Interstellar: Efficient GKR-based Folding/IVC Scheme with Collaborative Folding Extension

Jieyi Long

Theta Labs, Inc.
jieyi@thetalabs.org

Abstract. In this paper, we present Interstellar, a novel folding and IVC framework built on a technique we call *circuit interpolation*, designed specifically for circuit satisfiability. By incorporating the GKR protocol, our approach avoids commitments to full computation traces and cross-term vectors, requiring instead only commitments to the actual circuit witness and optionally a small subset of intermediate gate values. This design significantly reduces the size of the vectors to be committed to in each folding step, which is highly advantageous compared to existing schemes, as vector commitments involve costly group multi-scalar multiplications. Moreover, Interstellar is highly flexible. It can be extended to handle high-degree and lookup gates, enable multi-instance folding, and support non-uniform IVC efficiently, making it well-suited for practical applications ranging from zkML to proving program execution for zkVMs. Finally, we introduces a new concept called collaborative folding/IVC which allows multiple provers, each holding a private witness for the same public statement, to jointly perform the folding/IVC computation while preserving witness privacy. This extension makes Interstellar suitable for distributed and privacy-sensitive applications.

Keywords: GKR, Folding, IVC, SNARK, Collaborative Folding/IVC

1 Introduction

Incrementally verifiable computation (IVC) and Proof-Carrying Data (PCD) enable verifiable evaluation of iterative functions [7, 11, 12, 30, 33]. While early IVC/PCD constructions relied on SNARKs, recent works introduced folding schemes as a new primitive [7,8,11,18–20,30–33,40,53], where a folding protocol for a relation $\mathcal{R}$ reduces the verification of two instances to that of a single one.

Although building IVC/PCD from folding is already significantly more efficient than via SNARKs, the prover cost remains considerably high. This overhead primarily stems from two sources. First, existing folding schemes [7,8,11,18–20,30–33,40,53] all rely on constraint systems such as Plonkish [23] and CCS [46], which typically require the prover to generate a vector commitment to the values of all intermediate gate inputs and outputs. This is because the witness for these constraint systems often includes the full computation trace of the circuit, namely, the values of the inputs/outputs of all intermediary gates. Committing

to a large vector is costly, since it involves expensive group Multi-Scalar Multiplication (MSM) operations. Second, many folding protocols require an additional commitment to the cross-terms, typically the same size as the constraint system, further increasing the prover's workload. Coinciding with high prover cost, many practical applications often impose additional privacy constraints. For example, the folding/IVC computation might involve multiple parties, each holding a private witness for the same public statement. For instance, several hospitals may need to jointly aggregate statistics over sensitive patient data. Such a scenario motivates a new notion of *Collaborative Folding/IVC* inspired by collaborative zk-SNARKS [24, 37, 41], where the folding computation is shared among parties while preserving the privacy of their individual witnesses.

Aiming at solving these challenges, in this work, we propose Interstellar, a novel folding and IVC scheme based on a technique we call *circuit interpolation* which is tailored for circuit satisfiability instead of constrain systems. Circuit satisfiability is as expressive as constraint systems such as R1CS and CCS [46] since it is a NP-complete language. Our approach differs from traditional methods by requiring commitments only to the actual witness inputs to the circuit and optionally a subset of intermediate gate values (depending on the circuit flattening parameters; see Section 4.1), thereby eliminating the need to commit to the full computation trace. This efficiency is enabled by the integration of the GKR protocol, which allows the verifier to check the arithmetic circuit satisfiability with the prover committing solely to the witness inputs. Furthermore, our scheme avoids generating cross-term vectors during the folding process, removing the need for any corresponding commitments and further reducing prover overhead. Building on our folding scheme, we develop the Interstellar IVC protocol that capitalizes on the reduced prover costs and enhanced efficiency. We also show that Interstellar can be extended to handle high-degree gates and lookup gates, fold multiple instances at once, and support non-uniform IVC efficiently. Moreover, it can be adapted for collaborative folding and IVC with multiple provers each holding their own secret witness. The key contributions of our proposed schemes include:

- **Prover cost reduction** (Section 4 / 5): A simple yet effective folding scheme that targets circuit satisfiability directly, which eliminates the cross-terms and the need to commit to them. Moreover, the witness size could be much smaller compared to that of the schemes relying on constraint systems, which significantly reduces the prover cost for generating vector commitments.
- **Flexibility** (Section 6): An IVC construction that can be extended to support high-degree and lookup gates, enable multi-instance folding, and handle non-uniform IVC efficiently. All of these makes it well-suited for real-world applications ranging from zkML to proving program execution for zkVMs.
- **Collaborative Folding/IVC** (Section 7): We introduce the concept of collaborative folding/IVC for the first time in the literature, and present the first formal definitions and protocols that enable multiple provers/servers to jointly perform folding/IVC without revealing their private witnesses.

Table 1. A comparison of different folding protocols, where "$\mathbb{G}$ MSM" means group multi-scalar multiplication, "$\mathbb{G}$ ops" means group operations, "CRH" refers to collision resistant hashing, and "$\mathbb{F}$ ops" are field additions and multiplications. $|F|$ denotes the number of variables in the constraint system (i.e. R1CS, Plonkish, CCS) for circuit F, and $|\mathbf{w}|$ refers to the size of the witness $\mathbf{w}$ as defined in Equation 1. Since $|\mathbf{w}| < |F|$ (and for some circuits $|\mathbf{w}| \ll |F|$), compared to all other schemes, our prover time conducts less group computations, espcially the costly group MSMs. The prover of our scheme might require more field operations compared to HyperNova and Protostar when $D^* > D$, but those are much cheaper, espcially when the circuit uses a small field. D denotes the maximum degree of the constraint system used, and q referes to the number of monomials in HyperNova's CCS constraints. D^* and d^* refers to the degree and the number of layers of the normalized circuit F^* as defined in Section 4.1. Note that for the verification times, we list the time to verify a compressed proof (otherwise, they are $O(|F|)$, see also Section 5.2). For Mova and Protostar and Protogalaxy, since they do not explicitly mention the proof size compression scheme, we assume they also use the Spartan-based SNARK as described in Section 6 of the Nova paper for proof compression [33]. Column *Multi-inst* indicates whether the scheme supports folding multiple instances at once. *High-deg* and *Lookup* indicates whether the scheme supports high-degree and lookup gates, respectively. *NIVC* indicates whether the protocol can support non-uniform IVC efficiently. *Collab Folding* refers to whether the work discusses the support for collaborative folding/IVC.

	Prover Time	**Verifier Time**	**Multi-inst**	**High-deg**	**Lookup**	**NIVC**	**Collab Folding**
Nova [33]	2 size-$O(\|F\|)$ $\mathbb{G}$ MSM $O(1)$ $\mathbb{G}$ ops $O(1)$ CRH $O(\|F\|)$ $\mathbb{F}$ ops	$O(\log \|F\|)$ $\mathbb{F}$ ops	No	No	No	No	No
HyperNova [31]	1 size-$O(\|F\|)$ $\mathbb{G}$ MSM $O(1)$ $\mathbb{G}$ ops $O(\log \|F\|)$ CRH $O(q \cdot D \log^2 D \cdot \|F\|)$ $\mathbb{F}$ ops	$O(D \log \|F\|)$ $\mathbb{F}$ ops	**Yes**	**Yes**	**Yes**	**Yes**	No
Mova [18]	1 size-$O(\|F\|)$ $\mathbb{G}$ MSM $O(1)$ $\mathbb{G}$ ops $O(\log \|F\|)$ CRH $O(\|F\|)$ $\mathbb{F}$ ops	$O(\log \|F\|)$ $\mathbb{F}$ ops	No	No	No	No	No
Protostar [11]	1 size-$O(\|F\|)$ $\mathbb{G}$ MSM 2 size-$O(\sqrt{\|F\|})$ $\mathbb{G}$ MSM $O(1)$ $\mathbb{G}$ ops $O(1)$ CRH $O(D\|F\|)$ $\mathbb{F}$ ops	$O(D \log \|F\|)$ $\mathbb{F}$ ops	No	**Yes**	**Yes**	**Yes**	No
Our Work with $\mathsf{PolyCom}_{\mathsf{Dory}}$	1 size-$O(\|\mathbf{w}\|)$ $\mathbb{G}$ MSM $O(1)$ $\mathbb{G}$ ops $O(\log \|F\|)$ or O(1) CRH $O(D^*\|F\|)$ $\mathbb{F}$ ops	$O(d^* \log \|F\|)$ $\mathbb{F}$ ops	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

1.1 Related Work

In this section, we compare our proposed Interstellar IVC protocol with prior folding-based IVC schemes. Notably, none of the existing works address collaborative folding/IVC. To the best of our knowledge, this paper is the first to formalize collaborative folding/IVC and to present concrete constructions.

Nova, SuperNova, HyperNova, and NeutronNova. These are a line of innovative work in folding-based IVC techniques designed to enhance the efficiency and scalability of zero-knowledge proofs. Nova [33] introduced the concept of folding and proposed a folding-based IVC construction. Based on this, SuperNova [30] targets VM-style execution where each computation step has a program counter pc that selects one out of I circuits. SuperNova achieves the so-called "a la carte" cost profile, that is, when proving VM program executions,

the cost of proving a step of a program is proportional only to the size of the circuit representing the instruction invoked by the program step. HyperNova [31] further advanced the framework by supporting high-degree gates using the CCS constraint system and allowing multi-instance folding. NeutronNova [32] represents the latest evolution, focusing on supporting space-efficient zkVMs and multi-instance folding through the Sumcheck protocol. All of these works are based on a constraint system, while Interstellar targets circuit satisfiability directly, which leads to a potentially smaller witness vector to commit to for each folding round.

Sangria, Origami, and Mova. Sangria [40] extends Nova's R1CS-based folding scheme to handle Plonkish constraints. Origami [53] introduced a folding scheme for lookup constraints. Both Sangria and Origami produce cross-terms and incur the corresponding MSM computation cost. Mova [18] eliminates the cross-terms. However, it only handles R1CS constraints without support for high-degree and lookup gates. It also does not support folding multiple instances at once.

Protostar and Protogalaxy. Protostar [11] presented a generic compiler algorithm that can turn any special-sound protocol into a folding scheme. Protostar cannot efficiently fold more than two instances at once, as folding $k > 2$ instances at once blows up the degree of the polynomial involved exponentially in k [19]. To fold $k > 2$ instances, Protostar has to fold them two at a time, resulting in an increase in the size of the verifier circuit by a factor of $O(k)$ compared to folding two instances. Protogalaxy [19] is an extension of Protostar that supports folding $k > 2$ instances at once. Protogalaxy's prover time scales superlinearly with the number of instances folded, i.e., $O(k \log k)$. In comparison, our prover time scales linearly with k.

LatticeFold, LaticeFold+, and Lova. These works focus on constructing lattice-based vector/polynomial commitment schemes and incorporating them into folding-based IVC protocols. LatticeFold [7] improves on Hypernova, and builds a norm-preserving vector commitment scheme based on Ajtai commitment, which keeps the witness norm low by breaking the folding protocol into three steps, i.e. expansion, decomposition, and folding with range proof attached to guarantee extractability. LatticeFold+ [8] improves the efficiency of LatticeFold with a novel algebraic range proof and the double commitment technique. Both LatticeFold and LatticeFold+ rely on structured lattice assumptions, such as MSIS, which binds them to relatively complex arithmetic. Lova [20] is a lattice-based folding scheme that only requires the unstructured SIS assumption, and as a results, their implementation makes only use of arithmetic in hardware-friendly power-of-two moduli. Again, all of these work are based on a constraint system, which leads to a potentially larger witness vector to commit to for each folding round comared to Interstellar. In addition, none of these lattice-based folding schemes explicitly discusses how to support lookup gates.

In summary, Interstellar is overall more well rounded. It supports high-degree and lookup gates, allows folding multiple instances at once, handles non-uniform IVC efficiently, and can be adapted to support collaborative folding/IVC, while

each of the alternative approaches above lacks one feature or another except for HyperNova. However, compared to HyperNova, Interstellar achieves a lower prover cost, as it not only eliminates the cross-terms, but only commits to $\mathbf{w}$, which consists of the actual witness inputs and optionally an additional set of intermediate gates (see Section 4.1). The size of $\mathbf{w}$ could be much smaller than the number of constraints $|F|$ in HyperNova and other frameworks, which leads to significant vector commitment cost savings for the prover. Moreover, HyperNova's prover cost depends superlinearly on the constraint degree ($O(D \log^2 D)$). The cost also depends on q, the number of monomials in its CCS constraints. This could place limitations on handling certain high-degree gates, especially for those whose output has a large number of monomials when expressed as a polynomial of the inputs. LatticeFold, which combines HyperNova's linearization technique with lattice-based commitment schemes, also inherits these shortcomings. Table 1 summarizes the comparison.

2 Preliminaries

This section provides a brief overview of the preliminaries necessary to build the Interstellar framework. We defer the discussion of a few key concepts, such as IP/IOP, polynomial commitment schemes, further details of Sumcheck/GKR and SNARKs, to Appendix A.

2.1 Arithmetic Circuits and Circuit Satisfiability

An arithmetic circuit $\mathbf{y} = F(\mathbf{x}, \mathbf{w})$ over the field $\mathbb{F}$ is a directed *acyclic* graph whose nodes are labeled $+$, $-$, or $*$, computing the addition, subtraction, and multiplication of field elements, respectively, for the values on the incoming wires. The circuit satisfiability problem can be defined as follows:

Definition 1 (Circuit Satisfiability). *Let $F : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^m$ be an arithmetic circuit over field $\mathbb{F}$. The circuit takes in a public input $\mathbf{x} \in \mathbb{F}^n$, a private witness $\mathbf{w} \in \mathbb{F}^k$, and outputs $\mathbf{y} \in \mathbb{F}^m$. Given $(F, \mathbf{x}, \mathbf{y})$, The circuit satisfiability problem asks a witness $\mathbf{w} \in \mathbb{F}^k$ such that*

$$F(\mathbf{x}, \mathbf{w}) = \mathbf{y}$$

If such a $\mathbf{w}$ exists, we say that the instance $(F, \mathbf{x}, \mathbf{y})$ is satisfiable.

It is well known that the circuit satisfiability problem is NP-complete. Consequently, any instance of an NP relation can be efficiently reduced to an instance of circuit satisfiability. A circuit is layered if it can be partitioned into subsets called layers such that every wire in the circuit is between adjacent layers. We call the number of layers the depth of the circuit, which is denoted by d. Figure 1 depicts two small layered circuits where the left circuit only has public inputs, while the right one takes in both the public inputs and the private witness.

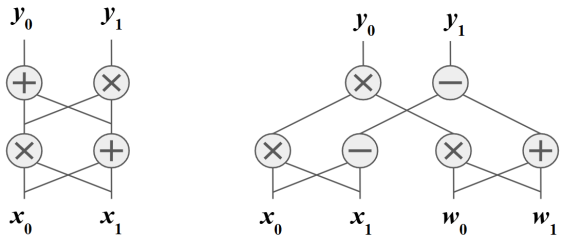


Fig. 1. Two layered circuit examples, where the left one only takes in the public input vector $\mathbf{x}$, while the right one takes in both the public input vector $\mathbf{x}$ and witness $\mathbf{w}$. The right circuit also uses subtraction gates.

Note that since the circuit only contains addition, subtraction, and multiplication gates, each output can be expressed as a multivariate polynomial in terms of the input variables. For example, for the left circuit in Figure 1, the two outputs can be expressed as $y_0 = x_0x_1 + x_0 + x_1$ and $y_1 = x_0^2x_1 + x_0x_1^2$, respectively. We use the letter D to denote the *total degree* of the circuit, which is the maximum total degree across all outputs expressed as a polynomial in terms of input variables. In this circuit, the degree of output y_0 is 2, and that of y_1 is 3. Hence, the total degree of the circuit is 3.

2.2 The GKR Protocol

The GKR protocol and its variants are important gradients for developing the Interstellar protocol framework. It is an example of interactive protocols defined in the last section. The GKR protocol was proposed in the seminal work by Goldwasser et al. [25] as an interactive protocol for proving the correctness of layered circuit evaluation. The GKR protocol uses the Sumcheck protocol as a core building block. In the very beginning, the prover and the verifier agree on a layered circuit F composed of addition, subtraction, and multiplication gates with two inputs. In the first message, the prover sends the claimed output of the circuit to the verifier. The verifier cannot verify this claim herself. However, she can work with the prover to reduce this claim to a claim about the previous layer. Then, the protocol processes the circuit layer by layer until the input layer, where the verifier has sufficient information to verify the final claim. In more detail, in the i^{th} iteration, the GKR protocol invokes the Sumcheck protocol to reduce a claim about the *multilinear extension* (see Section A.2) of the gate values at layer i to that of the gate values at layer $i + 1$. Finally, at the input layer, the verifier has all the data needed to derive the multilinear extension of the input vector on her own. Therefore, the verifier can check the final claim herself, which in turn proves or disproves the correctness of the circuit evaluation. In terms of protocol complexity, the latest variant of the GKR protocol can achieve $O(|F|)$ prover time [51], $O(|\mathbf{io}|+d\log|F|)$ verifier time, and $O(d\log|F|)$ proof size, where $|F|$, $|\mathbf{io}|$, and d are the size of the circuit, the size of the inputs/outputs, and

the depth of the circuit, respectively [48]. The soundness error of the protocol is $O(\frac{d \log |F|}{|\mathbb{F}|})$ [48, 51]. We will discuss further details of the GKR protocol in Appendix A.6.

2.3 Incrementally Verifiable Computation

Incrementally verifiable computation (IVC) enables verifiable evaluation of a function applied repeatedly [5, 49]. Given a function implemented by arithmetic circuit F and an initial input $\mathbf{z}_0$, an IVC scheme allows the prover to incrementally generate a proof Π_i asserting that $\mathbf{z}_i = F^{(i)}(\mathbf{z}_0)$, i.e., the result of applying F to $\mathbf{z}_0$ for i iterations. The proof Π_i is derived from the previous proof Π_{i-1}, which attests to the validity of $\mathbf{z}_{i-1} = F^{(i-1)}(\mathbf{z}_0)$. IVC also allows circuit F to take in a non-deterministic input $\boldsymbol{\omega}_{i-1}$ for the i^{th} incremental step, i.e. $\mathbf{z}_i = F(\mathbf{z}_{i-1}, \boldsymbol{\omega}_{i-1})$. IVC can be formally defined as (adapted from [33]):

Definition 2 (IVC). *Given arithmetic circuit F executing incremental computation $\mathbf{z}_i = F(\mathbf{z}_{i-1}, \boldsymbol{\omega}_{i-1})$, an incrementally verifiable computation (IVC) scheme is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and a deterministic $\mathcal{K}$ denoting the generator, the prover, the verifier, and the encoder respectively. An IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ satisfies* perfect completeness *if for any PPT adversary $\mathcal{A}$,*

$$\Pr\left[\mathcal{V}(\mathsf{vk}, i, \mathbf{z}_0, \mathbf{z}_i, \Pi_i) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ F, (i, \mathbf{z}_0, \mathbf{z}_i, \mathbf{z}_{i-1}, \boldsymbol{\omega}_{i-1}, \Pi_{i-1}) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F), \\ \mathbf{z}_i = F(\mathbf{z}_{i-1}, \boldsymbol{\omega}_{i-1}), \\ \mathcal{V}(\mathsf{vk}, i-1, \mathbf{z}_0, \mathbf{z}_{i-1}, \Pi_{i-1}) = 1, \\ \Pi_i \leftarrow \mathcal{P}(\mathsf{pk}, i, \mathbf{z}_0, \mathbf{z}_i, \mathbf{z}_{i-1}, \boldsymbol{\omega}_{i-1}, \Pi_{i-1}) \end{array}\right] = 1$$

Furthermore, an IVC scheme is said to satisfy knowledge soundness if, for any constant $n \in \mathbb{N}$ and for all expected polynomial-time adversaries $\mathcal{P}^*$, there exists an expected polynomial-time extractor $\mathcal{E}$ such that

$$\Pr_{\mathsf{r}}\left[\begin{array}{l} \mathbf{z}_n = \mathbf{z} \text{ where} \\ \mathbf{z}_{i+1} \leftarrow F(\mathbf{z}_i, \boldsymbol{\omega}_i) \\ \forall i \in \{0, \ldots, n-1\} \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (F, (\mathbf{z}_0, \mathbf{z}), \Pi) \leftarrow \mathcal{P}^*(\mathsf{pp}, r), \\ (\boldsymbol{\omega}_0, \ldots, \boldsymbol{\omega}_{n-1}) \leftarrow \mathcal{E}(\mathsf{pp}, r) \end{array}\right] \approx$$

$$\Pr_{\mathsf{r}}\left[\mathcal{V}(\mathsf{vk}, (n, \mathbf{z}_0, \mathbf{z}), \Pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (F, (\mathbf{z}_0, \mathbf{z}), \Pi) \leftarrow \mathcal{P}^*(\mathsf{pp}, r), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F) \end{array}\right]$$

where r denotes an arbitrarily long random tape. An IVC scheme is succinct if the size of the IVC proof Π does not grow with n.

