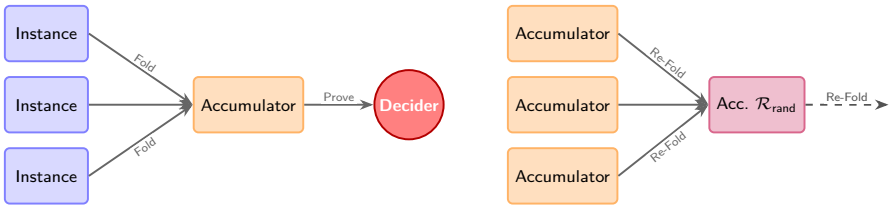

Fig. 5. On the left Accordion pipeline for proving batched instances and on the right Folding Accumulators for realizing PCD from Accordion

the vanishing polynomial of $\mathbb{H}$. *Then, we have that*

$$f\left(\sum_{i=1}^{k} L_i(X)\mathbf{w}_i\right) = \sum_{i=1}^{k} L_i(X) f(\mathbf{w}_i) \mod Z_{\mathbb{H}}(X)$$

Proof. It suffices to show that $Z_{\mathbb{H}}(X)$ divides the polynomial given by $H(X) = f\left(\sum_{i=1}^{k} L_i(X)\mathbf{w}_i\right) - \sum_{i=1}^{k} L_i(X) f(\mathbf{w}_i)$. To see this, note that given any $j \in [k]$, and taking $a_j \in \mathbb{H}$, we have that $L_j(a_j) = 1$ and $L_i(a_j) = 0$ for any $i \in [k], i \neq j$. Therefore,

$$\begin{aligned} H(a_j) &= f\left(\sum_{i=1}^{k} L_i(a_j)\mathbf{w}_i\right) - \sum_{i=1}^{k} L_i(a_j) f(\mathbf{w}_i) \\ &= f(L_j(a_j)\mathbf{w}_j) - L_j(a_j) f(\mathbf{w}_j) = 0 \end{aligned}$$

As $H(a_j) = 0$ for every $j \in [k]$, then $Z_{\mathbb{H}}(X)$ divides $H(X)$ as claimed. □

For the full soundness and completeness proofs, we refer to [12].

C Folding Accumulators for PCD

In this section, we introduce an extension of Protogalaxy that enables one to fold several accumulators with different randomizing coefficients into one. In practice, this immediately enables Proof-Carrying Data [11,9] from Protogalaxy.

We will fold the generalized randomized relation $\mathcal{R}_{\mathsf{gen}}$ from Definition 6, $\mathcal{R}_{\mathsf{gen}} \subseteq (\mathcal{CR} \times \mathbb{F}^n \times \mathbb{F}) \times \mathbb{F}^M$:

$$\mathcal{R}_{\mathsf{gen}} := \left\{ ((\mathbf{t}, \boldsymbol{b}, e), \mathbf{w}) \;\middle|\; \mathbf{t} = \mathsf{cm}(\mathbf{w}) \;\wedge\; \sum_{i=1}^{n} b_i f_i(\mathbf{w}) = e \right\}.$$

Our goal is to fold k instances of $\mathcal{R}_{\mathsf{gen}}$ into a single instance of $\mathcal{R}_{\mathsf{gen}}$. We denote the starting accumulators $\mathbf{w}_1, \ldots, \mathbf{w}_k$ with commitments $\mathbf{t}_1, \ldots, \mathbf{t}_k$, coefficients $\mathbf{b}_1, \ldots, \mathbf{b}_k$, and errors $e_1, \ldots, e_k$. The protocol will fold them into a single accumulator $((\mathbf{t}^*, \mathbf{b}^*, e^*), \mathbf{w}^*)$. Note that $\mathcal{H}$-consistency is not checked as all the initial relations are already accumulators.

