PROTOGALAXY: Efficient PROTOSTAR-style folding of multiple instances

Liam Eagen[1,2] and Ariel Gabizon[1]

[1]Zeta Function Technologies
[2]Blockstream Research

August 29, 2023

Abstract

We continue the recent line of work on folding schemes. Building on ideas from PROTOSTAR [BC23] we construct a folding scheme where the recursive verifier's "marginal work", beyond linearly combining witness commitments, consists only of a logarithmic number of field operations and a constant number of hashes. Moreover, our folding scheme performs well when *folding multiple instances at one step*, in which case the marginal number of verifier field operations per instance becomes constant, assuming constant degree gates.

1 Introduction

In the last two years, we have seen an explosion of interest in so-called folding[1] schemes [BCMS20, BCL+21, KST21, Moh, KS22, KS23, BC23]. Roughly speaking, a folding scheme "folds" several instances of a relation into one instance in a way that the correctness of this one instance implies the correctness of all original instances.

The original motivation for folding schemes was improving the efficiency of incrementally verifiable computation[Val08] and proof carrying data [CT10].

A related motivation is improving SNARK prover time when proving correctness of multiple instances of a relation: Given instances $\phi_1, \ldots, \phi_m$, rather than computing a SNARK proof Π_i for each one, we fold[2] them into a single instance ϕ^* for which produce a single SNARK proof Π. If the computation required per instance ϕ_i to show folding was done correctly is significantly smaller than that for producing the SNARK proof Π_i, the

[1]The term *accumulation schemes* [BCMS20, BCL+21] was introduced first for the same primitive. We alternate between the folding and accumulation terminologies.

[2]To apply folding correctly in this usecase requires some additional work to make sure we have indeed folded the instances we are interested in; for example, computing a running hash of the instances during folding, and comparing it to a running hash independently computed by the final verifier.

overall prover work will be smaller even after computing the proof for ϕ^*. Alternatively, in the IVC context it is often recommended to simply use the witness for ϕ^* as the final proof; then, at the expense of a longer proof, no SNARK is needed at all.

1.1 Single-instance vs Multi-instance folding

PROTOGALAXY focuses on *k-folding* - folding k instances in one folding operation; which was also addressed in [BCL+21, KS23]. The benefit of k-folding is giving a wider set of tradeoffs between folding and final decision costs, as well as reducing their combined cost.

Generally, the goal of folding schemes has been to simplify the verifier as much as possible and move as much computation as possible out of the verifier and prover and into the decider. This is a sensible assumption when a folding scheme is used to instantiate a long running IVC protocol, but limits applicability in other cases where the decider must be run frequently. For example, consider a decentralized computation where multiple parties perform computations in a directed graph, as in PCD. If the parties do not trust one another, each will need to run the decider on their input accumulators they receive before folding new instances. Since each party will run both the prover and the decider, moving computation from the prover to the decider does not reduce the overall complexity of the protocol. This situation may be of particular interest for decentralized cryptocurrencies, where a large group of mutually distrusting actors are simultaneously manipulating a shared state.

When folding multiple instances, we can take any combination of the following three approaches.

First, the prover can simply fold the function $F(X)$ and the folding verifier sequentially k times. This does not increase decider complexity, compared to folding once, but increases the prover complexity by k times the cost of proving the verifier, dominated by (in PROTOSTAR) $O(1)$ elliptic curve scalar multiplications and hash functions per verifier.

Second, the prover can fold the k composition of $F(X)$, $G(X) = F(F(\dots F(X)\dots)$ a single time. This keeps the recursion overhead small, but increases the decider complexity linearly in k times the complexity of $|F|$.

Finally, the prover can use a k-folding scheme like PROTOGALAXY. This allows keeping the decider complexity corresponding to a *single application* of F. Moreover, when a single party performs *both* the folding and deciding this can still result in smaller overall work. One reason for this is that when comparing to the second approach, k-folding requires an in circuit k-size MSM to combine instances, which due to Pippenger can be significantly faster than k separate scalar multiplications required in the second approach.

All three can be combined to find the optimal configuration for a particular F, i.e. a prover can fold a instances of the b fold composition of F sequentially c times where $k = abc$.

1.2 Previous work on folding schemes and our results

All known folding schemes rely on additively homomorphic vector commitments, and the following template: We assume the folding verifier is given homomorphic commitments to the witnesses. The verifier takes a random combination of the witness commitments. The prover computes the corresponding combination of the witnesses themselves. Since all folding schemes share this work, we can define the *marginal work* of the prover and verifier as the additional work beyond this linear combination of witnesses and their commitments.

It is hard pinning down precise prover marginal costs, especially for [BC23]. At a high level, assuming the verifier equation is computed by a constant size arithemtic circuit, all schemes require in total $O(n)$ field and group operations per folding operation for a size n witness. It is worth mentioning that this holds for HyperNova even when the verifer circuit contains at the base layer a *super-constant* linear component.[3]

Focusing on the verifier marginal work, we can easily see the difference between the two latest folding schemes - HyperNova [KS23] and ProtoStar [BC23], and ProtoGalaxy. The differences in verifier work are more crucial as they must be performed *in-circuit*.

- HyperNova performs a randomized sumcheck protocol to accumulate a new instance - except that the final (expensive) multilinear opening is delayed to the final decision. Thus, the verifier requires $d \log n$ field operations - to evaluate the intermediate univariates sent during sumcheck, and $\log n$ random-oracle like hashes - to obtain the challenges for the sumcheck.

- ProtoStar, rather than actually performing a sumcheck, simply reduces[4] the instance to another that is a randomized sum of the instance, and then folds *this claim about the randomized sum*. Most of the folding work, in fact, is related to combining the random coefficients for the new instance's sum with the current accumulator, and proving those coefficients are of an appropriate form - specifically, consecutive powers of a challenge β. This process requires *committing* to the vector of powers of β, which adds group operations to the folding verifier that needs to perform scalar multiplications on these commitments.

- ProtoGalaxy instead of combining randomized sums with different challenges, "converts" the accumulator to a claim using the new challenge. This avoids the need to commit to the power vector, and allows remaining with a succinct ($\log n$ length) representation of the sum's coefficients. An interesting point is that although this results in a polynomial of degree $\log n$ expressed as a sum of length n, we are able to recursively compute its coefficients in $O(n)$ rather than $O(n \log n)$ operations (cf. Claim 4.4).