2.4 Folding Schemes

IVC can be constructed from folding schemes (a.k.a. split accumulation) which were originally introduced in Halo [9] and further developed by Kothapalli et

al. [33] and Bunz et al. [12]. Later, the technique was generalized to a framework called reduction of knowledge [29], in which a prover and a verifier run an interactive protocol to reduce checking membership of an instance in a relation to checking membership of a related instance in a usually simpler relation. In particular, a folding scheme for a relation $\mathcal{R}$ is a protocol that transforms the problem of verifying two instances in $\mathcal{R}$ into the problem of verifying a single instance in $\mathcal{R}$ (adapted from [33]):

Definition 3 (Folding Scheme). *A folding scheme for relation $\mathcal{R}$ consists of a PPT generator algorithm $\mathcal{G}$, a deterministic encoder algorithm $\mathcal{K}$, and a pair of PPT algorithms prover $\mathcal{P}$ and verifier $\mathcal{V}$ respectively, defined as:*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$*: On input security parameter λ, samples public parameters* pp.
- $\mathcal{K}(\mathsf{pp}, s) \to (\mathsf{pk}, \mathsf{vk})$*: On input* pp*, and a common structure s between instances to be folded, outputs a prover key* pk *and a verifier key* vk.
- $\mathcal{P}(\mathsf{pk}, (\mathsf{u}_0, \mathbf{w}_0), (\mathsf{u}_1, \mathbf{w}_1)) \to (\mathsf{u}, \mathbf{w})$*: On input instance-witness tuples* $(\mathsf{u}_0, \mathbf{w}_0)$ *and* $(\mathsf{u}_1, \mathbf{w}_1)$*, outputs a new instance-witness tuple* $(\mathsf{u}, \mathbf{w})$ *of the same size.*
- $\mathcal{V}(\mathsf{vk}, \mathsf{u}_0, \mathsf{u}_1) \to \mathsf{u}$*: On input instances* u_0 *and* u_1*, outputs a new instance* u.

Let $(\mathsf{u}, \mathbf{w}) \leftarrow \langle \mathcal{P}(\mathsf{pk}, \mathbf{w}_0, \mathbf{w}_1),\ \mathcal{V}(\mathsf{vk})(\mathsf{u}_0, \mathsf{u}_1) \rangle$ *denote the verifier's output instance* u *and the prover's output witness* $\mathbf{w}$ *from the interaction of $\mathcal{P}$ and $\mathcal{V}$ on witnesses* $(\mathbf{w}_0, \mathbf{w}_1)$*, prover key* pk*, verifier key* vk *and instances* $(\mathsf{u}_0, \mathsf{u}_1)$*. A folding scheme satisfies perfect completeness if for all PPT adversaries $\mathcal{A}$*

$$\Pr\left[(\mathsf{pp}, s, \mathsf{u}, \mathbf{w}) \in \mathcal{R} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (s, (\mathsf{u}_0, \mathbf{w}_0), (\mathsf{u}_1, \mathbf{w}_1)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, s, \mathsf{u}_0, \mathbf{w}_0), (\mathsf{pp}, s, \mathsf{u}_1, \mathbf{w}_1) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, s), \\ (\mathsf{u}, \mathbf{w}) \leftarrow \langle \mathcal{P}(\mathsf{pk}, \mathbf{w}_0, \mathbf{w}_1),\ \mathcal{V}(\mathsf{vk}) \rangle (\mathsf{u}_0, \mathsf{u}_1) \end{array} \right] = 1$$

A folding scheme satisfies knowledge soundness if for any expected polynomial-time adversary $\mathcal{P}^*$ there is an expected polynomial-time extractor $\mathcal{E}$ such that

$$\Pr\left[\begin{array}{l} (\mathsf{pp}, s, \mathsf{u}_0, \mathbf{w}_0) \in \mathcal{R}, \\ (\mathsf{pp}, s, \mathsf{u}_1, \mathbf{w}_1) \in \mathcal{R} \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (s, (\mathsf{u}_0, \mathsf{u}_1)) \leftarrow \mathcal{P}^*(\mathsf{pp}, \rho), \\ (\mathbf{w}_0, \mathbf{w}_1) \leftarrow \mathcal{E}(\mathsf{pp}, \rho) \end{array} \right] \geq$$

$$\Pr\left[(\mathsf{pp}, s, \mathsf{u}, \mathbf{w}) \in \mathcal{R} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (s, (\mathsf{u}_0, \mathsf{u}_1)) \leftarrow \mathcal{P}^*(\mathsf{pp}, \rho), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, s), \\ (\mathsf{u}, \mathbf{w}) \leftarrow \langle \mathcal{P}^*(\mathsf{pk}, \rho),\ \mathcal{V}(\mathsf{vk}) \rangle (\mathsf{u}_0, \mathsf{u}_1) \end{array} \right] - \mathsf{negl}(\lambda)$$

where ρ denotes arbitrary input randomness for $\mathcal{P}^*$. A folding scheme is considered non-trivial if the communication overhead and the computational effort of the verifier $\mathcal{V}$ are lower when $\mathcal{V}$ verifies a single folded instance with a witness provided by $\mathcal{P}$, compared to verifying separate witnesses for each of the original instances individually.

3 Technical Overview

3.1 Common Notations

In the remainder of this paper, we will use $|\cdot|$ to denote the size/length of an object. For example, $|F|$ refers to the size of the circuit F, that is, the total number of gates in the circuit. $|\mathbf{io}|$ represents the total length of the public input and output vectors. $|\mathbf{w}|$ denotes the length of the witness vector. d represents the depth of the circuit. D represents the maximum total degree of the circuit (see Section 2.1). For a function defined over a Boolean hypercube, we will use the "tilde" symbol over the function to represent its multilinear extension, e.g. $\tilde{\mathbf{w}}(r)$. We will use λ to denote the security parameter, and $\mathsf{negl}(\lambda)$ to denote a negligible function in λ. "PPT" stands for probabilistic polynomial time.

3.2 A Motivating Example

Our main goal is to design an efficient folding scheme to reduce the prover cost through a technique we call *circuit interpolation*. As a motivating example, we use a simple arithmetic circuit to illustrate the basic ideas. Assume $\mathbb{F}$ is a large prime field, and we are given a circuit F which only takes in a witness vector $\mathbf{w} \in \mathbb{F}^m$ as its inputs and has a single output $y \in \mathbb{F}$, i.e. $y = F(\mathbf{w})$. *Single output* here means the output is a single field element as opposed to being a vector. Now, consider two instances $(\mathbf{w}_0, y_0)$ and $(\mathbf{w}_1, y_1)$. The prover creates an interpolation of the two witness vectors $\mathbf{w}(t) = (1-t)\cdot\mathbf{w}_0 + t\cdot\mathbf{w}_1$, where $t \in \mathbb{F}$ is a variable. Then, since F is an arithmetic circuit, $F(\mathbf{w}(t))$ can be expressed as a univariate polynomial of t. Let us assume that the circuit F has a degree D in terms of its inputs, as defined earlier. Since $\mathbf{w}(t)$ is a linear function of t, their composition $y(t) = F(\mathbf{w}(t)) = F((1-t)\cdot\mathbf{w}_0 + t\cdot\mathbf{w}_1)$ is a D-degree univariate polynomial of t. The prover sends $y(t)$ to the verifer [1]. The verifier first checks whether $y(0) = y_0$ and $y(1) = y_1$ both holds. Then, she randomly samples $\gamma \in \mathbb{F}$ and sends it to the prover. They can then independently create a randomly "interpolated" circuit instance $(\mathbf{w}_\gamma, y_\gamma)$ where $\mathbf{w}_\gamma = (1-\gamma)\cdot\mathbf{w}_0 + \gamma\cdot\mathbf{w}_1$ and $y_\gamma = y(\gamma)$. For completeness, it is obvious that $y_\gamma = F(\mathbf{w}_\gamma)$ if $y(\gamma)$ is honestly computed. For soundness, suppose the prover sends another $y'(t)$ instead, by the Schwartz-Zippel Lemma [44, 56], for a random $\gamma \in \mathbb{F}$, the probability that $y(\gamma) = y'(\gamma)$ is at most $D/|\mathbb{F}|$. As the verifier checks $y_\gamma = F(\mathbf{w}_\gamma)$, she can be convinced with high probability that what she received is the true $y(t) = F(\mathbf{w}(t))$. Moreover, the verifier already checks $y(0) = y_0$. Meanwhile, through homomorphic commitments discussed later in this paragraph, we can enforce that $\mathbf{w}(0) = \mathbf{w}_0$. Hence, we have $F(\mathbf{w}_0) = F(\mathbf{w}(0)) = y(0) = y_0$ must

[1] Alternatively, the prover can just sends $\{y(2), y(3), .., y(D)\}$, i.e. the evaluation of $y(t)$ at $2, 3, .., D$ to the verifier. Note that the prover does not need to send $y(0)$ and $y(1)$, since by construction $y(0) = y_0$ and $y(1) = y_1$, so the verifier already has them. Then, the verifer can evaluate $y(t)$ at any point $\gamma \in \mathbb{F}$ via barycentric interpolation. Since both the inputs and the output of the folded circuit instane can be obtained via polynomial interpolation, we name our technique *circuit interpolation*.

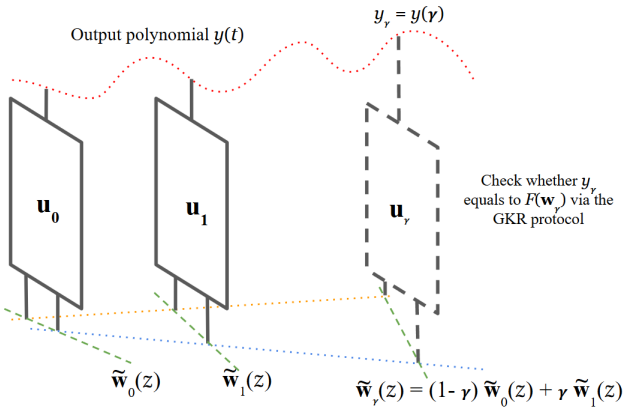


Fig. 2. Intuition of the *circuit interpolation* technique to fold two instances $\mathsf{u}_0 = ((com_{\mathbf{w}0}, y_0), \mathbf{w}0)$ and $\mathsf{u}_1 = ((com_{\mathbf{w}1}, y_1), \mathbf{w}1)$ into a single random instance $\mathsf{u}_\gamma = ((com_{\mathbf{w}r}, y_\gamma), \mathbf{w}r)$. The MLE of the witness to the random instance $\tilde{\mathbf{w}}_\gamma(z)$ is precisely the interpolation of the MLE of $\tilde{\mathbf{w}}_0(z)$ and $\tilde{\mathbf{w}}_1(z)$, i.e. $\tilde{\mathbf{w}}_\gamma(z) = (1-\gamma)\cdot\tilde{\mathbf{w}}_0(z)+\gamma\cdot\tilde{\mathbf{w}}_1(z)$. The prover and verifier can verify the folded instance and hence the original two instances by running the GKR protocol on the folded instance.

hold. The same argument applies to $\mathbf{w}_1$. Hence, both $(\mathbf{w}_0, y_0)$ and $(\mathbf{w}_1, y_1)$ are satisfying instances of circuit F. Circling back to checking if $y_\gamma = F(\mathbf{w}_\gamma)$, there is an issue that the verifier does not know $\mathbf{w}_\gamma$, since it is derived from private witnesses $\mathbf{w}_0$ and $\mathbf{w}_1$. To address this, we note that the multilinear extension of $\mathbf{w}_0$ and $\mathbf{w}_1$ can be written as $\tilde{\mathbf{w}}_0(z) = \sum_{i=0}^{|\mathbf{w}|-1} \tilde{\mathsf{eq}}(z,i)\cdot w_{0i}$ and $\tilde{\mathbf{w}}_1(z) = \sum_{i=0}^{|\mathbf{w}|-1} \tilde{\mathsf{eq}}(z,i)\cdot w_{1i}$, respectively. Their random linear combination $\tilde{\mathbf{w}}_\gamma(z) = (1-\gamma)\cdot\tilde{\mathbf{w}}_0(z) + \gamma\cdot\tilde{\mathbf{w}}_1(z) = \sum_{i=0}^{|\mathbf{w}|-1} \tilde{\mathsf{eq}}(z,i)\cdot((1-\gamma)\cdot w_{0i} + \gamma\cdot w_{1i})$ is precisely the multilinear extension of $\mathbf{w}_\gamma$. Thus, if in the beginning, the prover sends the verifier $com_{\mathbf{w}0}$ and $com_{\mathbf{w}1}$, the *homomorphic* polynomial commitments to $\tilde{\mathbf{w}}_0(z)$ and $\tilde{\mathbf{w}}_1(z)$, the verifier can independently compute the polynomial commitment $com_{\mathbf{w}r}$ to $\tilde{\mathbf{w}}_\gamma(z)$ by $com_{\mathbf{w}r} = (1-\gamma)\cdot com_{\mathbf{w}0} + \gamma\cdot com_{\mathbf{w}1}$. This way the prover and verifier jointly fold instance $((com_{\mathbf{w}0}, y_0), \mathbf{w}_0)$ and $((com_{\mathbf{w}1}, y_1), \mathbf{w}_1)$ into a single instance $((com_{\mathbf{w}\gamma}, y_\gamma), \mathbf{w}_\gamma)$. Finally, to verify the folded instance, the prover and verifier can run the GKR protocol to jointly prove whether $y_\gamma = F(\mathbf{w}_\gamma)$. With $com_{\mathbf{w}r}$ the verifier can check in the last GKR step if the prover sends her the correct values throughout the protocol execution. The homomorphism and the binding property of the commitment scheme also ensure $\mathbf{w}_\gamma$ is the correct random linear combination of $\mathbf{w}_0$ and $\mathbf{w}_1$ (i.e. enforcing $\mathbf{w}(0) = \mathbf{w}_0$ and $\mathbf{w}(1) = \mathbf{w}_1$). Figure 2 demonstrates *circuit interpolation* technique.

To see why this approach can significantly reduce the prover cost, it is worth noting that this process does not introduce cross-terms as in Nova and other folding works [30, 33, 40, 53]. As a result, the prover does not need to commit to these cross-terms, which is approximately the same size as the circuit and is one of the main contributors to the prover cost in Nova [33]. Secondly, most existing folding methods commit to the entire trace of the circuit, that is, the values of *all* intermidate gate outputs/inputs in addition to the actual witness

$\mathbf{w}$. As illustrated in the example above, by incorporating the GKR protocol, our framework requires only the prover to commit to $\mathbf{w}$, which could have a much smaller size than the entire trace for certain types of circuits.

3.3 Overview of the Interstellar Protocol Framework

We designed the Interstellar protocol framework to expand the core ideas described above into a full-fledged folding/IVC framework (Section 4 / 5). It is a folding scheme and IVC compiler that allow committing only to the input witness, and verify the proof via a GKR backend. The proof can be compressed with SNARK composition (Section 5.2). We then extend it to support high-degree gates, lookup gates, allow folding multiple instances, and handle non-uniform IVC efficiently (Section 6). We also enhance it with MPC such that multiple provers each having access to a portion of the witnesses can jointly fold the instances together and generate IVC proofs (Section 7). Below we provide an overview of how these can be achieved.

Circuit Normalization. Given a possibly deep and multi-output circuit F which performs incremental computation $\mathbf{z}_{i+1} = F(\mathbf{z}_i, \boldsymbol{\omega}_i)$, we first normalize it into an equivalent shallow circuit with a single output. This compressing to one output lets folding verify a single output constraint instead of many. It also reduces the prover overhead since shallower circuits tend to have a lower degree. The circuit normalization has three main steps. The first step is **circuit augmentation** which augments F to circuit F' in order to perform additional bookkeeping to fold proofs of prior invocations of itself. This is done by adding the verifier circuit of the non-interactive folding scheme to F similar to that described in Section 5.1 of the Nova paper [33]. The next step is **circuit flattening**. Assume the augmented circuit F' has d layers. We can split these layers into k blocks, each with a depth of at most $\lceil d/k \rceil$. Here k is a design parameter which trades off the circuit depth with the witness size. If d is not too large, we can simply set $k = 1$. The process of flattening F' into a circuit $\mathbf{y} = \bar{F}(\mathbf{x}, \mathbf{w})$ is depicted in Figure 3 and formally defined in Formula (1). Note that extra gates and wires are also added to the circuit, such that if the correct witness is provided, the output vector $\mathbf{y}$ of $\bar{F}$ should be a zero vector. The third step is **single output reduction**, which transforms $\bar{F}$ into F^* with a single output $y \in \mathbb{F}$. The reduction uses the random linear combination trick. If $\bar{F}$ has n output $\mathbf{y} = (y_0, y_1, ..., y_{n-1})$, we can sample $\log n$ independent random values $\boldsymbol{\beta} = (\beta_0, .., \beta_k, .., \beta_{\log n - 1}) \in \mathbb{F}^{\log n}$ and use a circuit with at most $\log n$ layers to expand them into a random vector $\mathbf{s} = \{s_i = \prod_{k=0}^{\log n - 1} (i_k \cdot \beta_k + (1 - i_k) \cdot (1 - \beta_k)) | i = 1, .., n\}$. It can be proved that $\mathbf{s} \cdot \mathbf{y} = 0$ guarantees that $\mathbf{y}$ is a zero vector with high probability. By treating $\boldsymbol{\beta}$ as a public input vector to the circuit, we finally convert the original circuit F into an equivalent shallow circuit with a single output, namely $y = F^*(\boldsymbol{\beta}, \mathbf{x}, \mathbf{w})$ where $y = 0$ indicates $\mathbf{z}_{i+1} = F(\mathbf{z}_i, \boldsymbol{\omega}_i)$ holds for the original circuit with high probability. Then, the prover and the verifier engage in an interactive protocol to fold the two instances of the normalized circuit F^* into a single instance.