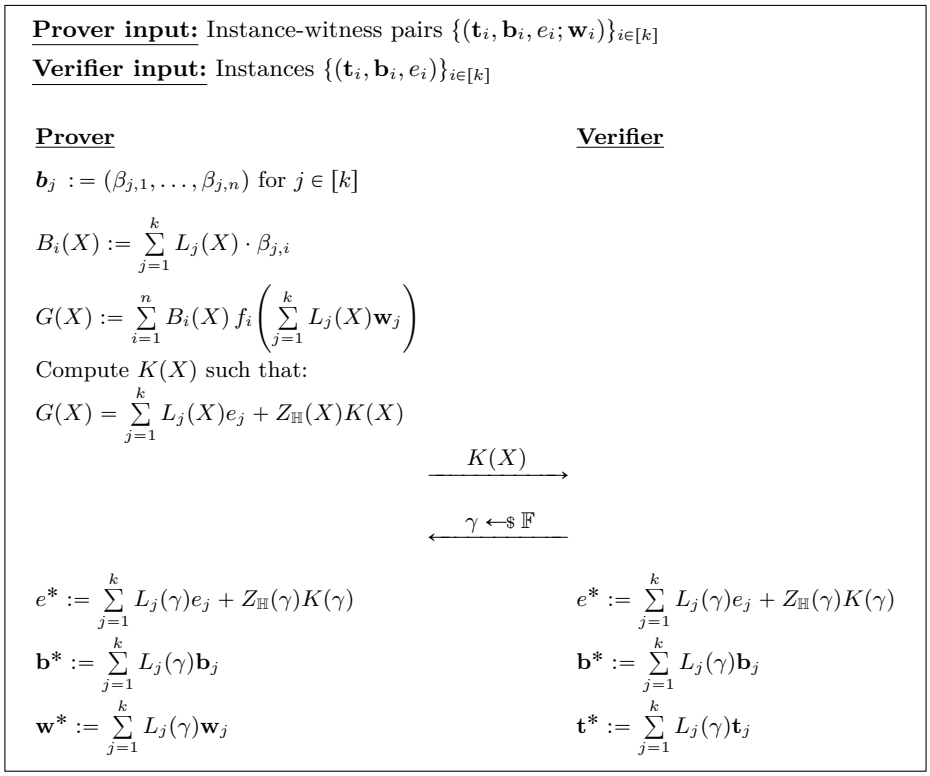

Fig. 6. Accumulator folding protocol $\mathcal{R}^k_{\mathsf{gen}} \to \mathcal{R}_{\mathsf{gen}}$

For simplicity, we present the intuition of the protocol for $\mathcal{R}_{\mathsf{gen}}$, that is, folding when $\mathbf{b}_j \in \mathbb{F}^n$ and the randomizers of $f_i(\mathbf{w}_j)$ are not structured. We start by considering a polynomial $G(X)$ as follows: $G(X) := \sum_{i\in[n]} \beta^*_i f_i\left(\sum_{j\in[k]} L_j(X) \cdot \mathbf{w}_j\right)$. Where $\mathbf{b}^* \in \mathbb{F}^n$ is some vector of randomizers which is yet undefined. In the folding process, our goal is to show that the evaluation of the final accumulator $G(\mathbf{w}^*) = e^*$ while connecting the evaluations of $G(X)$ to the errors e_j of the initial accumulators.

To do this, we first note that $\sum_{i\in[n]} \beta_{j,i} f_i(\mathbf{w}_j) = e_j$ for all $j \in [k]$, where $\mathbf{b}_j = (\beta_{j,1}, \ldots, \beta_{j,n})$ is the coefficient vector for instance j. Our approach is to consider $\mathbf{b}^*$ as a function itself, that is a n-coordinate map $B : \mathbb{F} \to \mathbb{F}^n$ such that $B_i(X)$ is a univariate polynomial for each $i \in [n]$. B must evaluate to $\mathbf{b}_j$ at $X = a_j$, where $a_j \in \mathbb{H}$ is the domain point such that $L_j(a_j) = 1$. Then, we can recover an algebraic relation between $G(X)$ and the errors e_j's. For this, we define the n components of $B(X)$ as $B_i(X) := \sum_{j\in[k]} L_j(X) \cdot \beta_{j,i}$. Then, we have that $G(X) := \sum_{i\in[n]} \left(\sum_{j\in[k]} L_j(X) \cdot \beta_{j,i}\right) f_i\left(\sum_{j\in[k]} L_j(X) \cdot \mathbf{w}_j\right)$.