[3] It seems this feature can be encorporated into ProtoStar and ProtoGalaxy by "delaying" linchecks to decision time in a similar way to HyperNova.

[4] We are describing what is presented in Section 3.5 of [BC23]: "Compressing verification checks for high-degree verifiers".

The second difference between PROTOSTAR and PROTOGALAXY, responsible for more efficient folding of multiple instances, is a transition from monomial to Lagrange base: PROTOSTAR's folding creates a polynomial whose constant coefficient is the current accumulator's sum, and the coefficient of X^d is the new instances's sum - where d is the maximal degree of the verifier's checks. Attempting to do this for k instances at once requires a degree growing exponentially in k to make sure coefficients relating to different instances don't "mix".

However, things are smoother in Lagrange base. We associate with each instance to be folded a different Lagrange coefficient. Essentially due to the property that a power of Lagrange polynomial $L_i(X)$ modulu the set's vanishing polynomial is $L_i(X)$ itself, there is no need for exponentially growing degrees in the Lagrange approach. Another simplification due to the Lagrange base, is avoiding the need to homogenize the verifier's constraint polynomials as done in PROTOSTAR. See Section 4 for the precise construction.

Table 1: Comparing folding verifiers. CRH = collision resistant hash, RO="random oracle like" hash. d=degree of verifier checks.

Scheme	**Marginal verifier work**
HyperNova	$O(d \log n)$ $\mathbb{F}$, $d \log n$ CRH, $\log n$ RO
PROTOSTAR	2 $\mathbb{G}$, $d + O(1)$ $\mathbb{F}$, $d + O(1)$ CRH, $O(1)$ RO
PROTOGALAXY	$d + \log n$ $\mathbb{F}$, $d + \log n$ CRH, $O(1)$ RO
PROTOGALAXY -k instances	$kd + \log n$ $\mathbb{F}$, $kd + \log n$ CRH, $O(1)$ RO
Section 5 -k instances	$\log n + d \log k$ $\mathbb{F}$, $kd + \log n$ CRH, $\log k$ RO
Section 5 -k accumulators	$\log k(\log n + d)$ $\mathbb{F}$, $k(d + \log n)$ CRH, $\log k$ RO

2 Terminology and Conventions

We assume our field $\mathbb{F}$ is of prime order. We assume all algorithms described receive as an implicit parameter the security parameter λ.

Whenever we use the term *efficient*, we mean an algorithm running in time $\mathsf{poly}(\lambda)$. Furthermore, we assume an *object generator* $\mathcal{O}$ that is run with input λ before all protocols, and returns all fields and groups used. Specifically, in our protocol $\mathcal{O}(\lambda) = (\mathbb{F}, \mathbb{G})$ where

- $\mathbb{F}$ is a prime field of super-polynomial size $r = \lambda^{\omega(1)}$.
- $\mathbb{G}$ is a group of size r.

We usually let the λ parameter be implicit, i.e. write $\mathbb{F}$ instead of $\mathbb{F}(\lambda)$. We write $\mathbb{G}$ additively.

We often denote by $[n]$ the integers $\{1, \ldots, n\}$. We use the acronym e.w.p for "except with probability"; i.e. e.w.p γ means with probability *at least* $1 - \gamma$.

3 Definitions based on PROTOSTAR

We review and adapt definitions from [BC23]. We recommend reading the referenced sections for full understanding and context. All protocols described are between a prover $\mathbf{P}$ and verifier $\mathbf{V}$. We use the convention that "the protocol has input $(a; b)$" means that both $\mathbf{P}$ and $\mathbf{V}$ get a, but only $\mathbf{P}$ gets b.

3.1 Special-sound protocols

Following [AFK22, BC23] we define special-sound protocols. We recall first the definition from Section 2.1 of [BC23], of an a^μ-out-of-N special-sound interactive protocol: Roughly speaking, this means a *public-coin* protocol where verifier challenges are chosen from a domain of size N, and a tree of accepting transcripts of arity a is sufficient for extracting a witness. Based on that definition, we say that a $(2\mu + 1)$-move interactive protocol (meaning $\mu+1$ prover messages and μ verifier messages) is *special-sound*, if it is a^μ-out-of N special-sound, for $a = \mathsf{poly}(\lambda)$ and $N = \lambda^{\omega(1)}$. As explained in [BC23], the results of [AFK22] impy the non-interactive version of the protocol in the random oracle model is knowledge-sound with error $\mathsf{negl}(\lambda)$.

3.2 Defining accumulation/folding schemes

We define accumulation schemes in an equivalent way to [BC23] but using different terminology, more focused on relations. In the definition below we are "accumulating" instances of a relation $\mathcal{R}$ into an "accumulator" which is an instance of a relation $\mathcal{R}_{\text{acc}}$. The reason we need the additional relation $\mathcal{R}_{\text{acc}}$ - rather than folding directly into $\mathcal{R}$ - is that the accumulator requires more flexibility in its structure. For example, in Nova[KST21], $\mathcal{R}_{\text{acc}}$ corresponds to *relaxed* committed R1CS.

Definition 3.1. *Fix relations $\mathcal{R}$ and $\mathcal{R}_{\text{acc}}$. An $(\mathcal{R} \mapsto \mathcal{R}_{\text{acc}})$*-accumulation/folding scheme *is a public-coin interactive protocol $\mathscr{P}$ between a prover $\mathbf{P}$ and verifier $\mathbf{V}$ such that*

1. *The protocol input is $(\phi, \phi'; \omega, \omega')$.*
2. *When the protocol ends $\mathbf{V}$ outputs ϕ^*, and $\mathbf{P}$ outputs ω^*.*
3. ***Completeness:** If $(\phi, \omega) \in \mathcal{R}_{\text{acc}}, (\phi', \omega') \in \mathcal{R}$, and $\mathbf{P}$,$\mathbf{V}$ follow the protocol, we have with probability one that $(\phi^*, \omega^*) \in \mathcal{R}_{\text{acc}}$.*
4. ***Knowledge soundness:***