The Folding and IVC Scheme. For circuit $y = F^*(\boldsymbol{\beta}, \mathbf{x}, \mathbf{w})$ resulting from the circuit normalization process described above, we call $(\boldsymbol{\beta}, \mathbf{x}, com_{\mathbf{w}}, y)$ a commit-

ted instance for F^*. An instance $(\boldsymbol{\beta}, \mathbf{x}, com_{\mathbf{w}}, y)$ is satisfied by a witness $(\mathbf{w}, r_{\mathbf{w}})$ where $r_{\mathbf{w}} \in \mathbb{F}$, if $y = F^*(\boldsymbol{\beta}, \mathbf{x}, \mathbf{w})$ and $com_{\mathbf{w}} = \mathsf{PolyCom}(\mathsf{pp}_{\mathbf{w}}, \mathbf{w}, r_{\mathbf{w}})$, namely $com_{\mathbf{w}}$ is a homomorphic polynomial commitments to $\tilde{\mathbf{w}}(z)$, the multilinear extension of $\mathbf{w}$. Given two satisfying instances $(\boldsymbol{\beta}_0, \mathbf{x}_0, \mathbf{w}_0, y_0)$ and $(\boldsymbol{\beta}_1, \mathbf{x}_1, \mathbf{w}_1, y_1)$, the goal of folding is to reduce the validity checks for them into that of a third instance. The core idea is to define the following interpolation $\boldsymbol{\beta}(t) = (1-t)\boldsymbol{\beta}_0 + t\boldsymbol{\beta}_1$, $\mathbf{x}(t) = (1-t)\mathbf{x}_0 + t\mathbf{x}_1$, $\mathbf{w}(t) = (1-t)\mathbf{w}_0 + t\mathbf{w}_1$, $\mathbf{y}(t) = (1-t)\mathbf{y}_0 + t\mathbf{y}_1$, and then derive the univariate output polynomial $y(t) = F^*(\boldsymbol{\beta}(t), \mathbf{x}(t), \mathbf{w}(t))$. As in the motivating example, the folding process interpolates a third circuit instance $F^*(\boldsymbol{\beta}(\gamma), \mathbf{x}(\gamma), \mathbf{w}(\gamma))$ induced by a value $\gamma \in \mathbb{F}$ randomly sampled by the verifier. The validity checks for this third instance essentially boils down to checking if the prover can respond with y_γ such that $y_\gamma = F^*(\boldsymbol{\beta}(\gamma), \mathbf{x}(\gamma), \mathbf{w}(\gamma))$ holds. The soundness of this protocol is derived from the Schwartz-Zippel Lemma, as explained in the motivating example. To complete the folding process, both the prover and verifier also need to generate $com_{\mathbf{w}\gamma}$, the commitment to $\mathbf{w}(\gamma)$. If we use a homomorphic commitment scheme, both the prover and verifier can independently derive this new commitment from $com_{\mathbf{w}0}$ and $com_{\mathbf{w}1}$ via random linear combination. The dominant group operations per fold is therefore just a single MSM of size $|\mathbf{w}|$, without needing to generate the cross-terms or full-trace commitment as in many other folding frameworks. The interactive folding protocol is also a public coin protocol, and therefore can be made non-interactive via the Fiat-Shamir transformation. The folding protocols are presented in Construction 12 and Construction 14.

With the non-interactive folding protocol, we can construct the IVC scheme which maintains an accumulated instance U_i for the i^{th} folding step, and folds the step instance u_i into it (see Construction 15). We prove in Theorem 16 that the proposed IVC scheme for the circuit F satisfy the completeness and knowledge soundness property. In terms of efficiency, for each folding step, the prover performs $O(\log |F|)$ hashes and $O(D^*|F|)$ field operations. On top of that, when instantiating $\mathsf{PolyCom}$ with the either the Bulletproofs or Dory commitment scheme, the prover performs a single group scalar multiplication and a single group MSM of size $|\mathbf{w}|$, where $\mathbf{w}$ is the witness to circuit F^*. The IVC proof generated by Construction 15 is linear in $|F|$. As F^* can be viewed as a layered circuit, to reduce the proof size, we propose to employ GKR-based SNARK protocols such as Libra to further compress the IVC proof. As stated in Theorem 18, the SNARK proof generation runs in time $O_\lambda(|F|)$, and the generated proof size can be reduced to $O_\lambda(d^* \log |F|)$.

Folding/IVC Scheme Extensions. Interstellar+ extends the basic folding/IVC protocol to incorporate high-degree gates and lookup gates, support folding multiple instances at once, and handle non-uniform IVC:

Support for high-degree gates (Section 6.1): Our protocol handles high-degree gates by simply unrolling them into sequences of standard multiplication and addition gates. For example, a gate $y = x^4$ can be expressed as $z = x^2$ followed by $y = z^2$. Unlike R1CS-based systems, where such unrolling increases the number of constraints, and therefore the size of the witness and cross-term

commitments, unrolling in our framework incurs almost no additional prover cost. In particular, if the circuit flattening step avoids cutting through unrolled gates, no extra witness elements are introduced, and so the prover costs for the folding process remain unchanged.

Handling lookup gates (Section 6.2): It is well established that proving the inputs and outputs of certain gates satisfy a lookup relationship is equivalent to proving that a sequence of values is contained within a predefined lookup table, where the sequence of values are derived from a subset of gate values in the circuit [2, 27, 47]. Thus, to handle lookup gates we can leverage a lemma from [27] which states that given a lookup table $\mathbf{p} = [p_i]_{i=0}^{P-1}$, we have sequence $\{v_i | i = 0, 1, .., l-1\} \subset \mathbf{p}$ as if and only if there exists a sequence $[m_i]_{i=0}^{P-1}$ of field elements such that $\sum_{i=0}^{l-1} \frac{1}{X+v_i} = \sum_{i=0}^{P-1} \frac{m_i}{X+p_i}$. In order to prove this condition can be met, we augment F^* with a set of auxiliary public inputs such as $\{m_i | i = 0, 1, .., P-1\}$. We also add circuits to perform the additional checks for these inputs which essentially equivalent to check if $\sum_{i=0}^{l-1} \frac{1}{X+v_i} = \sum_{i=0}^{P-1} \frac{m_i}{X+p_i}$ holds for the gate values and the additional inputs. Denoting the resulting circuit by G^*, we can then invoke the folding and IVC protocol for G^* which prove whether the original lookup relationship is indeed satisfied.

Fold multiple instances at once (Section 6.3): The basic folding scheme can be naturally extended to fold multiple instances at once, which enables our scheme to support PCD. The core idea is to generalize the linear interpolation $\mathbf{w}(t) = (1-t)\mathbf{w}_0 + t\mathbf{w}_1$ to $\mathbf{w}(t) = \sum_{j=0}^{k-1} \delta(t, j) \cdot \mathbf{w}_j$ where $\delta(t, j)$ is the Lagrange basis polynomials, and likewise for $\boldsymbol{\beta}$, $\mathbf{x}$, and $\mathbf{y}$. The same also applies for $com_{\mathbf{w}}$, thanks to its homomorphic property. Then, the folding and IVC protocol proceed similar to the basic two instances case. In terms of efficiency, as stated by Theorem 23, the prover performs $O(\log |F|)$ hashes and $O(kD^*|F|)$ field operations. The size of the folded instance has a constant of size $O_\lambda(|F|)$. On top of that, if we instantiate PolyCom with the either the Bulletproofs or Dory commitment scheme, the prover performs k size-$|\mathbf{w}|$ group MSMs and k group scalar-multiplications and additions.

Support non-uniform IVC (Appendix B.5): Non-uniform IVC aims at efficiently producing succinct proofs of correct execution of programs in a VM where only a small part of circuit is activated for each VM instruction [11, 30]. The cost for IVC proof generation should be proportional only to the size of the sub-circuit representing the instruction invoked by the program step. Assume that the VM has I subcircuit, each implementing a VM instruction, we vectorize the state to $(\mathbf{B}, \mathbf{X}, com_{\mathbf{W}}, \mathbf{Y})$ and $(\mathbf{W}, r_{\mathbf{W}})$, where $\mathbf{B} = (\boldsymbol{\beta}^{(0)}, .., \boldsymbol{\beta}^{(I-1)})$, and the same goes for $\mathbf{X}$, $com_{\mathbf{W}}$, $\mathbf{Y}$, and $\mathbf{W}$. In addition, we keep a "one-hot" program counter vector to keep track of subcircuit that is activated by the current VM instruction. Using this one-hot vector, we adapt our folding strategy in a way that only the active component of vector ($\mathbf{B}$, $\mathbf{X}$, $com_{\mathbf{W}}$, $\mathbf{Y}$, $\mathbf{W}$, and $r_{\mathbf{W}}$) are updated in each folding step. The remaining $I-1$ inactive components of these vectors stay the same post-folding so we do not need to touch them. This thus achieves the desired cost profile.

Collaborative Folding/IVC. All prior folding research works assume that a single prover possesses all witnesses. Inspired by collaborative zk-SNARKS [24,37,41], we present the first formal definition and practical realization of collaborative folding/IVC (Definition 24, Construction 43 / 28). It allows $k \geq 1$ prover clients and n untrusted assisting servers to jointly produce IVC proofs for computations such as $\mathbf{z}_{i+1} = F(\mathbf{w}^A, \mathbf{w}^B, \mathbf{z}_i)$ where secret witness $\mathbf{w}^A$ is only accessible by prover Alice and $\mathbf{w}^B$ is private to prover Bob, without revealing the private witnesses owned by each prover client. Moreover, the witness remains hidden as long as only a subset of the servers colludes. For this we analyze the baseline folding/IVC protocol in Construction 12 / 15 and identify the steps that touches the secret witnesses. These steps involve MSM computations, arithmetic circuit evaluation, and univariate polynomial commitments and evaluation proof generations on distributed secret witnesses. In particular, in each folding step, the prover clients need to commit to the secret witnesses. Since Pedersen's commitment boils down to an MSM, by adopting Garg et al.'s collaborative MSM [24], this step is efficiently parallelized across the n servers while maintaining privacy. Moreover, the folding step also requires evaluating $y(t)$ at $D^* + 1$ points from secret witnesses belong to multiple prover clients. This computation can be conducted via MPC using classic protocols such as BGW [3]. After evaluating $y(t)$, in the baseline folding protocol (Construction 12) the prover sends $y(2)$, $y(3)$, ..., $y(D^*)$ to the verifier in the clear, which could leak information about the witnesses in the MPC setting. To address this, we instead let the prover clients commit to $y(t)$ via MPC and only sends com_y, the zero-knowledge polynomial commitment of $y(t)$ to the verifier. The prover clients also need to evaluate $y(t)$ at a $\gamma \in \mathbb{F}$ randomly chosen by the verifier, and generate the corresponding evaluation proof. All these can be done securely and scalably via MPC as we will discuss in Section 7.

4 Folding via Circuit Interpolation

4.1 Normalize the Circuit in Preparation for Folding

As mentioned earlier, our proposed folding protocol works by first normalizing the circuit under consideration into a *shallow* circuit which has a *single* output.

Augment the Incremental Computation F. As in Nova, given a circuit F which performs incremental computation circuit $\mathbf{z}_{i+1} = F(\mathbf{z}_i, \boldsymbol{\omega}_i)$, we first need to augment it to circuit F' by adding the verifier circuit of the non-interactive folding scheme and a couple hashes, in order to perform additional bookkeeping to fold proofs of prior invocations of itself. The augmentation of F follows a process similar to that described in Section 5.1 and illustrated in Figure 3 of the Nova paper [33], except that we replace Nova's $\mathsf{NIFS.V}$ with our own version described in Construction 14. We rearrange the inputs to resulting circuit F' takes to a public input vector $\mathbf{x}'$ and a private witness vector $\mathbf{w}'$, and denote the output vector by $\mathbf{y}'$, i.e., $\mathbf{y}' = F'(\mathbf{x}', \mathbf{w}')$. The formal construction of circuit F' is provided in Appendix B.1.

Circuit Flattening for Total Degree Reduction. In some cases, the augmented circuit $\mathbf{y}' = F'(\mathbf{x}', \mathbf{w}')$ could be a "deep circuit " with many layers, and therefore its total degree D' could potentially be large. To address this, we add an extra step to "flatten" a d-layer circuit F' into a circuit with k parallel blocks, each with at most $l = \lceil d/k \rceil$ layers. Extra wires and gates are added to turn the intermediate values into the inputs of the normalized circuit. The process of flattening F' is depicted in Figure 3.

More formally, we denote the k blocks in the circuit F' (Figure 3(a)) by c_k, c_{k-1}, ..., c_1. The inputs and outputs of the i^{th} block c_i are denoted by $\mathbf{h}_i$ and $\mathbf{h}_{i-1}$, i.e., $\mathbf{h}_{i-1} = c_i(\mathbf{h}_i)$. In particular, $\mathbf{h}_k = (\mathbf{x}', \mathbf{w}')$, and $\mathbf{h}_0 = \mathbf{y}'$. Then, the public input vector $\mathbf{x}$, pubic output vector $\mathbf{y}$, and witness vector $\mathbf{w}$ of the l-layer flattened circuit $\bar{F}$ can be expressed as

$$\begin{aligned} \mathbf{x} &= (\mathbf{x}', \mathbf{y}') \\ \mathbf{y} &= (c_k(\mathbf{i}', \mathbf{w}') - \mathbf{h}_{k-1}, c_{k-1}(\mathbf{h}_{k-1}) - \mathbf{h}_{k-2}, ..., c_1(\mathbf{h}_1) - \mathbf{y}') \\ \mathbf{w} &= (\mathbf{w}', \mathbf{h}_{k-1}, ..., \mathbf{h}_1) \end{aligned} \tag{1}$$

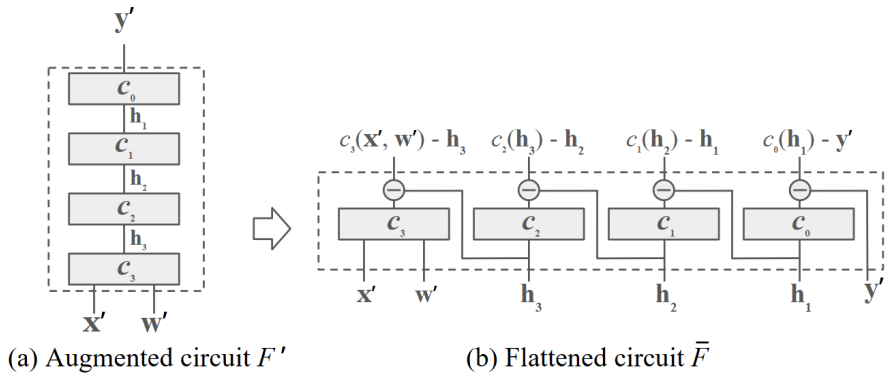
Note that we only need to flatten F' when D' is sufficiently large, as $k > 1$ would increase the witness size and therefore the commitment cost (see Remark 10 for further discussion). When D' is not too large (e.g. $D' < 100$), we can simply set $k = 1$. The net effect of $k = 1$ is just moving $\mathbf{y}'$ from the output to the input, so that the expected output of $\bar{F}$ is a zero vector, which sets the circuit up for the next normalization step. We have the following lemma regarding the completeness and soundness of the flattening procedure. The proof of the lemma is straightforward and is omitted here.

Lemma 4. *If* $(\mathbf{x}', \mathbf{y}', \mathbf{w}')$ *satisfies relation* $\mathbf{y}' = F'(\mathbf{x}', \mathbf{w}')$*, then* $\mathbf{y} = \bar{F}(\mathbf{x}, \mathbf{w})$ *is a zero vector. Conversely, if* $\mathbf{y}$ *is a zero vector,* $(\mathbf{x}', \mathbf{y}', \mathbf{w}')$ *satisfies* $\mathbf{y}' = F'(\mathbf{x}', \mathbf{w}')$*.*

Single Output Reduction. Next, we perform another circuit transformation such that the resulting circuit has a single output. Without this transformation, since $\bar{F}$ may have multiple outputs, the prover would need to send the output polynomial of all outputs to the verifier, which potentially incurs high communication overhead.

Let us assume that the output of the circuit $\bar{F}$ is an n dimensional vector $\mathbf{y} = (y_0, y_1, ..., y_{n-1}) \in \mathbb{F}^n$. To prove to the verifier that this is a zero vector, we can use the random combination trick. That is, come up with a random coefficients vector $\mathbf{s}$ and check if $\mathbf{s} \cdot \mathbf{y} = 0$. We propose to generate the random vector $\mathbf{s}$ using the following process: First, the verifier $\mathcal{V}$ samples $\log n$ independent random values $\boldsymbol{\beta} = (\beta_0, .., \beta_k, .., \beta_{\log n - 1}) \in \mathbb{F}^{\log n}$ and sends them to the prover $\mathcal{P}$. Next, both $\mathcal{P}$ and $\mathcal{V}$ can independently generate the following n random coefficients:

$$\mathbf{s} = \{s_i = \prod_{k=0}^{\log n - 1} (i_k \cdot \beta_k + (1 - i_k) \cdot (1 - \beta_k)) | i = 1, .., n\} \tag{2}$$



(a) Augmented circuit F' (b) Flattened circuit $\bar{F}$

Fig. 3. Flatten the augmented circuit F' to $\bar{F}$. In this example, circuit F' is divided into $k = 4$ blocks $c_1, .., c_4$, each with $l = \lceil d/4 \rceil$ layers. There are other flattening strategies, for example, minimizing the number of outputs of $\bar{F}$ while keeping the degree of $\bar{F}$ below a certain threshold (see also Remark 8).

where i_k is the k^{th} bit of the binary representation of index i. With $\mathbf{s}$ defined above, we will prove in Lemma 5 that we only need to check the following inner product to ensure that $\mathbf{y}' = F'(\mathbf{x}', \mathbf{w}')$ is a zero vector with high probability:

$$\mathbf{s} \cdot \mathbf{y} = 0 \tag{3}$$

We denote the resulting circuit by $y = F^*(\boldsymbol{\beta}, \mathbf{x}, \mathbf{w})$. This circuit has a single output $y \in \mathbb{F}$, and takes three inputs, namely the random vector $\boldsymbol{\beta}$, as well as $\mathbf{x}$ and $\mathbf{w}$ defined in Formula 1. In the following two lemmas we claim that $F'(\mathbf{x}', \mathbf{w}') = \mathbf{y}'$ holds with high probability if $y = 0$. In addition, the prover and the verifier can generate vector $\mathbf{s}$ independently with $O(n)$ field multiplications and additions. The proofs of these two lemmas are deferred to Appendix C.1.

Lemma 5. *If $\mathbf{w}'$ is a valid witness of F', i.e., $F'(\mathbf{x}', \mathbf{w}') = \mathbf{y}'$, and that $\boldsymbol{\beta} = (\beta_0, .., \beta_k, .., \beta_{\log n - 1}) \in \mathbb{F}^{\log n}$ are uniformly random, then the normalized circuit F^* evaluates to 0, i.e., $F^*(\boldsymbol{\beta}, \mathbf{x}, \mathbf{w}) = 0$. Conversely, if $F^*(\boldsymbol{\beta}, \mathbf{x}, \mathbf{w}) = 0$, then $F'(\mathbf{x}', \mathbf{w}') = \mathbf{y}'$ holds with probability at least $1 - \frac{\log n}{|\mathbb{F}|} \geq 1 - \frac{\log |F'|}{|\mathbb{F}|}$.*

Lemma 6. *Generating coefficient vector $\mathbf{s} = (s_0, .., s_i, .., s_{n-1})$ with s_i defined by Formula (2) takes $O_\lambda(n)$ time and $O_\lambda(n)$ memory, and therefore can be implemented as a $O_\lambda(n)$-size circuit.*

As shown in the following lemma, compared to $\bar{F}$, the normalization only increases the circuit size by a constant factor and the total degree by at most $\log n$. The proof of the lemma is provided in Appendix C.1.

Lemma 7. *The size of normalized circuit F^* is in the same order as $\bar{F}$ and F', i.e. $O(|F^*|) = O(|\bar{F}|) = O(|F'|) = O(|F|)$. Moreover, denoting the total degree of F^* and $\bar{F}$ by D^* and $\bar{D}$, respectively, we have $D^* = \bar{D} + \log n$. Denote the depth of F^* and $\bar{F}$ by d^* and $\bar{d}$, respectively, we have $d^* = \min\{\bar{d}, \log n\} + 2$.*

Remark 8. Note that $O(n) \leq O(|\bar{F}|)$ since n is the number of outputs of $\bar{F}$. Lemma 7 further shows $O(|\bar{F}|) = O(|F|)$. Hence, we have $O(n) \leq O(|F|)$. Also, depending on the circuit structure, in many cases we might have $n \ll |F|$ depending on how we flatten the circuit. Moreover, it is possible to reduce the degree of the circuit D^* to $\bar{D}+2$ and depth d^* to $\bar{d}+2$ using alternative methods to generate the vector $\mathbf{s}$, but with different trade-offs. The details are provided in Appendix B.2.

Remark 9. Protostar uses a similar random linear combination technique to collapse multiple constraints into one. However, their prover needs to keep track of the coefficients (i.e., $[\beta_i, \beta'_i]_{i=1}^{\sqrt{l}-1}$ in Figure 6 of the Protostar paper) as part of the witness. This results in two commitments to an extra $O(\sqrt{l})$ terms, which adds extra prover overhead compared to our approach.