And modulo $Z_{\mathbb{H}}(X)$, we can apply Lemma 2 to obtain the following expression

$$\begin{aligned} G(X) &= \sum_{i\in[n]} \left(\sum_{j\in[k]} L_j(X)\cdot\beta_{j,i}\right)\left(\sum_{j\in[k]} L_j(X)\cdot f_i(\mathbf{w}_j)\right) && \bmod Z_{\mathbb{H}}(X)\\ &= \sum_{i\in[n]}\sum_{j,j'\in[k]} L_j(X)\cdot\beta_{j,i}\cdot L_{j'}(X)\cdot f_i(\mathbf{w}_{j'}) && \bmod Z_{\mathbb{H}}(X)\\ &= \sum_{i\in[n]}\sum_{j\in[k]} L_j(X)\cdot\beta_{j,i}\cdot f_i(\mathbf{w}_j) && \bmod Z_{\mathbb{H}}(X)\\ &= \sum_{j\in[k]} L_j(X)\left(\sum_{i\in[n]}\beta_{j,i}\cdot f_i(\mathbf{w}_j)\right) = \sum_{j\in[k]} L_j(X)e_j && \bmod Z_{\mathbb{H}}(X) \end{aligned}$$

Where in the third step we have used that $L_j(X)\cdot L_{j'}(X) = 0$ for $j \neq j'$ and that $L_j(X)^2 = L_j(X)$ holds modulo $Z_{\mathbb{H}}(X)$. We now state the protocol as a reduction of knowledge and describe it in Figure 6.

Theorem 2. *The protocol in Figure 6 is a reduction of knowledge for the relations* $\mathcal{R}^k_{\mathsf{gen}} \to \mathcal{R}_{\mathsf{gen}}$.

Proof. Correctness follows easily from the relations on Lagrange polynomials that we introduced just above.

For security, consider a successful prover. The goal is to extract non-committed transcripts $\mathbf{w}_1,\ldots,\mathbf{w}_k$ and prove that they satisfy general Protogalaxy relations, i.e., that the initial transcripts are in $\mathcal{R}_{\mathsf{gen}}$. The extractor proceeds by rewinding the prover $(d+1)k$ times, obtaining a tree of transcripts. We start by taking k arbitrary executions with challenges $\gamma_1,\ldots,\gamma_k$ and apply the following lemma to obtain a tuple of witnesses $\mathbf{w}_j$ for every $j\in[k]$.

Lemma 3. *Given a* $((d+1)k)$*-valid tree of transcripts, there exists an efficient algorithm that finds either a collision on* cm *or a tuple of witnesses* $(\mathbf{w}_1,\ldots,\mathbf{w}_k)$ *such that:*

- $\mathsf{cm}(\mathbf{w}_j) = \mathbf{t}_j$ *for every* $j\in[k]$*, and*
- $\mathbf{w}^*_a = \sum_{j\in[k]} L_j(\gamma_a)\cdot\mathbf{w}_j$ *for every* $a\in[(d+1)k]$.

Proof. By the final verifier computation, we know that $\mathbf{t}^*_a = \sum_{j\in[k]} L_j(\gamma_a)\cdot\mathbf{t}_i$ for every $a\in[n]$. Take k arbitrary tree paths such that all γ_a are pairwise distinct, and denote the corresponding leaves as $(\mathbf{t}^*_i,\mathbf{w}^*_i)$ for every $j\in[k]$. Then, from the identity:

$$\mathbf{t}^*_i = \sum_{j\in[k]} L_j(\gamma_i)\cdot\mathbf{t}_j$$

obtain the $k \times k$ matrix of coefficients $\{c_{i,j}\}$ that result from inverting the matrix of $L_j(\gamma_i)$ coefficients. Such coefficients satisfy the identity

$$\mathbf{t}_i = \sum_{j \in [k]} c_{i,j} \cdot \mathbf{t}^*_j$$

and thus we can define

$$\mathbf{w}_i = \sum_{j \in [k]} c_{i,j} \cdot \mathbf{w}^*_j.$$

Note that, by the additive homomorphism of the commitment scheme, this means that $\mathbf{t}_j = \mathsf{cm}(\mathbf{w}_j)$ for every $j \in [k]$, satisfying the first property. Also, note that by construction the second property also holds for the selected witnesses. By repeating the procedure, we observe that the condition holds for all tree leaves (the matrix of coefficients is always invertible, as the Lagrange basis evaluation is an isomorphism) unless there exists a collision $\mathbf{t}^*_a = \mathbf{t}^*_{a'}$ for distinct leaves. In this case, if $\mathbf{w}^*_{a'} \neq \mathbf{w}^*_{a'}$, the algorithm stops and outputs a collision on the commitment scheme. Otherwise, one can simply recompute the coefficient matrix with difference instances. □