   The following protocol $\mathscr{P}^*$ between $\mathbf{P}^*$ and $\mathbf{V}^*$ is knowledge-sound with error $\mathsf{negl}(\lambda)$ for the relation $\mathcal{R}_{\text{acc}} \times \mathcal{R}$:

(a) Given inputs $(\phi, \phi'; \omega, \omega')$*,* $\mathbf{P}^*$ *and* $\mathbf{V}^*$ *run the protocol* $\mathscr{P}$ *as* $\mathbf{P}, \mathbf{V}$ *with the same inputs.*

(b) Let $\mathfrak{p}^* = (\phi^*, \omega^*)$ *be the final output of* $\mathbf{P}$*,*$\mathbf{V}$ *in* $\mathscr{P}$*.* $\mathbf{V}^*$ *outputs* accept *if and only if* $\mathfrak{p}^* \in \mathcal{R}_{\text{acc}}$*.*

In words, knowledge soundness means that if the accumulation protocol was followed correctly, and either the current accumulator ϕ, or the current instance ϕ' is invalid, then the new accumulator ϕ^* will also be invalid, and thus will fail a final decision stage - in which a prover would need to show $(\phi^*, \omega^*) \in \mathcal{R}_{\text{acc}}$ for some ω^*.

3.3 Relations based on interactive protocols

To define and motivate the relations we use in our accumulation scheme, we continue to review interactive protocol conventions from [BC23].

Interactive protocols from [BC23]: Given a relation $\mathcal{R}_0$, we look at interactive protocols for $\mathcal{R}_0$ of the form described in Section 3.1 of [BC23]:

- The protocol is parameterized by positive integers $\mathfrak{k}, d, n$ ([BC23] uses ℓ rather than n).
- The protocol input is $(\phi_0; \omega_0)$ such that $(\phi_0, \omega_0) \in \mathcal{R}_0$.
- We have $\mathfrak{k}$ rounds where at round i $\mathbf{P}$ sends a vector m_i over $\mathbb{F}$, and, if $i < \mathfrak{k}$, $\mathbf{V}$ responds with random challenge $r_i \in \mathbb{F}$.
- Denote by ω the transcript of the protocol - namely the concatentation of ϕ_0, the vectors m_i, and elements r_i. Suppose ω is of length M. The protocol's definition includes a mapping $f : \mathbb{F}^M \to \mathbb{F}^n$. At the end $\mathbf{V}$ accepts if and only if $f(\omega) = 0^n$.

"Committed" protocols: Let cm be an additively homomorphic commitment function, mapping vectors over $\mathbb{F}$ of length at most M, into $\mathbb{G}$. Given cm, and an interactive protocol like the one described above, [BC23] now looks at a "committed" version of the protocol, where

- $\mathbf{P}$'s messages m_i are replaced by the commitments $C_i = \mathsf{cm}(m_i)$.
- At the end of the protocol $\mathbf{P}$ sends the vectors m_i.
- Define ω again as the concatenation of ϕ_0, the vectors m_i, and elements r_i. $\mathbf{V}$ accepts if and only if $f(\omega) = 0^n$ and for each $i \in [\mathfrak{k}]$, $\mathsf{cm}(\mathsf{m}_i) = C_i$.

Committed relations: Given a vector ω representing a transcript of the original non-committed protocol, we abuse notation and denote by $\phi = \mathsf{cm}(\omega)$ a "committed" version of the transcript, where every vector m_i is replaced by $\mathsf{cm}(m_i)$, but ϕ_0 and the $\{r_i\}$ are left as is. That is, if $\omega = (\phi_0, m_1, \ldots, m_{k+1}, r_1, \ldots, r_k)$

$$\phi = \mathsf{cm}(\omega) = (\phi_0, \mathsf{cm}(m_1), \ldots, \mathsf{cm}(m_{k+1}), r_1, \ldots, r_k).$$

In particular, ϕ is a mixture of field and group elements.

Given this notation, we define the relation $\mathcal{R}_{f,\mathsf{cm}}$ "induced" by the committed protocol for $\mathcal{R}_0$. $\mathcal{R}_{f,\mathsf{cm}}$ is defined as pairs of a committed transcript and the corresponding plain transcript. Formally, as all the pairs (ϕ, ω) such that

1. $\phi = \mathsf{cm}(\omega)$.
2. $f(\omega) = 0^n$.

Knowledge soundness for committed relations

We now deal with a technicality that comes up in both PROTOSTAR and PROTOGALAXY's knowledge soundness proof. The definition of a special-sound protocol requires that we are always able to extract a witness given a large enough tree of accepting transcripts. However, we'll only be able to *either* extract a witness or produce of collision of cm. We show this suffices.

Lemma 3.2. *Suppose* $\mathsf{cm} : \mathbb{F}^M \mapsto \mathbb{G}$ *is a binding commitment. Given a relation* $\mathcal{R}$*, define the relation* $\mathcal{R}' = \mathcal{R} \cup \{\phi, (w, w') | \mathsf{cm}(w) = \mathsf{cm}(w'), |\omega| \geq \lambda\}$*; in words, a collision of* cm *of appropriate length is a satisfying witness for any instance.*

If $\mathscr{P}$ *is special-sound for* $\mathcal{R}'$ *then it is knowledge-sound with error* $\mathsf{negl}(\lambda)$ *for* $\mathcal{R}$*.*

Proof. We can use the result of [AFK22] to conclude $\mathscr{P}$ is knowledge-sound for $\mathcal{R}'$ with error $\mathsf{negl}(\lambda)$. This means there is an extractor E, that when $\mathcal{A}$ convinces the verifier of the validity of instance ϕ, produces a witness ω such that $(\phi, \omega) \in \mathcal{R}'$ e.w.p $\mathsf{negl}(\lambda)$. We define extractor E_1 for $\mathcal{R}$: E_1 runs E, aborts if E's output ω is a collision of cm; and otherwise outputs ω. By the binding property of cm, the failure probability of E_1 is larger by at most $\mathsf{negl}(\lambda)$ than that of E; and when it succeeds $(\phi, \omega) \in \mathcal{R}$. Thus, $\mathscr{P}$ is knowledge-sound with error $\mathsf{negl}(\lambda)$ for $\mathcal{R}$. □

The relation $\mathcal{R}^{\mathsf{rand}}$: Let $t := \log n$. For $i \in \{0, \ldots, n-1\}$, let $S \subset \{0, \ldots, t-1\}$ be the set such that $i = \sum_{j \in [S]} 2^j$. We define the t-variate polynomial pow_i as

$$\mathsf{pow}_i(X_1, \ldots, X_t) = \prod_{\ell \in S} X_\ell$$

Note that if $\boldsymbol{\beta} = (\beta, \beta^2, \beta^4, \ldots, \beta^{2^{t-1}})$, $\mathsf{pow}_i(\boldsymbol{\beta}) = \beta^i$. For brevity, let $\mathcal{R} = \mathcal{R}_{f,\mathsf{cm}}$. We define the "randomized relaxed" version of $\mathcal{R}, \mathcal{R}^{\mathsf{rand}}$ - as follows.