Deriving the Output Polynomial for F^*. After the above two-phase circuit normalization, the original circuit F' has been turned into a shallow and single-output circuit $y = F^*(\boldsymbol{\beta}, \mathbf{x}, \mathbf{w})$. Given two satisfying instances $(\boldsymbol{\beta}_0, \mathbf{x}_0, \mathbf{w}_0, y_0)$ and $(\boldsymbol{\beta}_1, \mathbf{x}_1, \mathbf{w}_1, y_1)$, the prover needs to interpolate them into a third instance, and derive the output polynomial $y(t) = F^*(\boldsymbol{\beta}(t), \mathbf{x}(t), \mathbf{w}(t))$ where $\mathbf{w}(t) = (1-t)\cdot\mathbf{w}_0 + t\cdot\mathbf{w}_1$ and $\boldsymbol{\beta}(t)$ and $\mathbf{x}(t)$ are defined similarly. To obtain the closed-form expression of $y(t)$, we can plug $\boldsymbol{\beta}(t)$, $\mathbf{x}(t)$, and $\mathbf{w}(t)$ into the inputs of circuit F^* and derive the closed-form expression of each intermediate gates layer by layer towards the output. Alternatively, since $y(t)$ is a degree-D^* univariate polynomial, we can evaluate $\boldsymbol{\beta}(t)$, $\mathbf{x}(t)$, and $\mathbf{w}(t)$ at D^*+1 points and then compute the corresponding circuit outputs. The prover cost of this approach is bounded by $O(D^* \cdot |F^*|) = O(D^* \cdot |F|)$ field operations. These D^*+1 evaluations of $y(t)$ can be sent directly to the verifier, which can evaluate $y(t)$ at any point $\gamma \in \mathbb{F}$ with $O(D^*)$ field operations online using the barycentric evaluation method with a one-time preprocessing cost of $O({D^*}^2)$ [6].

Remark 10. (Prover cost trade-off) Note that in the circuit flattening process, the number of "blocks" k can be any integer between 1 and d. A larger k reduces D^*, the total degree of F^*, and thus the prover cost for computing the expression of $y(t)$. However, a larger k also increases the size of the witness $\mathbf{w}$, leading to a higher cost to commit to the multilinear extension of $\mathbf{w}$. Hence, k can be a circuit-specific parameter we choose to optimize the prover cost.

4.2 Construction of the Folding Scheme

Definition 11. *Given a circuit F performing step computation $\mathbf{z}_{i+1} = F(\mathbf{z}_i, \boldsymbol{\omega}_i)$ where $\boldsymbol{\omega}_i$ is a non-deterministic input vector, we denote $y = F^*(\boldsymbol{\beta}, \mathbf{x}, \mathbf{w})$ the normalized circuit as described in Section 4.1. We call $(\boldsymbol{\beta}, \mathbf{x}, com_{\mathbf{w}}, y)$ the committed instance for F^*. An instance $(\boldsymbol{\beta}, \mathbf{x}, com_{\mathbf{w}}, y)$ is satisfied by a witness $(\mathbf{w}, r_{\mathbf{w}})$ where $r_{\mathbf{w}} \in \mathbb{F}$, if $com_{\mathbf{w}} = \mathsf{PolyCom}(\mathsf{pp}_{\mathbf{w}}, \mathbf{w}, r_{\mathbf{w}})$ and $y = F^*(\boldsymbol{\beta}, \mathbf{x}, \mathbf{w})$.*

Construction 12 (Interactive Folding Scheme (IFS)**)** *Consider a finite field* $\mathbb{F}$ *and a succinct, hiding, and homomorphic polynomial commitment scheme for multilinear polynomials* PolyCom *over* $\mathbb{F}$*. We define the generator* $\mathcal{G}$ *and encoder* $\mathcal{K}$ *as follows:*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$*: output size bound for* $|F|$*, and commitment parameters* $\mathsf{pp}_{\mathbf{w}}$*.*
- $\mathcal{K}(\mathsf{pp}, F) \to (\mathsf{pk}, \mathsf{vk})$*: output* $\mathsf{pk} \leftarrow (\mathsf{pp}, F^*)$ *and* $\mathsf{vk} \leftarrow \perp$*.*

The verifier $\mathcal{V}$ *takes two committed instances* $\mathsf{u}_0 = (\boldsymbol{\beta}_0, \mathbf{x}_0, com_{\mathbf{w}0}, y_0)$ *and* $\mathsf{u}_1 = (\perp, \mathbf{x}_1, com_{\mathbf{w}1}, 0)$ [2]*. The prover* $\mathcal{P}$*, in addition to the two instances, takes witnesses to both instances* $(\mathbf{w}_0, r_{\mathbf{w}0})$ *and* $(\mathbf{w}_1, r_{\mathbf{w}1})$*. The prover and verifier proceed as follows:*

- $\mathcal{V}$*: Sample independent random field elements* $\boldsymbol{\beta}_1 = (\beta_0, .., \beta_k, .., \beta_{\log n - 1}) \in \mathbb{F}^{\log n}$*, and send them to* $\mathcal{P}$*.*
- $\mathcal{P}$*: Compute the point evaluation form of polynomial* $y(t)$*, i.e. the evaluations of* $y(t) = F^*(\boldsymbol{\beta}(t), \mathbf{x}(t), \mathbf{w}(t))$ *as a function of* t *at* $t = 2, 3, .., D^*$*. Here,* $\mathbf{w}(t) = (1-t) \cdot \mathbf{w}_0 + t \cdot \mathbf{w}_1$*, and* $\boldsymbol{\beta}(t)$ *and* $\mathbf{x}(t)$ *are defined similarly. Send* $\{y(2), y(3), .., y(D^*)\}$[3] *to* $\mathcal{V}$*.*
- $\mathcal{V}$*: Randomly sample* $\gamma \in \mathbb{F}$*, and send it to* $\mathcal{P}$*.*
- $\mathcal{P}$ *and* $\mathcal{V}$*: Output the folded instance* $(\boldsymbol{\beta}_\gamma, \mathbf{x}_\gamma, com_{\mathbf{w}\gamma}, y_\gamma)$ *as follows, where* $y(\gamma)$ *can be efficiently calculated by interpolating a degree-*D^* *polynomial through points* $\{(0, y_0), (1, 0), (2, y(2)), (3, y(3)), .., (D^*, y(D^*))\}$ *via the barycentric evaluation method.*

$$\begin{aligned} \boldsymbol{\beta}_\gamma &\leftarrow (1-\gamma) \cdot \boldsymbol{\beta}_0 + \gamma \cdot \boldsymbol{\beta}_1 \\ \mathbf{x}_\gamma &\leftarrow (1-\gamma) \cdot \mathbf{x}_0 + \gamma \cdot \mathbf{x}_1 \\ com_{\mathbf{w}\gamma} &\leftarrow (1-\gamma) \cdot com_{\mathbf{w}0} + \gamma \cdot com_{\mathbf{w}1} \\ y_\gamma &\leftarrow y(\gamma) \end{aligned}$$

- $\mathcal{P}$*: Output the folded witness* $(\mathbf{w}_\gamma, r_{\mathbf{w}\gamma})$*, where*

$$\begin{aligned} \mathbf{w}_\gamma &\leftarrow (1-\gamma) \cdot \mathbf{w}_0 + \gamma \cdot \mathbf{w}_1 \\ r_{\mathbf{w}\gamma} &\leftarrow (1-\gamma) \cdot r_{\mathbf{w}0} + \gamma \cdot r_{\mathbf{w}1} \end{aligned}$$

Theorem 13. *Construction 12 is a public-coin folding scheme for committed instances of* F^* *that achieves completeness and knowledge soundness.*

[2] Note for u_1, the $\boldsymbol{\beta}$ vector is null $\perp$, meaning the value of $\boldsymbol{\beta}$ is not set when $\mathcal{P}$ and $\mathcal{V}$ enter the protocol. Also, $\mathsf{u}_1.y = 0$. This is for the ease of constructing the IVC protocol, where the accumulating instance u_0 has its $\boldsymbol{\beta}$ already set, while $\mathcal{P}$ and $\mathcal{V}$ will need to generate $\boldsymbol{\beta}$ for u_1, the instance to be folded. Also, per Lemma 5, $\mathsf{u}_1.y$ should always be 0 for any $\boldsymbol{\beta}$, if the witeness $(\mathbf{w}_1, r_{\mathbf{w}1})$ satisfies the instance u_1.

[3] Given $\mathbf{w}(t) = (1-t) \cdot \mathbf{w}_0 + t \cdot \mathbf{w}_1$, we have that $\mathbf{w}(0) = \mathbf{w}_0$ and $\mathbf{w}(1) = \mathbf{w}_1$. Thus, $y(0) = y_0$ and $y(1) = 0$ are already available in instances $\mathsf{u}_0 = (\boldsymbol{\beta}_0, \mathbf{x}_0, com_{\mathbf{w}0}, y_0)$ and $\mathsf{u}_1 = (\perp, \mathbf{x}_1, com_{\mathbf{w}1}, 0)$. Hence $\mathcal{P}$ does not need to send them to $\mathcal{V}$.

The proof for the above Theorem will be presented in Appendix C.1. Construction 12 can be turned non-interactive as follows. We also assume that it satisfies completeness and knowledge soundness in the standard model when ρ is instantiated with a cryptographic hash function [33].

Construction 14 (Non-Interactive Folding Scheme (NIFS)) *Under the random oracle model, non-interactivity can be achieved using the strong Fiat-Shamir transformation. Let ρ denote a random oracle sampled during parameter generation and provided to all parties. Let $(\mathsf{G}, \mathsf{K}, \mathsf{P}, \mathsf{V})$ represented the interactive folding scheme defined above in Construction 12. A non-interactive folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ can be constructed as follows:*

- $\mathcal{G}(1^\lambda)$*: output* $\mathsf{pp} \leftarrow \mathsf{G}(1^\lambda)$.
- $\mathcal{K}(\mathsf{pp}, F)$*:* $\mathsf{vk} \leftarrow \rho(\mathsf{pp}, F)$ *and* $\mathsf{pk} \leftarrow (\mathsf{pp}, F^*, \mathsf{vk})$*; output* $(\mathsf{vk}, \mathsf{pk})$.
- $\mathcal{P}(\mathsf{pk}, (\mathsf{u}_0, \mathbf{w}_0), (\mathsf{u}_1, \mathbf{w}_1))$*: Generate* $\log n$ *random values* $\beta_0 \leftarrow \rho(\mathsf{vk}, \mathsf{u}_0, \mathsf{u}_1)$, $\beta_1 \leftarrow \rho(\mathsf{vk}, \mathsf{u}_0, \mathsf{u}_1, \beta_0)$, .., $\beta_{\log n-1} \leftarrow \rho(\mathsf{vk}, \mathsf{u}_0, \mathsf{u}_1, \beta_0, .., \beta_{\log n-2})$*; Compute* $\{y(2), y(3), .., y(D^*)\}$ *and forward them to* IFS.P *and* IFS.V*; Generate randomness* $\gamma \leftarrow \rho(\mathsf{vk}, \mathsf{u}_0, \mathsf{u}_1, \beta_0, .., \beta_{\log n-2}, y(2), .., y(D^*))$*, and calculate* $y_\gamma = y(\gamma)$ *using the barycentric interpolation method and then output the resulting folded instance and witness.*
- $\mathcal{V}(\mathsf{vk}, \mathsf{u}_0, \mathsf{u}_1)$*: runs* IFS.V *with randomness* $(\beta_0, .., \beta_{\log n-1})$ *generated the same way as* $\mathcal{P}$*; Receive* $\{y(2), y(3), .., y(D^*)\}$ *and generaet* γ *the same way as* $\mathcal{P}$*. Calculate* $y_\gamma = y(\gamma)$ *via the barycentric interpolation method and output the resulting folded instance.*

5 The Interstellar IVC Scheme

With the non-interactive folding scheme defined in the previous section, we can show how to construct an IVC scheme, and turn it into a SNARK.

5.1 The IVC Protocol Construction

We use $\mathsf{U_i} = (\boldsymbol{\beta}_i, \mathbf{X}_i, com_{\mathbf{W}i}, Y_i)$ to represent the correct execution of invocations $1, .., i-1$ of F^*, and $\mathsf{u_i} = (\boldsymbol{\beta}_i, \mathbf{x}_i, com_{\mathbf{w}i}, y_i)$ to represent the correct execution of the i^{th} invocation of F^*. The construction of the IVC protocol is given below, whose completeness and knowledge soundness are proven in Theorem 16:

Construction 15 (Interstellar IVC Protocol (InterstellarIVC)) *Based on the above building blocks, the IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is defined as follows:*

- $\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}$*: Output* $\mathsf{NIFS.G}(1^\lambda)$.
- $\mathcal{K}(\mathsf{pp}, F) \rightarrow (\mathsf{pk}, \mathsf{vk})$*:*
  - *generate circuit* $F^* \leftarrow \mathsf{NORMALIZE}(\mathsf{pp}, F)$ *as described in Section 4.1;*
  - *generate* $(\mathsf{pk_{fs}}, \mathsf{vk_{fs}}) \leftarrow \mathsf{NIFS.K}(\mathsf{pp}, F^*)$*;*
  - *output* $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((\mathsf{pp}, (\mathsf{pk_{fs}}, F), (\mathsf{pp}, (\mathsf{vk_{fs}}, F^*))$.
- $\mathcal{P}(\mathsf{pk}, (i, \mathbf{z}_0, \mathbf{z}_i), \boldsymbol{\omega}_i, \Pi_i) \rightarrow \Pi_{i+1}$*:*

- *if* $i = 0$*, set* $(\mathsf{U}_{i+1}, \mathbf{W}_{i+1}) \leftarrow ((\boldsymbol{\beta} = \mathbf{0}, \mathbf{x} = \mathbf{0}, com_{\mathbf{0}}, y = 0), \mathbf{w}_0 = \mathbf{0})$*, where* $com_{\mathbf{0}}$ *is the commitment to the zero vector* $\mathbf{0}$*; Otherwise, parse* Π_i *as* $((\mathsf{U}_i, \mathbf{W}_i), (\mathsf{u}_i, \mathbf{w}_i), r_i)$ *and then compute* $(\mathsf{U}_{i+1}, \mathbf{W}_{i+1}) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}, (\mathsf{U}_i, \mathbf{W}_i), (\mathsf{u}_i, \mathbf{w}_i))$*;*
- *sample* $r_{i+1} \leftarrow \mathbb{F}$ *randomly;*
- *compute* $(\mathsf{u}_{i+1}, \mathbf{w}_{i+1}) \leftarrow \mathsf{trace}(F^*, (\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, \mathbf{z}_0, \mathbf{z}_i), \boldsymbol{\omega}_i, r_i, r_{i+1}))$*, where* $\mathsf{u}_{i+1}.\boldsymbol{\beta} = \perp$ *and* $\mathsf{u}_{i+1}.y = 0$*; Note that* trace *is a randomized algorithm that internally samples randomness* $r_{\mathbf{w},i+1}$ *to create hiding commitments* $com_{\mathbf{w},i+1} = \mathsf{PolyCom}(\tilde{\mathbf{w}}_{i+1}, r_{\mathbf{w},i+1})$ *inside* u_{i+1}*, where* $\tilde{\mathbf{w}}_{i+1}$ *is the multilinear extension of* $\mathbf{w}_{i+1}$*;*
- *output* $\Pi_{i+1} \leftarrow ((\mathsf{U}_{i+1}, \mathbf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathbf{w}_{i+1}), r_{i+1})$.

- $\mathcal{V}(\mathsf{vk}, (i, \mathbf{z}_0, \mathbf{z}_i), \Pi_i) \rightarrow \{0, 1\}$*:*
  - *if* $i = 0$*, check if* $\mathbf{z}_i = \mathbf{z}_0$*;*
  - *otherwise,*
    - *parse* Π_i *as* $((\mathsf{U}_i, \mathbf{W}_i), (\mathsf{u}_i, \mathbf{w}_i), r_i)$*,*
    - *check if* $\mathsf{u}_i.y = 0$ *and if* $\mathsf{u.x} = \mathsf{hash}(\mathsf{vk}, i, \mathbf{z}_0, \mathbf{z}_i, \mathsf{U}_i, r_i)$*,*
    - *randomly samples* $\boldsymbol{\beta}_i \in \mathbb{F}^{\log n}$ *and set* $\mathsf{u}_i.\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_i$*,*
    - *check if* $\mathbf{W}_i$ *and* $\mathbf{w}_i$ *are satisfying witnesses to* U_i *and* u_i *respectively using* $\mathsf{vk.pp}$ *and* $\mathsf{vk}.F^*$.

Instantiating PolyCom As a well-known fact, given a vector commitment to a n-sized vector over $\mathbb{F}$, a corresponding multilinear polynomial commitment scheme for $\log n$ variables can be constructed, if there exists an argument protocol to prove an inner product computation between a committed vector and an n-sized public vector $(r_0, 1 - r_0) \otimes .. \otimes (r_{n-1}, 1 - r_{n-1})$, where $\mathbf{r} \in \mathbb{F}^{\log n}$ is an evaluation point [33]. Examples of such vector commitments include Bulletproofs [13] and Dory [34]. In the following, we analyze the pros and cons of each option.

Bulletproofs $\mathsf{PolyCom}_{\mathsf{BP}}$. We can leverage Bulletproofs if we choose Pedersen's commitments as the PolyCom scheme used in the IVC protocol. For a $\log n$-variable multilinear polynomial, committing takes $O_\lambda(n)$ time to produce an $O_\lambda(1)$-sized commitment. To generate an evaluation proof, the prover time is $O_\lambda(n)$. The evaluation proof has size $O_\lambda(\log n)$ and takes the verifier $O_\lambda(n)$ time to verify. $\mathsf{PolyCom}_{\mathsf{BP}}$ has a transparent setup and does not assume pairing.

Dory $\mathsf{PolyCom}_{\mathsf{Dory}}$. For a $\log n$-variable multilinear polynomial, $\mathsf{PolyCom}_{\mathsf{Dory}}$ takes $O_\lambda(n)$ time to produce an $O_\lambda(1)$-sized commitment, same as $\mathsf{PolyCom}_{\mathsf{BP}}$. For evaluation, both the proof size and verification time are in $O_\lambda(\log n)$. While having a more efficient verifier than $\mathsf{PolyCom}_{\mathsf{BP}}$, $\mathsf{PolyCom}_{\mathsf{Dory}}$ assumes the hardness of SXDH, but also has a transparent setup.

The following theorem claims the completeness, knowledge soundness, and efficiency of the Interstellar IVC scheme when composing with different PolyCom schemes listed above. The proof follows the same reasoning as in our motivating example, with the full formal proof deferred to Appendix C.2.

Theorem 16. *Construction 15 is an IVC scheme for the circuit* F *that satisfies completeness and knowledge soundness. In terms of efficiency, for each folding step, the prover performs* $O(\log |F|)$ *hashes and* $O(D^*|F|)$ *field operations. On*

top of that, when we instantiate PolyCom *with the either the Bulletproofs or Dory commitment scheme, the prover performs a single group scalar multiplication and a single group MSM of size* $|\mathbf{w}|$*, where* $\mathbf{w}$ *is the witness to circuit* F^**.*

5.2 IVC Proof Compression by SNARK Composition

Note that the IVC proof generated with Construction 15 has a proof size linear in the size of circuit F. Similarly to Nova, we can add a second-stage SNARK protocol to further compress the IVC proof size. For that, we can apply a modified version of Construction 4 in the Nova paper [33]. The details are deferred to Construction 38 in Appendix B.3. A small optimization is to fold the last two statements, i.e. U_i and u_i, into a single statement U', so that the SNARK only needs to prove one single statement. Regarding this construction, we have the following theorem, whose proof is provided in Appendix C.3:

Theorem 17. *Construction 38 is a SNARK of a valid IVC proof produced by Construction 15.*

Construction 38 requires an SNARK scheme for the circuit satisfiability problem to succinctly prove the knowledge of a valid IVC proof. Our IVC scheme sets itself up nicely for employing GKR-based SNARK protocols, as the folded instance $(\boldsymbol{\beta}_\gamma, \mathbf{x}_\gamma, com_{\mathbf{w}\gamma}, y_\gamma)$ includes $(\boldsymbol{\beta}_\gamma, \mathbf{x}_\gamma, y_\gamma)$, the public inputs/output of the final folded instance for the circuit F^*, as well as $com_{\mathbf{w}\gamma}$, the polynomial commitment to the multilinear extension of the witness. Thus, we can apply GKR-based SNARK protocols such as Libra [51] to turn our IVC scheme into a SNARK. Note that the original version of Libra uses the KZG/PST commitment but can be replaced with the multilinear PolyCom schemes discussed in the previous section. We have the following theorem regarding the SNARK construction for circuit satisfiability, whose proof is presented in Appendix C.3.