It remains to show that the extracted witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_k$ are valid accumulators. Consider the verification equations. For every branch in the tree with challenge γ_a such that $a \in [(d+1)k]$, it holds that $e^*_a = K_a(\gamma_a) \cdot Z_{\mathbb{H}}(\gamma_a) + \sum_{j \in [k]} e_j L_j(\gamma_a)$. Similarly, as all instances in the tree leaves are valid instances, we have that $\sum_{i \in [n]} \beta^*_{a,i} f_i(\mathbf{w}^*_a) = e^*_a$.

Finally, we know that $\beta^*_{a,i} = \sum_{j \in [k]} L_j(\gamma_a)\beta_{j,i}$. Putting everything together,

$$\sum_{i \in [n]} \left(\sum_{j \in [k]} L_j(\gamma_a)\beta_{j,i} \right) f_i(\mathbf{w}^*_a) = K_a(\gamma_a) \cdot Z_{\mathbb{H}}(\gamma_a) + \sum_{j \in [k]} e_j L_j(\gamma_a)$$

Using the expression for $\mathbf{w}^*_a$ for the lemma, this yields

$$\sum_{i \in [n]} \left(\sum_{j \in [k]} L_j(\gamma_a)\beta_{j,i} \right) f_i \left(\sum_{j \in [k]} L_j(\gamma_a) \cdot \mathbf{w}_j \right) = K_a(\gamma_a) \cdot Z_{\mathbb{H}}(\gamma_a) + \sum_{j \in [k]} e_j L_j(\gamma_a)$$

As the equation holds for $(d+1)k$ different challenges, the following identity holds for polynomials, noting that these (and in particular $G(X)$) have degree at most $(d+1)(k-1)+1$:

$$\sum_{i \in [n]} \left(\sum_{j \in [k]} L_j(X)\beta_{j,i} \right) f_i \left(\sum_{j \in [k]} L_j(X) \cdot \mathbf{w}_j \right) = K_a(X) \cdot Z_{\mathbb{H}}(X) + \sum_{j \in [k]} e_j L_j(X)$$

If we consider it modulo $Z_{\mathbb{H}}(X)$, we can apply Lemma 2

$$\sum_{i \in [n]} \left(\sum_{j \in [k]} L_j(X)\beta_{j,i} \right) \sum_{j \in [k]} L_j(X) f_i\left(\mathbf{w}_j\right) - \sum_{j \in [k]} e_j L_j(X) = 0 \mod Z_{\mathbb{H}}(X)$$

which implies, using the properties of Lagrange polynomials modulo $Z_{\mathbb{H}}(X)$, that

$$\sum_{j\in[k]} L_j(X)\left(\sum_{i\in[n]} \beta_{j,i} f_i(\mathbf{w}_j) - e_j\right) = 0 \mod Z_{\mathbb{H}}(X)$$

As the Lagrange polynomials $\{L_j(X)\}_{j\in[k]}$ form a basis of $\mathbb{F}[X]/Z_{\mathbb{H}}(X)$ as an $\mathbb{F}$-module, this implies that $\sum_{i\in[n]} \beta_{j,i} f_i(\mathbf{w}_j) = e_j$ for every $j \in [k]$, and we conclude that $\mathbf{w}_j$ is a valid witness. □

Extending the protocol to structured relations. We presented the protocol for the general relation $\mathcal{R}_{\mathsf{gen}}$, for which $\mathbf{b} \in \mathbb{F}^n$. While elegant, this is a useful relation to work with only if one plans to prove the folded accumulator directly, e.g. by running the decider of the folding scheme or by using a SNARK to prove that $(\mathbf{w}^*, \mathbf{t}^*, \mathbf{b}^*, e^*) \in \mathcal{R}_{\mathsf{gen}}$. If one wants to continue folding, the structured relation $\mathcal{R}_{\mathsf{rand}}$ (Definition 12) where $\boldsymbol{\beta}^* \in \mathbb{F}^{\log n}$ such that $b_i = \mathsf{pow}_i(\boldsymbol{\beta}^*)$ needs to be supported. Fortunately, this is fairly straightforward. To preserve the structure, the coefficient polynomial $B(X)$ must be defined as $B_i(X) = \mathsf{pow}_i(\sum_{j\in[k]} L_j(X) \cdot \boldsymbol{\beta}_j)$ for $i \in [\log n]$. In this case, $\deg(B) = \log n \cdot (k-1)$. Correctness follows easily, as by the definition of the power function $\mathsf{pow}(X_1, \ldots, X_t) = (1, X_t) \otimes \cdots \otimes (1, X_1) \in \mathbb{F}[X_1, \ldots, X_t]^{2^t}$, where $\mathsf{pow}_i(X_1, \ldots, X_t)$ is its i-th coordinate:

$$\begin{aligned} B_i(X) &= \mathsf{pow}_i\left(\sum_{j\in[k]} L_j(X) \cdot \boldsymbol{\beta}_j\right) && \mod Z_{\mathbb{H}}(X) \\ &= \prod_{\ell\in S_i} L_j(X) \cdot \beta_{j,\ell} = L_j(X) \cdot \prod_{\ell\in S_i} \beta_{j,\ell} && \mod Z_{\mathbb{H}}(X) \end{aligned}$$

as $L_j(X)$ are idempotent modulo $Z_{\mathbb{H}}(X)$, and the rest follows as above.

For security, the only relevant change is to increase the size of the tree of transcripts. The degree of $G(X)$ is $d(k-1) + \deg(B)$, so in this case we have that $\deg(G) = (d + \log n)(k-1)$. Therefore, the tree of transcripts needs to have $\deg(G) + 1$ branches.

D Proof of Theorem 1

Proof. To prove security, we model our protocol as a two-round public-coin interactive protocol. The challenger of the verifier in the first round is β, and the one in the second round is γ. Our goal is to show that the protocol is a reduction of knowledge. In other words, we will show that there exists an extractor $\mathcal{E}$ such that, with rewinding access to any successful prover, it outputs either (a) a valid set of k witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_k$ such that $(\mathbf{t}_i, \mathbf{w}_i) \in \mathcal{R}$ for every $i \in [k]$, or (b) a collision on the commitment scheme.

The strategy of the extractor is as follows: by rewinding the prover sufficiently many times, it builds a $(n, dk+1)$-valid tree of transcripts, which can

be computed following the strategy in [2]. Given a valid $(n, dk+1)$-tree of transcripts, we show that the extractor succeeds in outputting either the collision or the witnesses. Recall that the statement of the proof includes the k folded instances $(\mathbf{t}_1, \ldots, \mathbf{t}_k)$.

Let $\beta_a, \gamma_{a,b}$ be the values for challenges β, γ used in the corresponding path of the tree of transcripts, and by $(\mathbf{t}^*_{a,b}, \mathbf{w}^*_{a,b}, e^*_{a,b})$ the final output in that path (i.e., located at a tree leaf), corresponding with a valid folded $\mathcal{R}_{\mathsf{P}}$ instance. The variables $a \in [n]$ and $b \in [dk+1]$ locate each path of the tree. We start by proving an initial lemma.

Lemma 4. *Given a $(n, dk+1)$-valid tree of transcripts, there exists an efficient algorithm that finds either a collision on cm or a tuple of witnesses $(\mathbf{w}_1, \ldots, \mathbf{w}_k)$ such that:*

- $\mathsf{cm}(\mathbf{w}_j) = \mathbf{t}_j$ *for every* $j \in [k]$*, and*
- $\mathbf{w}^*_{a,b} = \sum_{j\in[k]} L_j(\gamma_{a,b}) \cdot \mathbf{w}_j$ *for every* $a \in [n], b \in [dk+1]$.