$\mathcal{R}^{\mathsf{rand}}$ consists of the pairs $((\phi, \boldsymbol{\beta}, e), \omega)$ such that

1. $\phi = \mathsf{cm}(\omega)$.
2. $\boldsymbol{\beta} \in \mathbb{F}^t, e \in \mathbb{F}$ and we have

$$\sum_{i \in [n]} \mathsf{pow}_i(\boldsymbol{\beta}) f_i(\omega) = e.$$

(Here, f_i denotes the i'th output coordinate of f.)

4 Main Protocol

We are ready to present our main scheme. Fix positive integers $d, k < n$ such that $k+1$ is a power of two. Denote $t := \log n$. Assume $\mathbb{F}$ contains a multiplicative subgroup $\mathbb{H}$ of order $k+1$ and $L_0(X), L_1(X), \ldots, L_k(X)$ is its Lagrange base. Let $Z(X) := \prod_{a \in \mathbb{H}}(X - a)$ be $\mathbb{H}$'s vanishing polynomial.

PROTOGALAXY $(\Phi = (\phi, \boldsymbol{\beta}, e), (\phi_1, \ldots, \phi_k); \omega, (\omega_1, \ldots, \omega_k))$:

1. $\mathbf{V}$ sends a challenge $\delta \in \mathbb{F}$.
2. $\mathbf{P}$ and $\mathbf{V}$ compute $\boldsymbol{\delta} \in \mathbb{F}^t$, where $\boldsymbol{\delta} := (\delta, \delta^2, \ldots, \delta^{2^{t-1}})$.
3. $\mathbf{P}$ computes the polynomial

$$F(X) := \sum_{i \in [n]} \mathsf{pow}_i(\boldsymbol{\beta} + X\boldsymbol{\delta}) f_i(\omega).$$

   (Note that $F(0) = \sum_{i \in [n]} \mathsf{pow}_i(\boldsymbol{\beta}) f_i(\omega) = e$.)
4. $\mathbf{P}$ sends the non-constant coefficients $F_1, \ldots, F_t$ of $F(X)$ to $\mathbf{V}$.
5. $\mathbf{V}$ sends a random challenge $\alpha \in \mathbb{F}$.
6. $\mathbf{P}$ and $\mathbf{V}$ compute $F(\alpha) = e + \sum_{i \in [t]} F_i \alpha^i$.
7. $\mathbf{P}$ and $\mathbf{V}$ compute $\boldsymbol{\beta}^* \in \mathbb{F}^t$ where $\boldsymbol{\beta}^*_i := \boldsymbol{\beta}_i + \alpha \cdot \delta^{2^{i-1}}$.
8. $\mathbf{P}$ computes the polynomial

$$G(X) := \sum_{i \in [n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i(L_0(X)\omega + \sum_{j \in [k]} L_j(X)\omega_j).$$

9. $\mathbf{P}$ computes polynomial $K(X)$ such that

$$G(X) = F(\alpha)L_0(X) + Z(X)K(X).$$

10. $\mathbf{P}$ sends the coefficients of $K(X)$.

11. **V** sends a random challenge $\gamma \in \mathbb{F}$.

12. **P** and **V** compute
$$e^* := F(\alpha)L_0(\gamma) + Z(\gamma)K(\gamma).$$

At the end of the protocol

- **V** outputs the instance $\Phi^* = (\phi^*, \boldsymbol{\beta}^*, e^*)$, where
$$\phi^* := L_0(\gamma)\phi + \sum_{i\in[k]} L_i(\gamma)\phi_i.$$

- **P** outputs the witness $\omega^* := L_0(\gamma)\omega + \sum_{i\in[k]} L_i(\gamma)\omega_i$.

Remark 4.1. *Since* $\sum_{i=0}^{k} L_i(X) = 1$, **V** *can compute* ϕ^* *with* k *rather than* $k+1$ *scalar multiplications as*
$$\phi^* = \phi + \sum_{i\in[k]} L_i(\gamma)(\phi_i - \phi).$$

We make crucial use of the following easily checkable lemma. Loosely speaking, it says any polynomial $f(X)$ "commutes modulu $Z(X)$", with multiplication by Lagrange polynomials.

Lemma 4.2. *Fix any polynomial* $f(X) \in \mathbb{F}[X]$ *and* $a_0, \ldots, a_k \in \mathbb{F}$. *There exists* $Q(X) \in \mathbb{F}[X]$ *such that*
$$f\left(\sum_{i=0}^{k} a_i L_i(X)\right) = \sum_{i=0}^{k} f(a_i)L_i(X) + Z(X)Q(X).$$

Theorem 4.3. *Let* $\mathsf{cm} : \mathbb{F}^M \to \mathbb{G}$ *be an additively homomorphic binding commitment scheme. Let* $\mathcal{R} = \mathcal{R}_{f,\mathsf{cm}}$. *Assume* $dnk = \mathsf{poly}(\lambda)$, *and* $|\mathbb{F}| = \lambda^{\omega(1)}$. *Then* PROTOGALAXY *is an* $(\mathcal{R}^k \mapsto \mathcal{R}^{\mathsf{rand}})$*-accumulation/folding scheme.*

Proof. We prove completeness and soundness, and discuss efficiency in Claims 4.4 and 4.5.