Theorem 18. *There exists a SNARK in the random oracle model for the circuit satisfiability problem for circuit* F *with the following efficiency characteristics, where* d^* *is the number of layers in the normalized circuit* F^**:*

- *Assuming the hardness of the discrete logarithm problem, the prover runs in time* $O_\lambda(|F|)$*; the proof length is* $O_\lambda(d^* \log |F|)$*; and the verifier online run time is in* $O_\lambda(d^* \log |F| + |\mathbf{w}|)$*.*
- *Assuming the hardness of the SXDH problem, the prover runs in time* $O_\lambda(|F|)$*; the proof length is* $O_\lambda(d^* \log |F|)$*; and the verifier online run time* $O_\lambda(d^* \log |F|)$*.*

Remark 19. The original Libra SNARK protocol achieves linear prover time for any circuit. However, the verifier time is sublinear only for log-space uniform circuits. This is because the protocol requires the verifier to compute multilinear extensions $\tilde{add}_i(g, x, y)$ and $\tilde{mult}_i(g, x, y)$ for each layer when running the GKR protocol. As pointed out by other follow-up works in the literature (e.g. the caption of Figure 9 in the Brakedown paper [26] regarding Hyrax*), we can employ Spartan's *computation commitment* idea to outsource these computations to the prover. In more detail, we can introduce a preprocessing step for

the verifier to commit to all $\tilde{add}_i(g,x,y)$ and $\tilde{mult}_i(g,x,y)$. During GKR execution, the verifier can ask the prover to return their evaluations at random points with proofs. With these proofs and the polynomial commitments to $\tilde{add}_i(g,x,y)$ and $\tilde{mult}_i(g,x,y)$ generated earlier by herself, the verifier can check whether the evaluations are correct. This keeps the verifier's online run time in $O_\lambda(d^* \cdot \log |F|)$ even for non-uniform circuits when $\mathsf{PolyCom}_{\mathsf{Dory}}$ is used.

6 Interstellar+: Extended Folding/IVC Scheme

In this section, we explore extensions of the Interstellar Folding/IVC scheme to handle high-degree gates, look-up gates, multi-way folding, and support non-uniform IVC, which are critical for support many real-world applications. We defer the details of non-uniform IVC to Appendix B.5.

6.1 Handling High-Degree Gates

To handle high-degree gates in our framework, we can simply "unroll" them into multiplication and addition gates. For example, a high-degree gate $y = x^4$ can be implemented using two consecutive multiplication gates calculating $z = x^2$ and $y = z^2$, respectively. In other folding/IVC mechanisms using constraint systems such as R1CS, this unrolling could lead to a higher prover cost, as it increases the number of constraints and thus the witness and cross-term vector size, which results in more group operations for committing to these vectors.

The cost impact of the unrolling on our IVC scheme is minimal. First, for the folding operation, if the circuit flattening process does not "cut through" the unrolled circuit representing the high-degree gates, no extra witness will be created. Hence, there is zero additional overhead for folding the instance and witness, no matter how high the degree of the gate is. Thus, we can optimize the circuit flattening procedure so that it cuts through as few unrolled high-degree gates as possible, which adds minimal or even zero additional cost for the folding protocol. Secondly, for the SNARK proof generation step for the final folded circuit, unrolling could increase the number of layers of the circuit, and thus adds more work for the GKR protocol. However, SNARK proof generation occurs only once for the entire IVC. The amortized cost per folding step would approach zero as the number of steps increases. In summary, handling high-degree gates by unrolling them creates only a small overhead for our framework.

6.2 Support for Lookup Gates

Inspired by Protostar [11], we leverage the following lemma from [27] to handle lookup gates. We denote a size-P lookup table as $\mathbf{p} = \{p_i | i = 0, .., P-1\}$. Then we have [27]:

Lemma 20. *Let $\mathbb{F}$ be a field of characteristic $c > \max(l, P)$. Given two sequences of field elements $[v_i]_{i=0}^{l-1}$ and $[p_i]_{i=0}^{P-1}$, we have $\{v_i | i = 0, 1, .., l-1\} \subset \mathbf{p}$*

as sets (with multiples of values removed) if and only if there exists a sequence $[m_i]_{i=0}^{P-1}$ *of field elements such that*

$$\sum_{i=0}^{l-1} \frac{1}{X+v_i} = \sum_{i=0}^{P-1} \frac{m_i}{X+p_i} \tag{4}$$

Our idea is to further augment the circuit F^* with a sub-circuit for the prover and verifier to run an interactive protocol to evaluate Equation 4 at a random point τ. For that, we augment F^* with additional public inputs τ, $\mathbf{m} = \{m_i | i = 0, .., P-1\}$, $\mathbf{h} = \{h_i | i = 0, .., l-1\}$, and $\mathbf{g} = \{g_i | i = 0, .., P-1\}$. We also add circuits to perform the following checks for these inputs [11]:

$$(v_i + \tau) \cdot (h_i \cdot (v_i + \tau) - 1) = 0 \tag{5}$$
$$(p_i + \tau) \cdot (g_i \cdot (p_i + \tau) - m_i) = 0 \tag{6}$$

Here $\mathbf{v} = \{v_i | i = 0, 1, .., l-1\} \subseteq \mathbf{w}$ is a set of values (e.g. some of the intermediate gate inputs/outputs) that we want to perform the lookup checks. The lookup table $\mathbf{p} = \{p_i | i = 0, .., P-1\}$ is also an additional public input vector. We denote the augmented circuit by G^*. To simplify notation, we absorbed them into an extended public input vector $\mathbf{x}_A = (\mathbf{x}, \mathbf{p}, \tau, \mathbf{m}, \mathbf{h}, \mathbf{g})$.

Construction 21 (Interactive Folding Scheme with Lookup Gates) *Let us consider a finite field* $\mathbb{F}$ *and a succinct, hiding, and homomorphic polynomial commitment scheme for multilinear polynomials* PolyCom *over* $\mathbb{F}$*. We define the generator and encoder as follows:*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$*: output size bound for* $|F|$*, and commitment parameters* $\mathsf{pp}_{\mathbf{w}}$*.*
- $\mathcal{K}(\mathsf{pp}, F') \to (\mathsf{pk}, \mathsf{vk})$*: output* $\mathsf{pk} \leftarrow (\mathsf{pp}, G^*)$ *and* $\mathsf{vk} \leftarrow \perp$*.*

The verifier $\mathcal{V}$ *takes two committed satisfying instances* $\mathsf{u}_0 = (\boldsymbol{\beta}_0, \mathbf{x}_{A0}, com_{\mathbf{w}0}, y_0)$ *and* $\mathsf{u}_1 = (\perp, \mathbf{x}_{A1}, com_{\mathbf{w}1}, 0)$ *where* $\mathbf{x}_{A0}$ *is fully specified, but* $\mathbf{x}_{A1} = (\mathbf{x}_1, \mathbf{p}, \tau = \perp, \mathbf{m} = \perp, \mathbf{h} = \perp, \mathbf{g} = \perp)$ *is only partially specified. The prover* $\mathcal{P}$*, in addition to the two instances, takes witnesses to both instances* $\mathbf{w}_0$ *and* $\mathbf{w}_1$*. The prover and verifier proceed as follows:*

- $\mathcal{P}$*: send* $\mathbf{m} = \{m_i = \sum_{j=0}^{l-1} \mathbf{1}(v_j = p_i) | i = 0, .., l-1\}$ *to* $\mathcal{V}$*.*
- $\mathcal{V}$*: Randomly sample* $\tau \in \mathbb{F}$*, and send it to* $\mathcal{P}$*.*
- $\mathcal{P}$*: Compute* $\mathbf{h} = \{h_i = \frac{1}{v_i+\tau} \text{ if } v_i + \tau \neq 0 \text{ else } 0 | i = 0, .., l-1\}$ *and* $\mathbf{g} = \{g_i = \frac{m_i}{p_i+\tau} \text{if } p_i + \tau \neq 0 \text{ else } 0 | i = 0, .., P-1\}$*. Send* $\mathbf{h}$ *and* $\mathbf{g}$ *to* $\mathcal{V}$*.*
- $\mathcal{P}$ *and* $\mathcal{V}$*: independently update* $\mathbf{x}_{A1} \leftarrow (\mathbf{x}_1, \mathbf{p}, \tau, \mathbf{m}, \mathbf{h}, \mathbf{g})$*.*
- $\mathcal{P}$ *and* $\mathcal{V}$*: Execute the interactive folding protocol* IFS.P *and* IFS.V *in Construction 12 on* $\mathsf{u}_0 = (\boldsymbol{\beta}_0, \mathbf{x}_{A0}, com_{\mathbf{w}0}, y_0)$ *and* $\mathsf{u}_1 = (\perp, \mathbf{x}_{A1}, com_{\mathbf{w}1}, 0)$ *for circuit* G^**, and output the folded instances.* $\mathcal{P}$ *also outputs the folded witness.*

Theorem 22. *For circuits with lookup gates, the interactive folding scheme in Construction 21 achieves completeness and knowledge-soundness.*

The proof for the above theorem is presented in Appendix C.4. The non-interactive version of Construction 21 (using Fiat-Shamir transformation) can be a drop-in replacement for the NIFS scheme used in Construction 15 to extend the IVC scheme to support circuits with lookup gates.

6.3 Folding Multiple Instances at Once

Our folding scheme can be extended to support folding $k > 2$ instances at once. The folded instance has a constant size. First, we denote the witness vector for the k instances by $\{\mathbf{w}_i | j = 0, 1, .., k-1\}$. Let us define the following multilinear combination: $\mathbf{w}(t) = \sum_{j=0}^{k-1} \delta(t, j) \cdot \mathbf{w}_j$, where $\delta(t, j)$ is the Lagrange basis polynomial as defined in Section A.2 [4]. The verifier samples a random field element $\gamma \in \mathbb{F}$ and sends it to the prover to fold the k witnesses into one:

$$\mathbf{w}_\gamma = \sum_{j=0}^{k-1} \delta(\gamma, j) \cdot \mathbf{w}_j$$

In a similar fashion, both the prover and the verifier can fold and derive $\mathbf{x}_\gamma$ and $\boldsymbol{\beta}_\gamma$, that is, $\mathbf{x}_\gamma = \sum_{j=0}^{k-1} \delta(\gamma, j) \cdot \mathbf{x}_j$ and $\boldsymbol{\beta}_\gamma = \sum_{j=0}^{k-1} \delta(\gamma, j) \cdot \boldsymbol{\beta}_j$. An optimization is that we can reuse the same $\boldsymbol{\beta} = \{\beta_i \mid i = 0, 1, .., \log n - 1\}$ for $\{\mathsf{u}_1, \mathsf{u}_2, .., \mathsf{u}_{k-1}\}$. In other words, verifier samples $\log n$ independent random values $\boldsymbol{\beta} = (\beta_0, .., \beta_k, .., \beta_{\log n - 1}) \in \mathbb{F}^{\log n}$. And both the prover and verifier can set $\mathsf{u}_1.\boldsymbol{\beta} = \mathsf{u}_2.\boldsymbol{\beta} = .. = \mathsf{u}_{k-1}.\boldsymbol{\beta}_{k-1} = \boldsymbol{\beta}$. Moreover, since all instances are using the same $\boldsymbol{\beta}$, the circuitry to generate vector $\mathbf{s}$ (Formula 2) can be shared among the instances. We will prove in Theorem 23 that this is sufficient to guarantee knowledge soundness.

As for $com_{\mathbf{w}\gamma}$, we note that the multilinear extension of $\mathbf{w}_\gamma$ can be derived as $\tilde{\mathbf{w}}_\gamma(z) = \sum_{i=0}^{|\mathbf{w}|-1} \tilde{\mathsf{eq}}(z, i) \cdot \mathbf{w}_{\gamma i} = \sum_{i=0}^{|\mathbf{w}|-1} \tilde{\mathsf{eq}}(z, i) \cdot (\sum_{j=0}^{k-1} \delta(\gamma, j) \cdot \mathbf{w}_{ji}) = \sum_{j=0}^{k-1} \delta(\gamma, j) \cdot (\sum_{i=0}^{|\mathbf{w}|-1} \tilde{\mathsf{eq}}(z, i) \cdot \mathbf{w}_{ji}) = \sum_{j=0}^{k-1} \delta(\gamma, j) \cdot \tilde{\mathbf{w}}_j(z)$, which is just a linear combination of $\{\tilde{\mathbf{w}}_j(z) | j = 0, 1, .., k-1\}$. Then, given the additive homomorphic property of the $\mathsf{PolyCom}$ scheme, the prover and verifier can independently generate $com_{\mathbf{w}\gamma}$, the polynomial commitment to $\tilde{\mathbf{w}}_\gamma(z)$ as:

$$com_{\mathbf{w}\gamma} = \sum_{j=0}^{k-1} \delta(\gamma, j) \cdot com_{\mathbf{w}j}$$

As for the output polynomial $y(t)$, it is still a low-degree univariate polynomial whose degree in t is bounded by $k \cdot D^*$, since each of the Lagrange bases $\{\delta(t, j) | j = 0, 1, .., k-1\}$ is a degree-k univariate polynomial of t. The prover can derive the evaluation form of $y(t)$ and send it to the verifier to continue the folding process.

Through the above protocol, the prover and the verifier can independently output $(\boldsymbol{\beta}_\gamma, \mathbf{x}_\gamma, com_{\mathbf{w}\gamma}, y_\gamma)$, which folds the k instances into one. In addition, the

[4] Note that when $k = 2$, $\delta(t, 0) = 1 - t$ and $\delta(t, 1) = t$. Thus, the technique described in this section is a natural extension of Construction 12.

prover outputs the folded witness $(\mathbf{w}_\gamma, r_\mathbf{w})$. The above procedure can fold any number of instances at once, as long as $k \cdot D^*$, the total degree of the output polynomial is still sufficiently small compared to the field size. We provide the formal construction of the interactive and non-interactive protocol for folding multiple instances at once in Construction 39 and Construction 40 in Appendix B.4. The size of the folded instance remains constant, independently of how many instances are folded at once. Finally, it is worth pointing out that with the support for folding multiple instances at once, we can extend the IVC protocol to PCD rather straightforwardly. We omit the details, since this is not the focus of this paper. The following theorem states the completeness, knowledge soundness, and efficiency of Construction 39 and Construction 40. The proof of the theorem is presented in Appendix C.4.

Theorem 23. *The scheme for folding k instance at once in Construction 39 is a public coin protocol that achieves the completeness and knowledge-soundness property. In terms of efficiency, for the non-interactive version of the protocol (Construction 40), the prover performs* $O(\log |F|)$ *hashes and* $O(kD^*|F|)$ *field operations. The size of the folded instance has a constant of size* $O_\lambda(|F|)$*. On top of that, if we instantiate* PolyCom *with either the Bulletproofs or Dory commitment scheme, the prover performs k size-*$|\mathbf{w}|$ *group MSMs and k group scalar-multiplications and additions.*

7 Interstellar++: Support for Collaborative Folding/IVC

The techniques introduced above, and all the existing folding research works to the best of our knowledge [7,8,11,18–20,30–33,40,53], assume that a single party possesses the witnesses of all instances to be folded. However, in practice, it may arise that the witness itself is distributed across multiple parties, each of which wishes to preserve the privacy of their own portion. These scenarios have been studied in the context of collaborative zk-SNARKs [24, 36, 37, 41], where secret sharing and multi-party computation (MPC) are leveraged to enable distributed proof generation among untrusted parties.

Motivated by these developments, we extend our folding framework to the setting of *Collaborative Folding/IVC*, where folding and IVC can be executed jointly by multiple parties in a privacy-preserving and scalable manner. Moreover, the witness remains hidden as long as only a subset of the servers collude. As an example, assume we need to fold two instances of a circuit $y = F(\mathbf{w}^A, \mathbf{w}^B)$ where for each instance, witness $\mathbf{w}^A$ only knows to Alice, and $\mathbf{w}^B$ is only accessible by Bob. Also, similar to some of the collaborative zk-SNARKs works [24,37]. We assume there are n servers available to assist the folding/IVC proof generation but should not learn anything about the witnesses.

Let us first look at how to turn the IVC protocol in Construction 15 into a Collaborative IVC at a high level. In fact, there are only two places in the *prover protocol* which non-trivially touches the secret witnesses, namely the first and the third step. The first step invokes $\mathsf{NIFS.P}$ to fold $(\mathsf{U}_i, \mathbf{W}_i)$ and $(\mathsf{u}_i, \mathbf{w}_i)$

together. We will discuss in details later in this section on how to turn this folding step into an MPC. The third step creates commitment $com_{\mathbf{w},i+1} = \mathsf{PolyCom}(\tilde{\mathbf{w}}_{i+1}, r_{\mathbf{w},i+1})$. We note that this commitment can be generated via MPC efficiently if we use Pedersen's commitment as the $\mathsf{PolyCom}$ scheme. This is because the computational bottleneck of generating a Pedersen's commitment for a long vector $\mathbf{w}$ is the Multi-Scalar Multiplication (MSM) computation. For this, we can leverage Garg et al.'s approach [24] for collaborative MSM to offload the computation to the n servers efficiently.

7.1 Collaborative Folding Construction

Definition 24 (Collaborative Folding). *Let λ be the security parameter. Assume there are k the number of prover clients each holds a subset of the witness of each instance, and there are n servers that can assist with the proof generation. $\mathcal{R}$ is an NP relation. A collaborative folding scheme for relation $\mathcal{R}$ consists of a PPT generator algorithm $\mathcal{G}$, a deterministic encoder algorithm $\mathcal{K}$, and a multiparty protocol Π and verifier $\mathcal{V}$ respectively, defined as:*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$*: On input security parameter λ, samples public parameters* pp.
- $\mathcal{K}(\mathsf{pp}, s) \to (\{\mathsf{pk}_i | i \in [n]\}, \mathsf{vk})$*: On input* pp*, and a common structure s between instances to be folded, outputs prover key* pk_i *for the corresponding server and a verifier key* vk.
- $\Pi_{online}((\mathsf{u}_0, \{\mathbf{w}_0^j | j \in [k]\}), (\mathsf{u}_1, \{\mathbf{w}_1^j | j \in [k]\})) \to (\mathsf{u}, \{\mathbf{w}^j | j \in [k]\})$*: This is an MPC protocol between the verifier $\mathcal{V}$, a set of k prover clients $\mathcal{PC} = \{\mathcal{C}_j | j \in [k]\}$, and a set of n servers $\mathcal{PS} = \{\mathcal{S}_i | i \in [n]\}$. The j^{th} prover client holds $\mathbf{w}_0^j$ and $\mathbf{w}_1^j$, namely its own portion of witnesses $\mathbf{w}_0$ and $\mathbf{w}_1$ of the two instances to be folded. Given the two instances u_0 and u_1 the prover clients and servers then engage in an interactive protocol among themselves to compute the folded instance $(\mathsf{u}, \{\mathbf{w}^j | j \in [k]\})$ where the folded witness portion $\mathbf{w}^j$ is only accessible to the j^{th} prover client throughout the process. Here $\mathsf{u} = (\boldsymbol{\beta}, \mathbf{x}, \{com_{\mathbf{w}}^j | j \in [k]\}, y)$, and u_0 and u_1 are defined accordingly.*
- $\mathcal{V}(\mathsf{vk}, \mathsf{u}_0, \mathsf{u}_1) \to \mathsf{u}$*: On input instances* u_0 *and* u_1*, outputs a new instance* u.