Proof. By the final verifier computation, we know that $\mathbf{t}^*_{a,b} = \sum_{j\in[k]} L_j(\gamma_{a,b}) \cdot \mathbf{t}_i$ for every $a \in [n], b \in [dk+1]$. Take k arbitrary tree paths (a, b) such that all $\gamma_{a,b}$ are pairwise distinct, and denote the corresponding leaves as $(\mathbf{t}^*_i, \mathbf{w}^*_i)$ for every $j \in [k]$. Then, from the identity:

$$\mathbf{t}^*_i = \sum_{j\in[k]} L_j(\gamma_i) \cdot \mathbf{t}_j$$

obtain the $k \times k$ matrix of coefficients $\{c_{i,j}\}$ that result from inverting the matrix of $L_j(\gamma_i)$ coefficients. Such coefficients satisfy the identity

$$\mathbf{t}_i = \sum_{j\in[k]} c_{i,j} \cdot \mathbf{t}^*_j$$

and thus we can define

$$\mathbf{w}_i = \sum_{j\in[k]} c_{i,j} \cdot \mathbf{w}^*_j.$$

Note that, by the additive homomorphism of the commitment scheme, this means that $\mathbf{t}_j = \mathsf{cm}(\mathbf{w}_j)$ for every $j \in [k]$, satisfying the first property. Also, note that by construction the second property also holds for the selected witnesses. By repeating the procedure, we observe that the condition holds for all tree leaves except if the matrix of coefficients is not invertible, which occurs with negligible probability unless there exists a collision $\mathbf{t}^*_{a,b} = \mathbf{t}^*_{a',b'}$ for distinct leaves. In this case, if $\mathbf{w}^*_{a',b'} \neq \mathbf{w}^*_{a',b'}$, the algorithm stops and outputs a collision on the commitment scheme. Otherwise, one can simply recompute the coefficient matrix with another $k-1$ different instances.

Next, consider the verification equations of the protocol. Observe that, for every a, b in the tree, it holds that $e^*_{a,b} = Z_{\mathbb{H}}(\gamma_{a,b}) \cdot K_a(\gamma_{a,b})$ (note that there is a different polynomial K for each a. Similarly, since all instances at the tree leaves

are valid instances of $\mathcal{R}_{\mathsf{P}}$, it follows that $\sum_{i\in[n]} P_i(\beta_a) f_i(\mathbf{w}^*_{a,b}) = e^*_{a,b}$. Putting both equations together, we have

$$\sum_{i\in[n]} P_i(\beta_a) f_i(\mathbf{w}^*_{a,b}) = Z_{\mathbb{H}}(\gamma_{a,b}) \cdot K_a(\gamma_{a,b}).$$

Using the expression from Lemma 4, this yields

$$\sum_{i\in[n]} P_i(\beta_a) f_i \left(\sum_{j\in[k]} L_j(\gamma_{a,b}) \cdot \mathbf{w}_i \right) = Z_{\mathbb{H}}(\gamma_{a,b}) \cdot K_a(\gamma_{a,b}).$$

Note that the above equation holds for $dk+1$ different challenges $\gamma_{a,b}$ for a fixed a (i.e. for a fixed β_a). Besides, the degree of the polynomials of the equation on γ is at most dk, as d is the maximum degree of f_i and the Lagrange polynomials L_i have degree at most k. It follows that the above equation holds as a polynomial identity,

$$\sum_{i\in[n]} P_i(\beta_a) f_i \left(\sum_{j\in[k]} L_j(X) \cdot \mathbf{w}_j \right) = Z_{\mathbb{H}}(X) \cdot K_a(X).$$

Let us now consider the above equation mod $Z_{\mathbb{H}}(X)$. It holds that

$$\sum_{j\in[k]} L_j(X) \cdot \sum_{i\in[n]} P_i(\beta_a) f_i\left(\mathbf{w}_j\right) = 0 \mod Z_{\mathbb{H}}(X).$$

where we can factor out the $L_j(X)$ terms due to Lemma 2. Clearly the Lagrange polynomials are non-zero mod $Z_{\mathbb{H}}(X)$, so the equation implies that

$$\sum_{i\in[n]} P_i(\beta_a) f_i\left(\mathbf{w}_j\right) = 0 \text{ for every } j \in [k].$$

Finally, consider the polynomials $Q_j(Y) = \sum_{i\in[n]} P_i(Y) f_i\left(\mathbf{w}_j\right) = 0$ for each $j \in [k]$. Since it holds by the previous step that each $Q_j(\beta_a) = 0$ for every $a \in [n]$, the β_a are all distinct, and the polynomials $\{P_i(Y)\}_{i\in[n]}$ have degree at most $n-1$, it follows that every $Q_j(Y)$ must be identically zero. As the P_i's are linearly independent, we conclude that $f_i(\mathbf{w}_j) = 0$ for every $i \in [n], j \in [k]$.