Completeness: Assume $((\phi_1, \ldots, \phi_k), (\omega_1, \ldots, \omega_k)) \in \mathcal{R}^k$ and $((\phi, \beta, e), \omega) \in \mathcal{R}^{\mathsf{rand}}$. We need to show that if the protocol is followed correctly, $(\Phi^*, \omega^*) \in \mathcal{R}^{\mathsf{rand}}$ with probability one . It is immediate from construction that $\mathsf{cm}(\phi^*) = \omega^*$. So, to obtain $(\Phi^*, \omega^*)) \in \mathcal{R}^{\mathsf{rand}}$ it is left to check that
$$\sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i(\omega^*) = e^*.$$

Note first that when $((\phi, \boldsymbol{\beta}, e), \omega) \in \mathcal{R}^{\mathsf{rand}}$
$$F(0) = \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}) f_i(\omega) = e.$$

So in Step 6 $\mathbf{V}$ is computing $F(\alpha)$ correctly.

Therefore, using Lemma 4.2

$$G(X) \mod Z(X) = \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i \left(L_0(X)\omega + \sum_{j\in[k]} L_j(X)\omega_j \right) \mod Z(X)$$

$$= \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) \left(L_0(X) f_i(\omega) + \sum_{j\in[k]} L_j(X) f_i(\omega_j) \right)$$

$$= \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) L_0(X) f_i(\omega) = F(\alpha) L_0(X).$$

The second last equality uses $f_i(\omega_j) = 0$, for all $j \in [k], i \in [n]$.

Therefore, in step 10 $\mathbf{P}$ is indeed able to send $K(X)$ such that

$$G(X) = F(\alpha)L_0(X) + Z(X)K(X).$$

And so, in step 12, we have $e^* = G(\gamma)$. Thus, using the definitions of G and ω^*:

$$e^* = G(\gamma) = \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}_i^*) f_i(L_0(\gamma)\omega + \sum_{j\in[k]} L_j(\gamma)\omega_j) = \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}_i^*) f_i(\omega^*),$$

as required.

Knowledge soundness: Suppose we have a tree of arity $N = \max\{n, dk\}$ of accepting transcripts in the sense that $(\Phi^*, \omega^*) \in \mathcal{R}^{\mathsf{rand}}$; where the verifier input is $(\phi, (\phi_1, \ldots, \phi_k))$. We show that we can either produce a collision of cm or a witness $(\omega, (\omega_1, \ldots, \omega_k))$ such that $((\phi, (\phi_1, \ldots, \phi_k)), (\omega, (\omega_1, \ldots, \omega_k))) \in \mathcal{R}^{\mathsf{rand}} \times \mathcal{R}^k$. According to Lemma 3.2 this suffices.

Let us denote for $1 \le a, b, c \le N$ by $(\Phi^*_{a,b,c}, \omega^*_{a,b,c})$ the final output in the transcript using challenge values $\delta_a, \alpha_b, \gamma_c$. For convenience, let us denote $\phi_0 := \phi, \omega_0 := \omega$. In our first step described next we will either find a collision of cm or find $\omega, \omega_1, \ldots, \omega_k$ such that

- For $i = 0, \ldots, k$, $\phi_i = \mathsf{cm}(\omega_i)$.
- For all $1 \le a, b, c \le N$,

$$\omega^*_{a,b,c} = \sum_{i=0}^{k} L_i(\gamma_{a,b,c})\omega_i.$$

For this purpose, choose any $k+1$ accepting transcripts using distinct values $\gamma_0, \ldots, \gamma_k$ for the challenge γ and let $(\Phi^*_0, \omega^*_0), \ldots, (\Phi^*_k, \omega^*_k))$ be their final outputs and hose transcripts. We have for each $j = 0, \ldots, k$

$$\Phi^*_j = \sum_{i=0}^{k} L_i(\gamma_i)\phi_i.$$

Since the $(k+1)\times(k+1)$ matrix with $L_i(\gamma_j)$ in index (i,j) is invertible, it means there exists for each $j=0,\ldots,k$ coefficients $\{c_{j,\ell}\}$ such that $\phi_j = \sum_{\ell=0}^{k} c_{j,\ell}\Phi^*_\ell$. We define for $j=0,\ldots,k$,

$$\omega_j := \sum_{\ell=0}^{k} c_{j,\ell}\omega^*_\ell$$

From linearity of cm and $\mathsf{cm}(\omega^*_\ell) = \Phi^*_\ell$, we have that $\phi_j = \mathsf{cm}(\omega_j)$ for each $j \in 0,\ldots,k$. We apply this process more times such that each accepting transcript has been used at least once. If at any point, for some j we obtain a *different* vector $\omega'_j \neq \omega_j$ such that $\mathsf{cm}(\omega'_j) = \phi_j$ we output (ω_j, ω'_j) as a collision.

To proceed, we first define several polynomials. For $i \in [n]$ define the polynomial

$$P_i(Y_1, Y_2) := \mathsf{pow}_i(\boldsymbol{\beta} + Y_1 \cdot (Y_2, Y_2^2, \ldots, Y_2^{2^{t-1}})).$$

Note that $\{P_i(Y_1,Y_2)\}_{i\in[n]}$ are linearly independent polynomials - since the degree of Y_2 in P_i is precisely i. For $j \in [k]$, define the polynomial $Q_j(Y_1,Y_2) := \sum_{i\in[n]} P_i(Y_1,Y_2)\cdot f_i(\omega_j)$. We will show Q_j is the zero polynomial for each $j=1,\ldots,k$. As the P_i are linearly independent this implies $f_i(\omega_j)=0$ for all $i\in[n], j\in[k]$; which in turn implies $((\phi_1,\ldots,\phi_k),(\omega_1,\ldots,\omega_k)) \in \mathcal{R}^k$.