We say that $\Pi = (\mathcal{K}, \Pi_{online})$ is a ***collaborative folding*** *scheme for relation $\mathcal{R}$, if the following properties are satisfied:*

- ***Completeness****. For all PPT adversaries $\mathcal{A}$*

$$\Pr\left[(\mathsf{pp}, s, \mathsf{u}, \mathbf{w}) \in \mathcal{R} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (s, (\mathsf{u}_0, \{\mathbf{w}_0^j | j \in [k]\}), (\mathsf{u}_1, \{\mathbf{w}_1^j | j \in [k]\})) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, s, \mathsf{u}_0, \{\mathbf{w}_0^j | j \in [k]\}), (\mathsf{pp}, s, \mathsf{u}_1, \{\mathbf{w}_1^j | j \in [k]\}) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, s), \\ (\mathsf{u}, \{\mathbf{w}^j | j \in [k]\}) \leftarrow \Pi_{online}(\\ \quad \{\mathsf{pk}_i | i \in [n]\}, (\mathsf{u}_0, \{\mathbf{w}_0^j | j \in [k]\}), (\mathsf{u}_1, \{\mathbf{w}_1^j | j \in [k]\})) \end{array}\right] = 1$$

- ***Knowledge-Soundness***. *A collaborative folding scheme satisfies knowledge soundness if for any expected polynomial-time adversary prover clients $\mathcal{PC}^*$ there is an expected polynomial-time extractor $\mathcal{E}$ such that*

$$\Pr\left[\begin{array}{l|l}(\mathsf{pp}, s, \mathsf{u}_0, \{\mathbf{w}_0^j | j \in [k]\}) \in \mathcal{R}, & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{pp}, s, \mathsf{u}_1, \{\mathbf{w}_1^j | j \in [k]\}) \in \mathcal{R} & (s, (\mathsf{u}_0, \mathsf{u}_1)) \leftarrow \mathcal{PC}^*(\mathsf{pp}, \rho), \\ & (\{\mathbf{w}_0^j | j \in [k]\}, \{\mathbf{w}_1^j | j \in [k]\}) \leftarrow \mathcal{E}(\mathsf{pp}, \rho)\end{array}\right] \geq$$

$$\Pr\left[(\mathsf{pp}, s, \mathsf{u}, \mathbf{w}) \in \mathcal{R} \;\middle|\; \begin{array}{l}\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (s, (\mathsf{u}_0, \mathsf{u}_1)) \leftarrow \mathcal{PC}^*(\mathsf{pp}, \rho), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, s), \\ (\mathsf{u}, \{\mathbf{w}^j | j \in [k]\}) \leftarrow \langle \mathcal{PC}^*(\mathsf{pk}, \rho),\ \mathcal{V}(\mathsf{vk})\rangle(\mathsf{u}_0, \mathsf{u}_1)\end{array}\right] - \mathsf{negl}(\lambda)$$

- ***m-Zero-Knowledge***. *For all semi-honest PPT adversaries $\mathcal{A}$ controlling at most a m-sized subset $\mathsf{Corr} \subset [n]$ of the servers, and an honest verifier, there exists an efficient simulator Sim_{cf}, such that the following holds*

$$\begin{aligned}&\mathsf{view}^{\mathcal{A}}_{\Pi_{online}}[\{\mathsf{pk}_i | i \in [n]\}, (\mathsf{u}_0, \{\mathbf{w}_0^j | j \in [k]\}), (\mathsf{u}_1, \{\mathbf{w}_1^j | j \in [k]\})] \\ &\approx_c \mathsf{Sim}_{cf}(\mathsf{pp}, \{\mathsf{pk}_i | i \in [n]\}, (\mathsf{u}_0, \{\mathbf{w}_0^j | j \in [k]\}), (\mathsf{u}_1, \{\mathbf{w}_1^j | j \in [k]\})\}\end{aligned}$$

For ease of presentation below we assume the circuit flattening step in Section 4.1 is skipped, such that $\{\mathbf{w}_i^j\}$ are the original witnesses without intermediate gate values. The protocols can be extended to handle the more general case with circuit flattening by modifying the above definition and requiring the n servers to maintain the shares of the extended witness after each folding step. Given the linearity of the witness folding process, they can continue the folding with these shares to collectively generate the IVC proof.

To construct a collaborative folding scheme, we first identify the steps that touches the secret witnesses in Construction 12. The first one is to compute the evaluation form fo $y(t)$. This computation requires the secret witnesses from different prover clients. In the end, the prover also folds the witnesses of the two instances and this accesses the witnesses again. This final step is easy to handle in an MPC setting, as each prover client $\mathcal{C}_i$ can just fold her own secret witness.

Going back to the first step, securely computing $y(t)$ in an MPC setup can be done through classic MPC methods such as the BGW protocol [3]. As explained in Section 4.1, since $y(t)$ is a degree-D^* univariate polynomial, we can evaluate $\boldsymbol{\beta}(t)$, $\mathbf{x}(t)$, and $\mathbf{w}(t)$ at D^*+1 points $y(0), y(1), .., y(D^*)$ and then compute the corresponding circuit outputs layer by layer to obtain the evaluation form of $y(t)$ through MPC. As an optimization, exploiting the structual regularity of this computation, we can formulate it in a way that allows us to leverage *Single Instruction Multiple Data* (SIMD) in this evaluation process. In particular, the prover clients can secret share the witnesses with the servers via *Packed Secret Sharing* (PSS) [22] using a vector of polynomials $\mathbf{f}(x)$ where $\mathbf{f}(-i) = \mathbf{w}(i)$ for

$i = 0, 1, .., D^*$. Then, the servers can run an PSS-based MPC to compute the evaluation form of $y(t)$ layer by layer in a single pass.

However, there are other subtleties. In particular, if the servers in the MPC computation compute these $\{y(i)|0 \leq i \leq D^*\}$ values in the clear, extra information about the witnesses could get leaked. To address this issue, We instead propose to ask the servers to commit to the univariate polynomial $y(t)$ via MPC, and later generate evaluation proof at a random point γ securely. Committing to and generating evaluation proof for univariate polynomial $y(t)$ via MPC securely can be achieved using the techniques such as the one proposed by Garg et al. [24]. However, their approach requires a lead server which carries out heavy computations and is the efficiency bottleneck.

Remark 25 (Bulletproofs based collaborative polynomial commitment). To remove the efficiency barrier discussed above, we observe that $y(t)$ can be expressed in the inner product form $y(t) = \sum_{j=0}^{D^*} \delta(t, j) \cdot \mathbf{y}(j)$ where $\{\delta(t, j)|j = 0, 1, .., D^*\}$ are the Lagrange basis polynomials, and each of the n servers already holds additive shares of $\mathbf{y}(j)|j = 0, 1, .., D^*$. This structure enables us to use inner-product argument (IPA) based univariate polynomial commitment schemes such as Bulletproofs [13] in an MPC setting. In Bulletproofs, the core "2-to-1" random combination involves MSMs over group elements weighted by $\mathbf{y}(j)$ in the exponent. Garg et al.'s collaborative MSM protocol [24] supports exactly this setting – it allows the servers to jointly compute MSMs using only their exponent shares, without revealing them, and distributes the workload evenly among all n servers. Thus, by replacing the standard MSMs in Bulletproofs with Garg et al.'s collaborative MSMs, we can generate polynomial commitments and evaluation proofs collaboratively, with balanced computation and full witness privacy. The detailed construction is omitted here due to page limit.

Based on the above discussion, we can modify Construction 12 to a collaborative folding scheme below:

Construction 26 (Interactive Collaborative Folding Scheme (ICFS)**)** *Let us consider a finite field $\mathbb{F}$ and a succinct, hiding, and homomorphic polynomial commitment scheme for multilinear polynomials* PolyCom *over $\mathbb{F}$. We define the generator and encoder as follows:*

- $\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}$*: output size bound for $|F|$, and commitment parameters* $\mathsf{pp}_{\mathbf{w}}$*.*
- $\mathcal{K}(\mathsf{pp}, F) \rightarrow (\mathsf{pk}, \mathsf{vk})$*: output* $\mathsf{pk} \leftarrow (\mathsf{pp}, F^*)$ *and* $\mathsf{vk} \leftarrow \perp$*.*

The verifier $\mathcal{V}$ takes two committed instances $\mathsf{u}_0 = (\boldsymbol{\beta}_0, \mathbf{x}_0, \{com^j_{\mathbf{w}0}|j \in [k]\}, y_0)$ and $\mathsf{u}_1 = (\perp, \mathbf{x}_1, \{com^j_{\mathbf{w}1}|j \in [k]\}, 0)$. The set of prover clients $\mathcal{PC} = \{\mathcal{C}_j|j \in [k]\}$, in addition to the two instances, takes witnesses to both instances $(\{\mathbf{w}^j_0, r^j_{\mathbf{w}0}|j \in [k]\})$ and $(\{\mathbf{w}^j_1, r^j_{\mathbf{w}1}|j \in [k]\})$. The online protocol Π_{online} then proceeds as follows:

- $\mathcal{V}$*: Sample independent random field elements $\boldsymbol{\beta}_1 = (\beta_0, .., \beta_k, .., \beta_{\log n - 1}) \in \mathbb{F}^{\log n}$, and send them to the set of prover clients $\mathcal{PC}$ in the clear.*

- *$\mathcal{PC}$: Run MPC-based zero-knowledge univariate polynomial commitment scheme (using [24] or the Bulletproofs based scheme described above) with the servers $\mathcal{PS} = \{\mathcal{S}_i | i \in [n]\}$ to create com_y, the commitment to $y(t) = F^*(\boldsymbol{\beta}(t), \mathbf{x}(t), \mathbf{w}(t))$. $\mathcal{PC}$ then sends com_y to $\mathcal{V}$. In more details:*
  - *The prover clients $\mathcal{PC}$ secret shares the witnesses with the servers $\mathcal{PS}$;*
  - *Run MPC (e.g. the BGW protocol) to securely compute $y(2), y(3), .., y(D^*)$. In the end, each server owns a share of each of $y(2), y(3), .., y(D^*)$;*
  - *With the shares, the servers create com_y, the zero-knowledge polynomial commitment to $y(t)$ via MPC, and send com_y to the prover clients;*
  - *The prover clients relay com_y to $\mathcal{V}$.*
- *$\mathcal{V}$: Randomly sample $\gamma \in \mathbb{F}$, and send it to $\mathcal{C}$.*
- *$\mathcal{PC}$: Run MPC with the servers $\mathcal{PS}$ to evaluate $y(0)$, $y(1)$, and $y(\gamma)$, and generate the corresponding evaluation proofs π_{y0}, π_{y1}, and $\pi_{y\gamma}$. Send $(y(0), \pi_{y0})$, $(y(1), \pi_{y1})$, $(y(\gamma), \pi_{y\gamma})$ to $\mathcal{V}$* [5].
- *$\mathcal{V}$: Check if $y(0) - y_0 = 0$ and $y(1) = 0$. Then check if $y(0)$, $y(1)$, and $y(\gamma)$ are the correct evaluations of $y(t)$ at 0, 1, and γ using com_y and π_{y0}, π_{y1}, and $\pi_{y\gamma}$* [6].
- *$\mathcal{PC}$ and $\mathcal{V}$: Output the folded instance $(\boldsymbol{\beta}_\gamma, \mathbf{x}_\gamma, \{com^j_{\mathbf{w}} | j \in [k]\}, y_\gamma)$ as follows:*

$$\begin{aligned}
\boldsymbol{\beta}_\gamma &\leftarrow (1-\gamma) \cdot \boldsymbol{\beta}_0 + \gamma \cdot \boldsymbol{\beta}_1 \\
\mathbf{x}_\gamma &\leftarrow (1-\gamma) \cdot \mathbf{x}_0 + \gamma \cdot \mathbf{x}_1 \\
com^j_{\mathbf{w}\gamma} &\leftarrow (1-\gamma) \cdot com^j_{\mathbf{w}0} + \gamma \cdot com^j_{\mathbf{w}1} \mid j \in [k] \\
y_\gamma &\leftarrow y(\gamma)
\end{aligned}$$

- *$\mathcal{PC}$: The j^{th} prover client $\mathcal{C}_j$ outputs the folded witness $(\mathbf{w}^j_\gamma, r^j_{\mathbf{w}\gamma})$ to herself, where*

$$\begin{aligned}
\mathbf{w}^j_\gamma &\leftarrow (1-\gamma) \cdot \mathbf{w}^j_0 + \gamma \cdot \mathbf{w}^j_1 \\
r^j_{\mathbf{w}\gamma} &\leftarrow (1-\gamma) \cdot r^j_{\mathbf{w}0} + \gamma \cdot r^j_{\mathbf{w}1}
\end{aligned}$$

Theorem 27. *Assuming at most $m < \frac{n}{2} - D^*$ servers are corrupted, and the corrupted servers are semi-honest, Construction 26 achieves the completeness, knowledge-soundness, and m-zero-knowledge property as defined in Definition 24.*

The proof of the above theorem can be found in Appendix C.5. Moreover, note that the interaction between between $\mathcal{PC}$ and $\mathcal{V}$ is a public coin protocol. Thus it can be turned into a non-interactive protocol via the Fiat-Shamir transformation. The detailed construction is also provided in Appendix C.5.

[5] As an optimization, the servers can batch the opening proof of $y(0)$, $y(1)$, and $y(\gamma)$ into a single proof, which will reduce the verifier circuit size.

[6] These checks add five more constraints to the verifier circuit. We can treat them as five independent sub-circuits, each expected to evaluate to 0. Thus, they can all be absorbed into the zero vector check in Equation 3.

7.2 Collaborative IVC Construction

Construction 28 (Collaborative IVC CIVC)) *Based on the above building blocks, the CIVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{PC}, \mathcal{PS}, \mathcal{V})$ is defined as follows:*

- $\underline{\mathcal{G}(1^\lambda) \to \mathsf{pp}}$*: Output* $\mathsf{NIFS.G}(1^\lambda)$.
- $\underline{\mathcal{K}(\mathsf{pp}, F) \to (\mathsf{pk}, \mathsf{vk})}$*:*
  - *generate circuit* $F^* \leftarrow \mathsf{NORMALIZE}(\mathsf{pp}, F)$ *as described in Section 4.1;*
  - *generate* $(\mathsf{pk}_{\mathsf{fs}}, \mathsf{vk}_{\mathsf{fs}}) \leftarrow \mathsf{NIFS.K}(\mathsf{pp}, F^*)$*;*
  - *output* $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((\mathsf{pp}, (\mathsf{pk}_{\mathsf{fs}}, F), (\mathsf{pp}, (\mathsf{vk}_{\mathsf{fs}}, F^*))$.
- $\underline{\mathcal{PC}(\mathsf{pk}, (i, \mathbf{z}_0, \mathbf{z}_i), \{\boldsymbol{\omega}_i^j | j \in [k]\}, \Pi_i) \to \Pi_{i+1}}$*:*
  - *if* $i = 0$*, set* $(\mathsf{U}_{i+1}, \{\mathbf{W}_{i+1}^j | j \in [k]\}) \leftarrow ((\boldsymbol{\beta} = \mathbf{0}, \mathbf{x} = \mathbf{0}, \{com_{\mathbf{0}}\}^k, y = 0), \{\mathbf{w}_0^j | j \in [k]\} = \{\mathbf{0}\}^k)$*, where* $com_{\mathbf{0}}$ *is the commitment to the zero vector* $\mathbf{0}$*; Otherwise, parse* Π_i *as* $((\mathsf{U}_i, \{\mathbf{W}_i^j | j \in [k]\}), (\mathsf{u}_i, \{\mathbf{w}_i^j | j \in [k]\}), r_i)$ *and then compute* $(\mathsf{U}_{i+1}, \{\mathbf{W}_{i+1}^j | j \in [k]\}) \leftarrow \mathsf{NICFS.PC}(\mathsf{pk}, (\mathsf{U}_i, \{\mathbf{W}_i^j | j \in [k]\}), (\mathsf{u}_i, \{\mathbf{w}_i^j | j \in [k]\}))$*;*
  - *sample* $r_{i+1} \leftarrow \mathbb{F}$ *randomly;*
  - *compute* $(\mathsf{u}_{i+1}, \{\mathbf{w}_{i+1}^j | j \in [k]\}) \leftarrow \mathsf{trace}(F^*, (\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, \mathbf{z}_0, \mathbf{z}_i), \{\boldsymbol{\omega}_i^j | j \in [k]\}, r_i, r_{i+1}))$*, where* $\mathsf{u}_{i+1}.\boldsymbol{\beta} = \perp$ *and* $\mathsf{u}_{i+1}.y = 0$*; Note that* trace *is a randomized algorithm that internally samples randomness* $\{r_{\mathbf{w},i+1}^j | j \in [k]\}$ *to create hiding commitments* $\{com_{\mathbf{w},i+1}^j = \mathsf{PolyCom}(\tilde{\mathbf{w}}_{i+1}^j, r_{\mathbf{w},i+1}^j) | j \in [k]\}$ *inside* u_{i+1}*, where* $\tilde{\mathbf{w}}_{i+1}^j$ *is the multilinear extension of* $\mathbf{w}_{i+1}^j$*; If* $\mathsf{PolyCom}$ *is implemented via Pedersen's commitment, Garg et al.'s collaborative MSM approach [24] can be leveraged to offload the computation to the n servers efficiently;*
  - *output* $\Pi_{i+1} \leftarrow ((\mathsf{U}_{i+1}, \{\mathbf{W}_{i+1}^j | j \in [k]\}), (\mathsf{u}_{i+1}, \{\mathbf{w}_{i+1}^j | j \in [k]\}), r_{i+1})$.
- $\underline{\mathcal{V}(\mathsf{vk}, (i, \mathbf{z}_0, \mathbf{z}_i), \Pi_i) \to \{0, 1\}}$*:*
  - *if* $i = 0$*, check if* $\mathbf{z}_i = \mathbf{z}_0$*;*
  - *otherwise,*
    - *parse* Π_i *as* $((\mathsf{U}_i, \{\mathbf{W}_i^j | j \in [k]\}), (\mathsf{u}_i, \{\mathbf{w}_i^j | j \in [k]\}), r_i)$,
    - *check if* $\mathsf{u}_i.y = 0$ *and if* $\mathsf{u.x} = \mathsf{hash}(\mathsf{vk}, i, \mathbf{z}_0, \mathbf{z}_i, \mathsf{U}_i, r_i)$,
    - *randomly samples* $\boldsymbol{\beta}_i \in \mathbb{F}^{\log n}$ *and set* $\mathsf{u}_i.\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_i$,
    - *check if* $\{\mathbf{W}_i^j | j \in [k]\}$ *and* $\{\mathbf{w}_i^j | j \in [k]\}$ *are satisfying witnesses to* U_i *and* u_i *respectively using* $\mathsf{vk.pp}$ *and* $\mathsf{vk}.F^*$.

It is worth pointing out that the special case where $\mathcal{PC}$ contains a single prover client is similar to the zkSaaS setup [24] which allows a resource-limited prover to outsource the IVC prover generation to n servers. Moreover, as Section 5.2 discussed, the IVC proof generated in the above protocol can be compressed using a second-stage SNARK protocol. For collaborative IVC, this compression can also be implemented collaboratively, for example, by using the collaborative GKR/Libra-based zk-SNARKs protocol proposed by Liu et al. [36].