For given $1 \le a,b \le N$ we know for $1 \le c \le N$ that

$$\sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta} + \alpha_{a,b}\boldsymbol{\delta}_a) f_i(\omega^*_{a,b,c}) = F_{\alpha_{a,b}} L_0(\gamma_c) + Z_{\alpha,\beta}(\gamma) K(\gamma)$$

Recall that $\omega^*_{a,b,c} = \sum_{j=0}^{k} L_j(\gamma_c)\omega_j$. We thus have as polynomials that

$$\sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta} + \alpha_{a,b}\boldsymbol{\delta}_a) f_i(\sum_{j=0}^{k} L_j(X)\omega_j) = F_{\alpha_{a,b}} L_0(X) + Z(X)K(X)$$

On the other hand, similarly to the completeness case, we have that

$$\sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}+\alpha_{a,b}\boldsymbol{\delta}_a) f_i(\sum_{j=0}^{k} L_j(X)\omega_j) \mod Z(X) \equiv \sum_{j=0}^{k} L_j(X) \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}+\alpha_b\boldsymbol{\delta}_a) f_i(\omega_j)$$

And so, comparing coefficients, we have that $\sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}+\alpha_{a,b}\boldsymbol{\delta}_a) f_i(\omega_0) = F_{a,b}(\alpha_{a,b})$, and for $j\in[k]$ that

$$\sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta} + \alpha_{a,b}\boldsymbol{\delta}_a) f_i(\omega_j) = 0$$

Now, for $j\in[k]$ note that

$$Q_j(\alpha_{a,b}, \delta_a) = \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta} + \alpha_{a,b}\boldsymbol{\delta}_a) f_i(\omega_j) = 0$$

Since this holds for all $1 \le a,b \le N$ we have that $Q_j(Y_1,Y_2)$ is identically zero as we intended to show.

To show we have a valid witness for $\mathcal{R}^{\mathsf{rand}} \times \mathcal{R}^k$ it is left to show that $\sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}) f_i(\omega_0) = e$. Fix some value $1 \le a \le N$, and the challenge $\delta = \delta_a$ sent in the transcripts in the a'th branch of the tree. Denote by F the polynomial $F(X) = e + \sum_{i\in[t]} F_i X^i$, when we let $\{F_i\}$ be the values sent in step 4 in those transcripts. We know for N different values $\alpha_{a,b}$ that

$$\sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta} + \alpha_{a,b}\boldsymbol{\delta}_a) f_i(\omega_0) = F(\alpha_{a,b})$$

So we have a polynomial identity

$$H(X) := \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta} + X\boldsymbol{\delta}_a) f_i(\omega_0) = F(X),$$

which means the constant coefficient of $H(X)$ is e on one hand, but on the other hand $\sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}) f_i(\omega_0)$. In other words,

$$e = \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}) f_i(\omega_0),$$

as required. □

Claim 4.4. *Given $\omega, \boldsymbol{\beta}, \boldsymbol{\delta}$, the polynomial $F(X)$ in step 3 can be computed $O(n)$ $\mathbb{F}$-operations*

Proof. We can expand $F(X)$ as a sum over n terms of degree t. These terms are all products of $t+1$ linear factors, and computing each individually would take $O(nt) = O(n \log n)$ time in general. However, we can exploit the structure of $\mathsf{pow}_i(\boldsymbol{\beta} + X\boldsymbol{\delta})$ to compute the sum in $O(n)$ time.

Consider the full binary tree of n leaves, with each leaf i from left to right labelled by $f_i(\omega)$. Define the *level* of a node to be its distance from the leaves. At each internal, non-leaf node of the tree at level $\ell > 0$, label the left edge from the node with 1 and the right edge with $\boldsymbol{\beta}_\ell + X\boldsymbol{\delta}_\ell$. Given the path from the root to leaf i, if we multiply the labels of each edge along the path and the label at the leaf, we have exactly the i'th term from $F(X)$, $\mathsf{pow}_i(\boldsymbol{\beta} + X\boldsymbol{\delta}) f_i(\omega)$. Now, we can recursively construct the labels of each internal node. For each internal node at level k, let $\mathfrak{n}_l$ and $\mathfrak{n}_r$ be the labels of the left and right children and let e_l and e_r be the labels of the edges connecting the left and right children. Define the label of each internal node to be $\mathfrak{n} := \mathfrak{n}_l e_l + \mathfrak{n}_r e_r$. Since $\deg(e_l) = 0$ and $\deg(e_r) = 1$, it must be the case that $\deg(\mathfrak{n}) = \max(\deg(\mathfrak{n}_l), 1 + \deg(\mathfrak{n}_r)) \le k$. The yields $F(X)$ as the label for the root node.

We want to show that we can compute all the labels in $O(n)$ time. Note that computing each $\mathfrak{n}$ at level ℓ takes $O(\ell)$ field operations, since multiplying by e_r takes $O(\ell)$ operations. There are $2^{t-\ell}$ nodes at level ℓ, so the total number of field operations at level ℓ is $\ell 2^{t-\ell}$. Summing over ℓ from 1 to t, we find the total number of field operations is $2n - t - 1 = O(n)$

$$\sum_{i=1}^{t} i 2^{t-i} = 2^{t-1} \sum_{i=0}^{t-1} (i+1) 2^{-i} = 2^{t-1} \frac{d}{dx} \frac{x^{t+1} - 1}{x - 1}\bigg|_{x=1/2} = 2n - t - 2 = O(n).$$

Since we can compute all the labels in $O(n)$ time, we can compute the root label in $O(n)$ time. Note that this technique can be generalized to trees of arbitrary arity. □

In the following claim we denote by C the maximum over $i \in [n]$ of the number of operations needed to compute $f_i(\omega)$ for a given input $\omega \in \mathbb{F}^n$.

Claim 4.5. *The polynomial $G(X)$ in step 8 can be computed in $O(kd\log(kd)M + ndkC)$ $\mathbb{F}$-operations*

Proof. The degree of $G(X)$ is dk, so given $dk+1$ evaluations of $G(X)$ we can compute its coefficients. However, in the honest prover case we already know k of the evaluations, so we only need $(d-1)k+1$ additional evaluations. We need to evaluate the following inner expression at $(d-1)k+1$ points

$$L_0(X)\omega + \sum_{j\in[k]} L_j(X)\omega_j.$$

This requires $O(k(d-1)\log(k(d-1))M)$ operations. Given these evaluations, we need to evaluate each f_i on these values and sum the results, which takes $O(n(d-1)kC)$ time. Finally, we need to subtract off $L_0(X)e$, divide by $Z(X)$, and then perform an FFT to compute $Q(X)$. All of this takes $O(d(k-1)\log d(k-1))$, which gives total complexity