References

1. Arora, S., Safra, S.: Probabilistic checking of proofs; A new characterization of NP. In: 33rd FOCS. pp. 2–13. IEEE Computer Society Press (Oct 1992)
2. Arun, A., Setty, S., Thaler, J.: Jolt: SNARKs for virtual machines via lookups. Cryptology ePrint Archive, Paper 2023/1217 (2023), https://eprint.iacr.org/2023/1217
3. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. p. 1–10. STOC '88, Association for Computing Machinery, New York, NY, USA (1988), https://doi.org/10.1145/62212.62213
4. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (Oct / Nov 2016)
5. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 276–294. Springer, Heidelberg (Aug 2014)
6. Berrut, J.P., Trefethen, L.N.: Barycentric lagrange interpolation. SIAM Review 46(3), 501–517 (2004), https://doi.org/10.1137/S0036144502417715
7. Boneh, D., Chen, B.: LatticeFold: A lattice-based folding scheme and its applications to succinct proof systems. Cryptology ePrint Archive, Paper 2024/257 (2024), https://eprint.iacr.org/2024/257
8. Boneh, D., Chen, B.: LatticeFold+: Faster, simpler, shorter lattice-based folding for succinct proof systems. Cryptology ePrint Archive, Paper 2025/247 (2025), https://eprint.iacr.org/2025/247
9. 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
10. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. 37(2), 156–189 (oct 1988), https://doi.org/10.1016/0022-0000(88)90005-0
11. Bünz, B., Chen, B.: Protostar: Generic efficient accumulation/folding for special-sound protocols. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology – ASIACRYPT 2023. pp. 77–110. Springer Nature Singapore, Singapore (2023)
12. Bünz, B., Chiesa, A., Lin, W., Mishra, P., Spooner, N.: Proof-carrying data without succinct arguments. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 681–710. Springer, Heidelberg, Virtual Event (Aug 2021)
13. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 315–334 (2018)
14. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. Cryptology ePrint Archive, Paper 2022/1355 (2022), https://eprint.iacr.org/2022/1355, https://eprint.iacr.org/2022/1355
15. Chiesa, A., Forbes, M.A., Spooner, N.: A zero knowledge sumcheck and its applications. Cryptology ePrint Archive, Report 2017/305 (2017), https://eprint.iacr.org/2017/305
16. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai,

Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 738–768. Springer, Heidelberg (May 2020)
17. Cormode, G., Mitzenmacher, M., Thaler, J.: Practical verified computation with streaming interactive proofs. In: Goldwasser, S. (ed.) ITCS 2012. pp. 90–112. ACM (Jan 2012)
18. Dimitriou, N., Garreta, A., Manzur, I., Vlasov, I.: Mova: Nova folding without committing to error terms. Cryptology ePrint Archive, Paper 2024/1220 (2024), https://eprint.iacr.org/2024/1220
19. Eagen, L., Gabizon, A.: ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances. Cryptology ePrint Archive, Paper 2023/1106 (2023), https://eprint.iacr.org/2023/1106
20. Fenzi, G., Knabenhans, C., Nguyen, N.K., Pham, D.T.: Lova: Lattice-based folding scheme from unstructured lattices. Cryptology ePrint Archive, Paper 2024/1964 (2024), https://eprint.iacr.org/2024/1964
21. 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. Springer, Heidelberg (Aug 1987)
22. Franklin, M., Yung, M.: Communication complexity of secure computation (extended abstract). In: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing. p. 699–710. STOC '92, Association for Computing Machinery, New York, NY, USA (1992), https://doi.org/10.1145/129712.129780
23. Gabizon, A., Williamson, Z.J., Ciobotaru, O.M.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch. 2019, 953 (2019)
24. Garg, S., Goel, A., Jain, A., Policharla, G.V., Sekar, S.: zksaas: zero-knowledge snarks as a service. In: Proceedings of the 32nd USENIX Conference on Security Symposium. SEC '23, USENIX Association, USA (2023)
25. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: Interactive proofs for muggles. J. ACM 62(4) (sep 2015), https://doi.org/10.1145/2699436
26. Golovnev, A., Lee, J., Setty, S., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic snarks for r1cs. In: Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part II. p. 193–226. Springer-Verlag, Berlin, Heidelberg (2023), https://doi.org/10.1007/978-3-031-38545-2_7
27. Haböck, U.: Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Paper 2022/1530 (2022), https://eprint.iacr.org/2022/1530
28. 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. Springer, Heidelberg (Dec 2010)
29. Kothapalli, A., Parno, B.: Algebraic reductions of knowledge. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023. pp. 669–701. Springer Nature Switzerland, Cham (2023)
30. Kothapalli, A., Setty, S.: SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Paper 2022/1758 (2022), https://eprint.iacr.org/2022/1758
31. Kothapalli, A., Setty, S.: Hypernova: Recursive arguments for customizable constraint systems. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024. pp. 345–379. Springer Nature Switzerland, Cham (2024)
32. Kothapalli, A., Setty, S.: NeutronNova: Folding everything that reduces to zero-check. Cryptology ePrint Archive, Paper 2024/1606 (2024), https://eprint.iacr.org/2024/1606

33. 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. Springer, Heidelberg (Aug 2022)
34. Lee, J.: Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In: Nissim, K., Waters, B. (eds.) Theory of Cryptography. pp. 1–34. Springer International Publishing, Cham (2021)
35. Liu, T., Xie, X., Zhang, Y.: zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2968–2985. ACM Press (Nov 2021)
36. Liu, X., Zhou, Z., Wang, Y., Pang, Y., He, J., Zhang, B., Yang, X., Zhang, J.: Scalable collaborative zk-SNARK and its application to fully distributed proof delegation. Cryptology ePrint Archive, Paper 2024/940 (2024), https://eprint.iacr.org/2024/940
37. Liu, X., Zhou, Z., Wang, Y., Pang, Y., He, J., Zhang, B., Yang, X., Zhang, J.: Scalable collaborative zk-SNARK and its application to fully distributed proof delegation. USENIX Association, USA (2025)
38. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM 39(4), 859–868 (oct 1992), https://doi.org/10.1145/146585.146605
39. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2111–2128. ACM Press (Nov 2019)
40. Mohnblatt, N.: Sangria: a folding scheme for plonk (2023), https://github.com/geometryxyz/technical_notes/blob/main/sangria_folding_plonk.pdf
41. Ozdemir, A., Boneh, D.: Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. Cryptology ePrint Archive, Paper 2021/1530 (2021), https://eprint.iacr.org/2021/1530
42. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 222–242. Springer, Heidelberg (Mar 2013)
43. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC. pp. 49–62. ACM Press (Jun 2016)
44. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. J. ACM 27(4), 701–717 (oct 1980), https://doi.org/10.1145/322217.322225
45. 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. Springer, Heidelberg (Aug 2020)
46. Setty, S., Thaler, J., Wahby, R.: Customizable constraint systems for succinct arguments. Cryptology ePrint Archive, Paper 2023/552 (2023), https://eprint.iacr.org/2023/552
47. Setty, S., Thaler, J., Wahby, R.: Unlocking the lookup singularity with lasso. In: Joye, M., Leander, G. (eds.) Advances in Cryptology – EUROCRYPT 2024. pp. 180–209. Springer Nature Switzerland, Cham (2024)
48. Thaler, J.: Proofs, arguments, and zero-knowledge https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html
49. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) Theory of Cryptography. pp. 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)

50. Wahby, R.S., Tzialla, I., shelat, a., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: 2018 IEEE Symposium on Security and Privacy. pp. 926–943. IEEE Computer Society Press (May 2018)
51. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 733–764. Springer, Heidelberg (Aug 2019)
52. Zhang, J., Liu, T., Wang, W., Zhang, Y., Song, D., Xie, X., Zhang, Y.: Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 159–177. ACM Press (Nov 2021)
53. Zhang, Y.X., Vark, A.: Origami - a folding scheme for halo2 lookups (2023), `https://hackmd.io/@aardvark/rkHqa3NZ2`
54. Zhang, Y.: New (zero-knowledge) arguments and their applications to verifiable computation. Ph.D. Thesis (2018), `https://www.cs.umd.edu/~jkatz/THESES/yupeng.pdf`
55. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In: 2017 IEEE Symposium on Security and Privacy. pp. 863–880. IEEE Computer Society Press (May 2017)
56. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: Ng, E.W. (ed.) Symbolic and Algebraic Computation. pp. 216–226. Springer Berlin Heidelberg, Berlin, Heidelberg (1979)

A Additional Preliminaries

A.1 IP, IOP, and Polynomial IOP

Interactive Proof and Argument. An interactive proof (IP) or argument is a protocol that runs several rounds between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. In each round, $\mathcal{V}$ is allowed to ask questions based on $\mathcal{P}$' answers from previous rounds. In the context of circuit satisfiability, we are given a circuit F with a public input vector $\mathbf{x}$ and an expected output vector $\mathbf{y}$. The circuit F also takes in a witness vector $\mathbf{w}$ private to $\mathcal{P}$, and $\mathcal{P}$ tries to convince $\mathcal{V}$ that $F(\mathbf{x}, \mathbf{w}) = \mathbf{y}$ through the interactions. We formalize interactive proofs for the circuit satisfiability problem in the following [14]:

Definition 29 (Interactive Proof of Knowledge for Circuit Satisfiability). *Consider a pair of prover and verifier $\mathcal{P}, \mathcal{V}$. Both $\mathcal{P}$ and $\mathcal{V}$ are given an arithmetic circuit F, a public input vector $\mathbf{x}$, and an output vector $\mathbf{y}$. In addition, the prover has access to a private witness vector $\mathbf{w}$. For a given security parameter λ, there is a setup process which outputs prover parameter pp and verifier parameter vp, i.e. $(pp, vp) \leftarrow \mathsf{Setup}(1^\lambda)$. $\mathcal{P}$ and $\mathcal{V}$ exchange a sequence of messages π, and then $\mathcal{V}$ outputs a Boolean bit accept$(\mathcal{P}(pp, \mathbf{w}), \mathcal{V}(vp), \pi)$. We call the interactive protocol $\Pi = (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ between $\mathcal{P}$ and $\mathcal{V}$ an interactive proof of knowledge for the satisfiability of the circuit F if the following holds:*

- ***Perfect Completeness**. For every $(pp, vp) \leftarrow \mathsf{Setup}(1^\lambda)$, and every tuple of vectors $(\mathbf{x}, \mathbf{w}, \mathbf{y})$ such that $F(\mathbf{x}, \mathbf{w}) = \mathbf{y}$*

$$\Pr\left[accept(\mathcal{P}(pp,\mathbf{w}),\mathcal{V}(vp),\pi)=1 \,\middle|\, \begin{matrix}(pp,vp)\leftarrow \mathsf{Setup}(1^\lambda)\\ F(\mathbf{x},\mathbf{w})=\mathbf{y}\end{matrix}\right]=1.$$

- δ***-Soundness***. *We call interactive proof Π δ-sound for $\delta<1$ if for any (adversarial) prover $\mathcal{P}^*$ and the sequence of messages π^* between $\mathcal{P}^*$ and $\mathcal{V}$, the following holds*

$$\Pr\left[\begin{matrix}accept(\mathcal{P}^*(pp,\mathbf{w}),\mathcal{V}(vp),\pi^*)=1\\ \wedge\\ F(\mathbf{x},\mathbf{w})\neq\mathbf{y}\end{matrix} \,\middle|\, (pp,vp)\leftarrow \mathsf{Setup}(1^\lambda)\right]\leq\delta.$$

  We say Π is statistically sound if $\mathcal{P}^*$ is unbounded, and δ is negligible in terms of λ, i.e., $\delta\leq\mathsf{negl}(\lambda)$. We say Π is computationally sound if $\mathcal{P}^*$ is a probabilistic polynomial time (PPT) adversarial prover and $\delta\leq\mathsf{negl}(\lambda)$.
- δ***-Knowledge Soundness***. *For any PPT adversarial prover $\mathcal{P}^*$, there exists a PPT algorithm $\mathcal{E}$ called the extractor such that given the access to the entire executing process and randomness of $\mathcal{P}^*$, $\mathcal{E}$ can extract a witness vector $\mathbf{w}$ such that the following holds:*

$$\Pr\left[\begin{matrix}accept(\mathcal{P}^*(pp,\mathbf{w}),\mathcal{V}(vp),\pi^*)=1\\ \wedge\\ F(\mathbf{x},\mathbf{w})\neq\mathbf{y}\end{matrix} \,\middle|\, \begin{matrix}(pp,vp)\leftarrow \mathsf{Setup}(1^\lambda)\\ \mathbf{w}\leftarrow\mathcal{E}^{\mathcal{P}^*}(pp,\mathbf{x},\mathbf{y},\pi^*)\end{matrix}\right]\leq\delta.$$

  An interactive protocol is "knowledge sound", or simply an "argument of knowledge", if the knowledge error $\delta\leq\mathsf{negl}(\lambda)$. If the adversary is unbounded, then the argument is called an interactive proof of knowledge.
- ***Public Coin***. *An interactive protocol is considered to be public coin if all the verifier messages including the final output can be computed as a deterministic function given a random public input.*
- ***Zero-Knowledge***. *An interactive protocol has the zero-knowledge property if there exists a PPT simulator algorithm $\mathcal{S}$ such that for any PPT algorithm $\mathcal{V}^*$, $(pp,vp)\leftarrow\mathsf{Setup}(1^\lambda)$, auxiliary input $\mathbf{z}\in\{0,1\}^{\mathsf{poly}(\lambda)}$, it holds that the following two probability distributions are indistinguishable:*

$$\mathbf{View}(\langle\mathcal{P}(pp,w),\mathcal{V}^*(vp,\mathbf{z})\rangle(F,\mathbf{x},\mathbf{y}))\approx\mathcal{S}^{\mathcal{V}^*}(F,\mathbf{x},\mathbf{y},\mathbf{z})$$

  where $\mathbf{View}(\langle\mathcal{P}(pp,w),\mathcal{V}^*(vp,\mathbf{z})\rangle(F,\mathbf{x},\mathbf{y}))$ denotes the distribution of transcripts $\mathcal{V}^*$ sees when interacting with an honest prover, and $\mathcal{S}^{\mathcal{V}^*}(F,\mathbf{x},\mathbf{y},\mathbf{z})$ represents the distribution of the output of the simulator $\mathcal{S}$.

Interactive Oracle Proof. We define an interactive oracle proof (IOP) [4, 43] as an interactive proof where in each round, the prover sends a string as an oracle, and the verifier is not required to read through the prover's messages. Instead, the verifier has oracle access to the prover's messages and can make probabilistic queries to them. Interactive oracle proof can be viewed as a hybrid of interactive proof and probabilistically checkable proofs (PCP) [1].

Polynomial Interactive Oracle Proof. In some of the communication steps of an IOP protocol, the prover sends a polynomial as an oracle, and the verifier query may request an evaluation of the polynomial at a point in its domain. We call such a protocol a Polynomial Interactive Oracle Proof (Polynomial IOP). Below, we formalize the concept of Polynomial IOP:

Definition 30. *A polynomial interactive oracle proof is a public-coin interactive oracle proof protocol where, in one or more rounds, the message sent from $\mathcal{P}$ to $\mathcal{V}$ is an oracle to a μ-variate polynomial f over a field of choice $\mathbb{F}$. These oracles specify μ and the degree in each variable, and they can be queried by $\mathcal{V}$ at arbitrary points in $\mathbb{F}^\mu$ to evaluate the polynomial at these points.*

Many modern SNARKs are constructed by combining a polynomial IOP with a cryptographic primitive called polynomial commitment schemes [28]. This combination yields a succinct interactive argument, which can then be made non-interactive via the Fiat-Shamir transformation [21], resulting in a SNARK [48].

A.2 Polynomial Interpolation

Lagrange interpolation. Given a set of ν distinct field elements $\{t_1, t_2, ..., t_\nu\}$, the univariate *Lagrange basis* for polynomials of degree $\leq \nu - 1$ is a set of polynomials $\{\delta(t, t_1), \delta(t, t_2), ..., \delta(t, t_\nu)\}$ each of degree $\nu - 1$, such that

$$\delta(t, t_i) = \begin{cases} 1, & t = t_i \\ 0, & \forall\ t \neq t_i, 1 \leq i \leq \nu \end{cases}$$

We can express $\delta(t, t_i)$ explicitly as follows:

$$\delta(t, t_i) = \prod_{1 \leq j \leq \nu,\ j \neq i} \frac{t - t_j}{t_i - t_j}, \text{ for } i = 1, 2, ...\nu \tag{7}$$

Then, given ν points $\{(t_1, y_1)(t_2, y_2), ..., (t_\nu, y_\nu)\}$, the *unique* univariate polynomial with degree $\leq \nu - 1$ passing through these points can be written as:

$$L(t) = \sum_{i=1}^{\nu} y_i \delta(t, t_i) \tag{8}$$

In many applications, the evaluation points $\{t_i\}$ are simply integers, for example, $t_1 = 1, t_2 = 2, .., t_\nu = \nu$. Another popular choice is to let $t_i = \omega^{i-1}$ where ω is the ν^{th} root of the unity of field $\mathbb{F}$. This choice can sometimes simplify calculations.

Multilinear extension. A multilinear function is a function of multiple variables that is linear separately in each variable. Given a function $f : \mathbb{F}^\mu \to \mathbb{F}$ and its evaluations over the Boolean hypercube $\{0, 1\}^\mu$, its multilinear extension $\tilde{f}(\mathbf{t})$ is a multilinear function which agrees with $f(\mathbf{t})$ for any $\mathbf{t} \in \{0, 1\}^\mu$ [48]. To derive the explicit expression of the multilinear expression for $f(\cdot)$, let us first define the *identity function*:

$$\tilde{\mathsf{eq}}(\mathbf{x}, \mathbf{t}) = \prod_{k=1}^{\mu} (t_k \cdot x_k + (1 - t_k) \cdot (1 - x_k)) \tag{9}$$

where $\mathbf{x} \in \mathbb{F}^\mu$ is a vector of variables, and t_k is the k^{th} element of $\mathbf{t} \in \{0,1\}^\mu$. A key property of the identity function is that $\tilde{\mathsf{eq}}(\mathbf{x}, \mathbf{t}) = 1$ if $\mathbf{x}, \mathbf{t} \in \{0,1\}^\mu$, and $\mathbf{x} = \mathbf{t}$. Otherwise, $\tilde{\mathsf{eq}}(\mathbf{x}, \mathbf{t}) = 0$. Using the property, we can derive the explicit expression of $\tilde{f}(\cdot)$ [14]:

Lemma 31 (Multilinear extensions (MLE)). *For every function $f : \{0,1\}^\mu \to \mathbb{F}$, there is a unique multilinear polynomial $\tilde{f} : \mathbb{F}^\mu \to \mathbb{F}$ such that $\tilde{f}(\mathbf{t}) = f(\mathbf{t})$ for all $\mathbf{t} \in \{0,1\}^\mu$. We call $\tilde{f}$ the multilinear extension of f, and $\tilde{f}$ can be explicitly expressed as $\tilde{f}(\mathbf{x}) = \sum_{\mathbf{t} \in \{0,1\}^\mu} \tilde{\mathsf{eq}}(\mathbf{x}, \mathbf{t}) \cdot f(\mathbf{t})$.*

A.3 Polynomial Evaluation

Univariate polynomials have two equivalent forms of representation, the coefficient form which uses the monomial basis, and the point-value form which uses the Lagrange basis.

Single point evaluation. Given a univariate polynomial $L(t)$, and an evaluation point $r \in \mathbb{F}$, our goal is to calculate $L(r)$.

- **Coefficient form.** Horner's method applies to the coefficient form $L(t) = a_0 + a_1 t + a_2 t^2 + ... + a_\nu t^\nu$. The trick is to rewrite the polynomial as $L(t) = a_0 + t(a_1 + t(a_2 + t(a_3 + ... + t(a_{\nu-1} + t a_\nu)...)))$. This allows us to compute $L(r)$ from the innermost parentheses with $O(\nu)$ field multiplications and additions.
- **Point-value form.** The *barycentric interpolation/evaluation* method [6] applies to the point-value form $L(t) = \sum_{i=0}^{\nu-1} y_i \delta(t, t_i)$. Let us use $M(t)$ to denote the product $\prod_{0 \le j \le \nu-1} (t - t_j)$. Let $d_i = \prod_{0 \le j \le \nu-1, j \ne i} (t_i - t_j)$. We can rewrite $L(t)$ as $L(t) = M(t) \sum_{i=0}^{\nu-1} \frac{y_i}{d_i (t - t_i)}$. It is shown that provided $\{(0, y_0), (1, y_1), .., (i, y_i), .., (\nu, y_\nu)\}$, we can calculate $L(r)$ for a given $r \in \mathbb{F}$ with $O(\nu)$ field multiplication and additions [6].