$$O(k(d-1)\log(k(d-1))M + ndkC).$$

□

5 Alternative construction for large k

In this section we assume familiarity with the sumcheck protocol; for an introduction, see Section 4.1 of Thaler[Tha]. In the protocol of the previous section the folding prover's costs scale quasilinearly with the number of instances k - $O(k\log k)$; and the verifier's cost scales linearly with k. Here, we improve those dependencies to linear and logarithmic respectively, at the cost of a more complex protocol with potentially worse constants. Roughly, we replace the vanishing check mod $Z(X)$ with a sumcheck over a domain of size k. For simplicity, we assume this sumcheck works over the boolean hypercube, but it naturally generalizes to larger bases. Given a collection of $k = 2^s - 1$ instances, folding begins in the same way as before by refreshing the randomness β with δ. More precisely, **P** and **V** will run the protocol of the previous section up to step 7. Now, using the notation there, denote $\boldsymbol{\delta}^n := (\delta^n, \delta^{n\cdot 2}, \ldots, \delta^{n\cdot 2^{s-1}}) \in \mathbb{F}^s$, and $\omega_0 = \omega$; and $e_0 := F(\alpha)$ as computed by **V** in step 6. Note that when **P** is honest, we have

$$\sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i(\omega_0) = e_0.$$

Below we denote by eq the $2s$-variate multilinear polynomial with $\mathsf{eq}(a,b) = 1$ for $a = b \in \{0,1\}^s$ and $\mathsf{eq}(a,b) = 0$ for $a \neq b \in \{0,1\}^s$; and overload notation and denote

by $j \in \{0, \ldots, k\}$ the s-length binary decomposition of j. $\mathbf{P}$ will show the following expression holds:

$$\sum_{j=0}^{k} \mathsf{eq}(j, \boldsymbol{\delta}^n) \sum_{i \in [n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i \left(\sum_{\ell=0}^{k} \mathsf{eq}(j, \ell) \omega_\ell \right) = \mathsf{eq}(0, \boldsymbol{\delta}^n) \cdot e_0.$$

We can think of the lhs of the above identity, as summing the following s-variate polynomial $G(\mathfrak{J})$ over $j \in \{0, \ldots, k\}$:

$$G(\mathfrak{J}) := \mathsf{eq}(\mathfrak{J}, \boldsymbol{\delta}^n) \sum_{i \in [n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i \left(\sum_{\ell=0}^{k} \mathsf{eq}(\mathfrak{J}, \ell) \omega_\ell \right).$$

We now run a sumcheck protocol for G over $\{0,1\}^s \sim \{0, \ldots, k\}$ to check it sums to $\mathsf{eq}(0, \boldsymbol{\delta}^n) \cdot e_0$. For $a \in [s]$, in the a'th round $\mathbf{P}$ sends a univariate polynomial P_a of degree d, and replaces the a'th variable with a verifier challenge γ_a. Following these polynomials, the correctness of the sumcheck is reduced to checking $G(\boldsymbol{\gamma}) = P_s(\gamma_s)$; which is in turn equivalent to showing[5] $((\phi^*, \boldsymbol{\beta}^*, e^*), \omega^*) \in \mathcal{R}^{\mathsf{rand}}$ with

- $\phi^* := \sum_{\ell=0}^{k} \mathsf{eq}(\gamma, \ell) \phi_\ell$, where $\forall \ell \in \{0, \ldots, k\}$, $\phi_\ell := \mathsf{cm}(\omega_\ell)$
- $\omega^* := \sum_{\ell=0}^{k} \mathsf{eq}(\gamma, \ell) \omega_\ell$,
- $e^* := P_s(\gamma_s) / \mathsf{eq}(\boldsymbol{\gamma}, \boldsymbol{\delta}^n)$.

In total the folding verifier needs to evaluate $s = \log k$ hashes and perform $O(\log n + d \log k)$ field operations - the $\log n$ operations coming from "converting" the current accumulator to a new challenge $\boldsymbol{\beta}^*$ as done in the protocol of the previous section via the polynomial $F(X)$. The prover also only sends ds scalars to communicate the $P_a(X)$ polynomials as compared to dk for the univariate PROTOGALAXY protocol. In total the prover requires only $O(kdM + nkdC)$ field operations. The main thing to show is that the univariates sent during sumcheck can be computed efficiently.

Claim 5.1. *The polynomials $P_a(X)$ can all be computed in a total of $O(kdM + nkdC)$ field operations.*

Proof. We show for $i \in [s]$, that computing $P_a(X)$ takes $O(2^{s-a}(kdM + ndC))$ field operations. If this is the case, then it follows that the total computation is $O(kdM + nkdC)$.

We illustrate the claim on P_1. We have

$$P_1(X) = (1 + \delta^n X) \sum_{j \in [k/2]} \mathsf{eq}(j, \boldsymbol{\delta}'^n) \sum_{i \in [n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i \left(\sum_{\ell \in [k/2]} \mathsf{eq}(j, \ell)(X \cdot \omega_{2\ell} + (1 - X)\omega_{2\ell+1}) \right)$$

[5] A formal knowledge soundness proof for this protocol would proceed similarly to that of Theorem 4.3 extracting the $\{\omega_\ell\}$ from the $\{\phi_\ell\}$ and address the case of finding a collision of cm. Since the details are almost identical, we omit them.

First, for each $j \in [k/2]$, we precompute for the length M vector of polynomials $v_i(X) := X \cdot \omega_{2i} + (1-X)\omega_{2i+1}$, d evaluations for each $v_{i,\ell}(X)$. This takes $O(kdM)$ field operations.

To compute $P_1(X)$, we must compute it at d values. For each evaluation, we need to evaluate the outer sum at $k/2$ values $j \in [k/2]$. For each such evaluation, we need to evaluate a sum of length n whose i'th term is

$$\mathsf{pow}_i(\boldsymbol{\beta}^*) f_i(v_i(X)).$$

As we have already computed $d+1$ values of v_i's argument, we can evaluate $f_i(v_i(X))$ at d inputs in $O(dC)$ operations. So in total we get $O(nkdC)$. Since the other P_a's have the same form as P_1 on domains of size going down by two, the claim follows. □

Folding Many Accumulators

This technique transposes into accumulator folding as well. Denote now $k = 2^s$, and let $K := \{0, \ldots, k-1\}$. We fold k accumulators by replacing $\boldsymbol{\beta}$ with a linear combination of the $\boldsymbol{\beta}_j$ values from each instance. Assume we have k instances $\left\{(\phi_j, \boldsymbol{\beta}_j, e_j)\right\}_{j \in K}$ with corresponding witnesses $\{\omega_j\}_{j \in K}$. $\mathbf{V}$ chooses a random $\delta \in \mathbb{F}$, and we set $\boldsymbol{\delta} := (\delta, \delta^2, \ldots, \delta^{2^{s-1}}) \in \mathbb{F}^s$. We look at the polynomial

$$G(\mathfrak{J}) := \mathsf{eq}(\mathfrak{J}, \boldsymbol{\delta}) \sum_{i \in [n]} \mathsf{pow}_i \left(\sum_{\ell \in K} \mathsf{eq}(\mathfrak{J}, \ell) \boldsymbol{\beta}_\ell \right) f_i \left(\sum_{\ell \in K} \mathsf{eq}(\mathfrak{J}, \ell) \omega_\ell \right).$$

When $\mathbf{P}$ is honest, we have

$$\begin{aligned} \sum_{j \in K} G(j) &= \sum_{j \in K} \mathsf{eq}(j, \boldsymbol{\delta}) \sum_{i \in [n]} \mathsf{pow}_i \left(\sum_{\ell \in K} \mathsf{eq}(j, \ell) \boldsymbol{\beta}_\ell \right) f_i \left(\sum_{\ell \in K} \mathsf{eq}(j, \ell) \omega_\ell \right) \\ &= \sum_{j \in K} \mathsf{eq}(j, \boldsymbol{\delta}) \sum_{i \in [n]} \mathsf{pow}_i \left(\boldsymbol{\beta}_j \right) f_i(\omega_j) = \sum_{j \in K} \mathsf{eq}(j, \boldsymbol{\delta}) e_j. \end{aligned}$$

We again run a sumcheck protocol for G over $j \in [k]$, up to and not including the final step of evaluating $G(\boldsymbol{\gamma})$. Denote again by $P_a(X)$, for $a \in [s]$, the univariate polynomials sent in the rounds of the sumcheck. Note that now each P_a has degree at most $d+t$. Following s rounds, the sumcheck verifier needs to check

$$G(\boldsymbol{\gamma}) = \mathsf{eq}(\gamma, \boldsymbol{\delta}) \sum_{i \in [n]} \mathsf{pow}_i \left(\sum_{\ell \in K} \mathsf{eq}(\boldsymbol{\gamma}, \ell) \boldsymbol{\beta}_\ell \right) f_i \left(\sum_{\ell \in K} \mathsf{eq}(\boldsymbol{\gamma}, \ell) \omega_\ell \right).$$

This corresponds to showing $((\phi^*, \boldsymbol{\beta}^*, e^*), \omega^*) \in \mathcal{R}^{\mathsf{rand}}$ where

$$\begin{aligned}
\phi^* &:= \sum_{\ell \in K} \mathsf{eq}(\gamma, \ell)\phi_\ell, \\
\boldsymbol{\beta}^* &= \sum_{\ell \in K} \mathsf{eq}(\boldsymbol{\gamma}, \ell)\boldsymbol{\beta}_\ell, \\
e^* &= \mathsf{eq}(\boldsymbol{\gamma}, \boldsymbol{\delta})^{-1} \cdot P_s(\gamma_s), \\
\omega^* &= \sum_{\ell \in K} \mathsf{eq}(\boldsymbol{\gamma}, \ell)\omega_\ell.
\end{aligned}$$

Thus, we have reduced the k instances of $\mathcal{R}^{\mathsf{rand}}$ to the instance $\Phi^* = (\phi^*, \boldsymbol{\beta}^*, e^*)$, where $\phi^* := \mathsf{cm}(\omega^*)$. The folding verifier can compute $\phi^* = \sum_{\ell \in K} \mathsf{eq}(\boldsymbol{\gamma}, \ell)\phi_\ell$ using k scalar multiplications; as well as $\boldsymbol{\beta}^*$ and e^* using $O((k+d)t)$ field operations.

The main thing to address is the efficiency of computing $\{P_a(X)\}_{a \in [s]}$. Using a combination of the techniques used in Claims 4.4 and 5.1, one can show

Claim 5.2. *The polynomials $P_a(X)$ can all be computed in a total of $O(kdM + nkdC)$ field operations.*

It follows from Claim 5.2 that $\mathbf{P}$ requires in total $O(kdM + nkdC)$ $\mathbb{F}$-operations.

Acknowledgements

We thanks the Benedikt Bünz and Binyi Chen for discussions on Protostar. We thank the Ethereum Foundation and 0xPARC for supporting this work. We thank Cody Gunton, Thor Kampefner, Ivan Mikushin, Pratyush Mishra, Srinath Setty, Zac Williamson and David Wong for corrections and comments.

References

[AFK22] T. Attema, S. Fehr, and M. Klooß. Fiat-shamir transformation of multi-round interactive proofs. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022, Proceedings, Part I*, volume 13747 of *Lecture Notes in Computer Science*, pages 113–142. Springer, 2022.

[BC23] B. Bünz and B. Chen. Protostar: Generic efficient accumulation/folding for special sound protocols. *IACR Cryptol. ePrint Arch.*, page 620, 2023.

[BCL+21] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner. Proof-carrying data without succinct arguments. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021,*

Proceedings, Part I, volume 12825 of *Lecture Notes in Computer Science*, pages 681–710. Springer, 2021.

[BCMS20] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. Recursive proof composition from accumulation schemes. In *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2020.

[CT10] A. Chiesa and E. Tromer. 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*, pages 310–331. Tsinghua University Press, 2010.

[KS22] A. Kothapalli and S. T. V. Setty. Supernova: Proving universal machine executions without universal circuits. *IACR Cryptol. ePrint Arch.*, page 1758, 2022.

[KS23] A. Kothapalli and S. Setty. Hypernova: Recursive arguments for customizable constraint systems. *IACR Cryptol. ePrint Arch.*, page 573, 2023.

[KST21] A. Kothapalli, S. T. V. Setty, and I. Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. *IACR Cryptol. ePrint Arch.*, page 370, 2021.

[Moh] N. Mohnblatt. Sangria: a folding scheme for plonk.

[Tha] Justin Thaler. Proofs, arguments, and zero-knowledge.

[Val08] P. Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.