A.4 Polynomial Commitment Schemes

At a high level, a polynomial commitment scheme allows a prover to compute a commitment to a polynomial [16, 28, 39, 42, 45, 50, 54, 55]. The commitment can later be "opened" at any evaluation point. As an example, to prove the committed univariate polynomial $f(x)$ is evaluated to y_r at the point r, the prover presents the claimed value of $f(r)$ and an associated opening proof π_r to the verifier. The verifier can then check the claimed value and proof against the commitment *com*, and decide if the claimed value is correct. More formally, a polynomial commitment scheme (for multivariate polynomials) can be defined as [28, 42, 54]:

Definition 32. *Let $\mathbb{F}$ be a finite field. $\mathcal{F}$ is a family of μ-variate polynomials over $\mathbb{F}$, and ν is a variable degree parameter. $(KeyGen, Commit, Open, Verify)$ form an extractable polynomial commitment scheme for $\mathcal{F}$ if the following holds:*

- ***Perfect Completeness**. For any polynomial $f \in \mathcal{F}$ it holds that*

$$\Pr\left[\begin{array}{l|l} Verify(com, \mathbf{t}, y, \pi, vp) = 1 & (pp, vp) \leftarrow KeyGen(1^\lambda, \mu, \nu) \\ \wedge & com \leftarrow Commit(f, pp) \\ y = f(\mathbf{t}) & (y, \pi) \leftarrow Open(f, \mathbf{t}, pp) \end{array}\right] = 1.$$

- ***Soundness**. For any PPT adversary $\mathcal{A}$, the following probability is negligible:*

$$\Pr\left[\begin{array}{l|l} Verify(com, \mathbf{t}^*, y^*, \pi^*, vp) = 1 & (pp, vp) \leftarrow KeyGen(1^\lambda, \mu, \nu) \\ \wedge & com \leftarrow Commit(f^*, pp) \\ y^* \neq f^*(\mathbf{t}^*) & (f^*, \mathbf{t}^*, y^*, \pi^*) \leftarrow \mathcal{A}(1^\lambda, pp) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

- ***Extractability**. For any PPT adversary $\mathcal{A}$, there exists a polynomial time algorithm $\mathcal{E}$ with access to $\mathcal{A}$'s randomness such that for all benign auxiliary inputs $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$ the following probability is negligible:*

$$\Pr\left[\begin{array}{l|l} CheckCom(com^*, vp) = 1 & (pp, vp) \leftarrow KeyGen(1^\lambda, \mu, \nu) \\ \wedge & f' \leftarrow \mathcal{E}(1^\lambda, pp, z) \\ com^* \neq Commit(f', pp) & com* \leftarrow \mathcal{A}(1^\lambda, pp, z) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

A.5 More on the Sumcheck Protocol

The Sumcheck protocol is a protocol to prove the following claim:

$$\sum_{b_1, b_2, .., b_\mu \in \{0,1\}} f(b_1, b_2, .., b_\mu) = H$$

where $f : \mathbb{F}^\mu \to \mathbb{F}$ is a multivariate polynomial and H is the claimed sum of the evaluation of f over the Boolean hypercube $\{0,1\}^\mu$. Since there are $N = 2^\mu$ combinations of $b_1, .., b_\mu$, directly calculating the sum could require $O(N)$ time, The Sumcheck protocol delegates the computation to a potentially unbounded prover, and proves that the claimed sum H is computed correctly. We present the details of the Sumcheck protocol in Protocol 1.

Protocol 1 (Sumcheck). *The protocol proceeds in μ rounds.*

- *In the 1^{st} round, $\mathcal{P}$ sends $\mathcal{V}$ a univariate polynomial*

$$f_1(x_1) = \sum_{b_2,..,b_\mu \in \{0,1\}} f(x_1, b_2, .., b_\mu)$$

  $\mathcal{V}$ checks $H = f_1(0) + f_1(1)$. Then $\mathcal{V}$ sends a random challenge $r_1 \in \mathbb{F}$ to $\mathcal{P}$.

- *For round $i \in [2, 3, .., \mu - 1]$, $\mathcal{P}$ sends $\mathcal{V}$ a univariate polynomial*

$$f_i(x_i) = \sum_{b_{i+1},..,b_n \in \{0,1\}} f(r_1, .., r_{i-1}, x_i, b_{i+1}, .., b_\mu)$$

  $\mathcal{V}$ checks $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$, and sends a random $r_i \in \mathbb{F}$ to $\mathcal{P}$.

- *In the μ^{th} round, $\mathcal{P}$ sends $\mathcal{V}$ a univariate polynomial*

$$f_\mu(x_\mu) = f(r_1, .., r_{\mu-1}, x_\mu)$$

  $\mathcal{V}$ checks $f_{\mu-1}(r_{\mu-1}) = f_\mu(0) + f_\mu(1)$. $\mathcal{V}$ then samples a random challenge $r_\mu \in \mathbb{F}$. Next, $\mathcal{V}$ queries the oracle $[[f]]$ to obtain the evaluation $f(r_1, r_2, .., r_\mu)$. $\mathcal{V}$ accepts if and only if $f_\mu(r_\mu) = f(r_1, r_2, .., r_\mu)$.

The Sumcheck protocol is complete and sound with soundness error $\delta = \frac{\nu\mu}{|\mathbb{F}|} = \frac{\nu \log N}{|\mathbb{F}|}$, where ν is the maximum degree of the individual variables of f [35, 38, 48]. Its proof size is $O(\nu\mu) = O(\nu \log N)$. This is because the entire protocol has $\mu = \log N$ rounds and in each round, the prover $\mathcal{P}$ sends a univariate polynomial which can be uniquely determined by $\nu + 1$ points. The verifier time over the entire execution of the Sumcheck protocol is proportional to the total communication, plus the cost of a single oracle query to $[[f]]$ to compute $f(r_1, r_2, .., r_\mu)$. Thus, the verifier time is $O(\nu \log N + T)$, where T is the time complexity of one query to the oracle $[[f]]$ [48]. The instantiation of the oracle access to $[[f]]$ depends on the application of the Sumcheck protocol. In some applications, the verifier might be able to query other parties, for example, the prover to obtain the evaluation results. In other applications, the verifier may have sufficient data to compute $f(r_1, r_2, .., r_\mu)$ directly on her own.

Analyzing the prover time is a bit more involved, as it depends on various factors, including the degree and the sparsity of f. In general, the prover time can be bounded by $O(N\nu T)$ [48]. However, for some highly important cases, the prover time can be further optimized. In particular, we have the following lemma (adapted from Lemma 4.5 in Chapter 4 of [48]):

Lemma 33. *Let p_1, p_2, .., p_k be μ-variate multilinear polynomials. Suppose that for each p_i there is an algorithm that evaluates p_i at all inputs in $\{0,1\}^\mu$ in $O(2^\mu)$ field multiplications or additions. Let $f = p_1 \cdot p_2 \cdots p_k$ be the product of these multilinear polynomials. Then, when the Sumcheck protocol is applied to polynomial f, the honest prover can be implemented in $O(k \cdot 2^\mu)$ field multiplications or additions.*

For a polynomial f that satisfies the conditions specified in the above lemma, the prover time can be reduced to $O(k \cdot 2^{\mu}) = O(kN)$. If k is in $O(1)$, the prover time would be linear in terms of N.

A.6 More on the GKR Protocol

As briefly discussed in Section 2.2, the prover $\mathcal{P}$ and the the the verifier $\mathcal{V}$ can use the GKR protocol to prove the correctness of a circuit evaluation. The protocol processes the circuit under consideration layer by layer, from the output all the way to the input. In each round, $\mathcal{P}$ and $\mathcal{V}$ run a Sumcheck protocol to reduce the correctness of the evaluation of the current layer to the next layer. Finally, at the input layer, the verifier queries the oracle to determine whether to accept or reject the claimed circuit output.

To formally describe the GKR protocol, we first introduce some notation. We denote the number of gates in the i^{th} layer by N_i. Without loss of generality, we also assume that N_i is a power of two. We let $\mu_i = \log N_i$. Then, we can define a function $v_i : \{0,1\}^{\mu_i} \to \mathbb{F}$ that returns the output of the gate b in layer i where $b \in \{0,1\}^{\mu_i}$ is a binary string. We call b the label of the gate. With our layer indexing convention, v_d corresponds to the input layer and v_0 corresponds to the output layer. Next, we introduce three functions $add_i, sub_i, mult_i : \{1,0\}^{\mu_{i-1}+2\mu_i} \to \{0,1\}$ for the addition, subtraction, and multiplication gates. These are referred to as *wiring predicates* in the literature. With these definitions, we have the following formula to express v_i in terms of v_{i+1}:

$$\begin{aligned} v_i(z) = \sum_{x,y \in \{0,1\}^{\mu_{i+1}}} & (add_{i+1}(z,x,y)(v_{i+1}(x) + v_{i+1}(y)) \\ & + sub_{i+1}(z,x,y)(v_{i+1}(x) - v_{i+1}(y)) \\ & + mult_{i+1}(z,x,y)(v_{i+1}(x)v_{i+1}(y))) \end{aligned} \tag{10}$$

for any $z \in \{0,1\}^{\mu_i}$. Note that in the above, v_i is expressed as a summation over a Boolean hypercube. Hence, $\mathcal{P}$ and $\mathcal{V}$ can execute a Sumcheck protocol to verify if the layer is computed correctly. Since the Sumcheck protocol requires the function for the summation to be a polynomial, we can rewrite the above equation using multilinear extensions:

$$\begin{aligned} \tilde{v}_i(g) &= \sum_{x,y \in \{0,1\}^{\mu_{i+1}}} f_i(x,y) \\ &= \sum_{x,y \in \{0,1\}^{\mu_{i+1}}} (\tilde{add}_{i+1}(g,x,y)(\tilde{v}_{i+1}(x) + \tilde{v}_{i+1}(y)) \\ &\qquad + \tilde{sub}_{i+1}(g,x,y)(\tilde{v}_{i+1}(x) - \tilde{v}_{i+1}(y)) \\ &\qquad + \tilde{mult}_{i+1}(g,x,y)(\tilde{v}_{i+1}(x)\tilde{v}_{i+1}(y))) \end{aligned} \tag{11}$$

where $g \in \mathbb{F}^{\mu_i}$ is a random vector.

With the above equation derived, we can present the details of the GKR protocol in Protocol 2. In the beginning, $\mathcal{P}$ sends the claimed outputs $\mathbf{v}_0$ to $\mathcal{V}$. $\mathcal{V}$ then derives $\tilde{v}_0$ and computes $\tilde{v}_0(g)$ for a random $g \in \mathbb{F}^{\mu_0}$. Next, $\mathcal{P}$ and $\mathcal{V}$ run a Sumcheck protocol on Equation (11) starting from $i = 0$. In the last step of the Sumcheck protocol for layer i, $\mathcal{V}$ needs an oracle query for $f_i(\rho, \eta)$, where ρ and η are randomly generated vectors in $\mathbb{F}^{\mu_{i+1}}$. According to Equation (11), computing $f_i(\rho, \eta)$ requires $\tilde{add}_{i+1}(g, x, y)$, $\tilde{sub}_{i+1}(g, x, y)$, $\tilde{mult}_{i+1}(g, x, y)$, as well as $\tilde{v}_{i+1}(\rho)$ and $\tilde{v}_{i+1}(\eta)$. We note that $\tilde{add}_{i+1}(g, x, y)$, $\tilde{sub}_{i+1}(g, x, y)$ and $\tilde{mult}_{i+1}(g, x, y)$ can be computed by $\mathcal{V}$ locally, since they only depend on the circuit wiring pattern, and not the values of the wires. On the other hand, $\mathcal{V}$ can ask $\mathcal{P}$ to send her $\tilde{v}_{i+1}(\rho)$ and $\tilde{v}_{i+1}(\eta)$. With all these values, $\mathcal{V}$ can finally calculate $f_i(\rho, \eta)$. This effectively reduces the claim about layer i to two claims about layer $i + 1$. Thus, the claim about the output layer can eventually be reduced to claims about the input layer, where $\mathcal{V}$ has access to all input values to verify the claims herself.

Protocol 2 (GKR). *Given prime field $\mathbb{F}$, let F be a layered arithmetic circuit with depth d. $\mathcal{P}$ claims that $F(\mathbf{v}_d) = \mathbf{v}_0$, where $\mathbf{v}_d$ and $\mathbf{v}_0$ are the input and output vector of the circuit, respectively. We denote the multilinear extension of $\mathbf{v}_d$ and $\mathbf{v}_0$ by $\tilde{v}_d$ and $\tilde{v}_0$, respectively.*

- *$\mathcal{V}$ samples a random $g \in \mathbb{F}^{\mu_0}$ and send it to $\mathcal{P}$. Then, both party compute $\tilde{v}_0(g)$.*
- *$\mathcal{P}$ and $\mathcal{V}$ run a Sumcheck protocol on polynomial $f_0(x, y)$ as defined in Equation (11) for $i = 0$. If the Sumcheck fails, $\mathcal{V}$ rejects the claimed outputs.*
- *For $i \in [1, 2, ..., d-1]$ do*
  - *$\mathcal{V}$ randomly sample $\alpha_i, \beta_i \in \mathbb{F}$ and send them to $\mathcal{P}$.*
  - *$\mathcal{P}$ and $\mathcal{V}$ runs the Sumcheck protocol on $\alpha_i \tilde{v}_i(\rho) + \beta_i \tilde{v}_i(\eta)$ as defined in Equation (12).*
  - *At the end of the Sumcheck protocol, $\mathcal{P}$ sends $\mathcal{V}$ $\tilde{v}_{i+1}(\rho_{i+1})$ and $\tilde{v}_{i+1}(\eta_{i+1})$. Based on these values, $\mathcal{V}$ check if Equation (12) holds for layer i. If the check fails, $\mathcal{V}$ rejects the claimed outputs.*
- *At the input layer d, $\mathcal{V}$ has two claims $\tilde{v}_d(\rho_d)$ and $\tilde{v}_d(\eta_d)$. Based on the inputs, $\mathcal{V}$ evaluates $\tilde{v}_d$ at ρ_d and η_d and checks if they are the same as the two claimed values. If so, $\mathcal{V}$ accepts the claimed output. Otherwise, $\mathcal{V}$ rejects the claimed output.*

However, since a claim about layer i translates into two claims about layer $i+1$, which in turn requires the execution of two Sumcheck protocols, the number of Sumcheck protocols $\mathcal{P}$ and $\mathcal{V}$ that must run could grow exponentially with d, the depth of the circuit. To address this issue, Chiesa et al. proposed to combine the two claims $\tilde{v}_i(\rho)$ and $\tilde{v}_i(\eta)$ into one using a random linear combination [15]. To be more specific, $\mathcal{V}$ randomly selects $\alpha_i, \beta_i \in \mathbb{F}$ and computes $\alpha_i \tilde{v}_i(\rho) + \beta_i \tilde{v}_i(\eta)$ as follows:

$$
\begin{aligned}
&\alpha_i \tilde{v}_i(\rho) + \beta_i \tilde{v}_i(\eta) \\
&= \sum_{x,y \in \{0,1\}^{\mu_{i+1}}} ((\alpha_i \tilde{add}_{i+1}(\rho, x, y) + \beta_i \tilde{add}_{i+1}(\eta, x, y))(\tilde{v}_{i+1}(x) + \tilde{v}_{i+1}(y)) \\
&\qquad + (\alpha_i \tilde{sub}_{i+1}(\rho, x, y) + \beta_i \tilde{sub}_{i+1}(\eta, x, y))(\tilde{v}_{i+1}(x) - \tilde{v}_{i+1}(y)) \\
&\qquad + (\alpha_i \tilde{mult}_{i+1}(\rho, x, y) + \beta_i \tilde{mult}_{i+1}(\eta, x, y))(\tilde{v}_{i+1}(x)\tilde{v}_{i+1}(y)))
\end{aligned} \tag{12}
$$

$\mathcal{P}$ and $\mathcal{V}$ then run the Sumcheck protocol on the above polynomial instead. In the last step of the Sumcheck, $\mathcal{V}$ still receives two claims about $\tilde{v}_{i+1}$. Since Equation (12) can handle two random points ρ and η at once, the exponential invocations of the Sumcheck protocol can be avoided. Thus, starting from the outputs, $\mathcal{P}$ and $\mathcal{V}$ can recursively process the circuit until the input layer.

In terms of protocol complexity, it can be shown that the GKR protocol above can have a $O(d \log |F|)$ proof size. The verifier time is $O(|\mathbf{io}| + d \log |F|)$. The soundness error of the protocol is $O(\frac{d \log |F|}{|\mathbb{F}|})$ [48, 51]. The prover time complexity depends on the implementation. Naively running the Sumcheck protocol on polynomial (12) incurs $O(|F|^2)$ prover time. Later, Cormode et al. observed that these polynomials are sparse, containing only $O(|F|)$ nonzero monomials and improved the prover time to $O(|F| \log |F|)$ [17]. Xie et al. further achieved linear $O(|F|)$ prover time [51].

Remark 34. Over the years, researchers have made substantial improvements to the GKR protocol, some of which can be adopted directly in the construction of our proposed Interstellar protocol framework. For example, the original GKR protocol applies only to layered circuits, and yet Zhang et al. showed that even non-layered circuits can be efficiently handled by a variant of the GKR protocol [52]. Notably, after normalization as discussed in Section 4.1, the resulting circuit C' is not strictly a layered circuit. Hence, Zhang et al.'s techniques can be applied. Moreover, Cormode et al. discussed how to efficiently handle gates with fan-in larger than two [17]. In particular, it is shown that a tree of addition gates at the circuit output can be converted into a high fan-in addition gate with the number of inputs equal to the number of leaves of the addition gate tree. We can run a single Sumcheck protocol to check if the claimed output $v_0 = \sum_{x \in \{0,1\}^{\log N_1}} \tilde{v}_1(x)$, where N_1 is the number of fan-ins of the addition gate. Since $\tilde{v}_1(x)$ is a multilinear function and $\mathcal{P}$ can evaluate it for $\mathcal{V}$, based on our analysis of the Sumcheck protocol in Section A.5, the prover and verifier time complexity for this check are $O(N_1)$ and $O(\log N_1)$. The proof size is $O(\log N_1)$, and the soundness error is $O(\frac{\log N_1}{|\mathbb{F}|})$.

A.7 Succinct Non-interactive Argument Systems

Argument systems assume *computationally bounded* provers [10, 48]. In more detail, an argument system for a circuit F is an interactive proof for the satisfiability of F in which the soundness condition is only required to hold against prover

strategies that run in polynomial time. This notion of soundness is called *computational soundness*. Arguments are allowed to use cryptographic primitives, as we assume that a polynomial time prover is unable to break the primitives. Incorporating cryptography achieves additional useful properties that are otherwise unattainable, such as public verifiablity, succinctness, and non-interactivity.

In particular, *succinctness* means that the argument is short, at least sublinear of the size of the witness. *Non-interactivity* means that the argument is static, consisting of a single message from the prover to the verifier. An interactive proof or argument can be turned into a non-interactive argument by applying the Fiat-Shamir transformation [4, 14, 21, 35, 51]. There are other desirable properties of an argument, for example, *argument of knowledge* which is closely related to the concept of *knowledge soundness* defined in Section A.1. It roughly means that not only that a statement is valid, but also that the prover knows a witness to the veracity of the statement. To be more precise, if the prover can convince the verifier to accept the statement with non-negligible probability, then there exists an extractor algorithm which can efficiently extract the witness from the prover. Argument systems that satisfy all or even a subset of these properties have a myriad of applications in various fields, both in theory and practice.

B More Details of the IVC Scheme

B.1 Construction of F'

We use $\mathsf{U_i} = (\boldsymbol{\beta}_i, \mathbf{X}_i, com_{\mathbf{W}i}, Y_i)$ to represent the correct execution of invocations $1, .., i-1$ of F^*, and $\mathsf{u_i} = (\boldsymbol{\beta}_i, \mathbf{x}_i, com_{\mathbf{w}i}, y_i)$ to represent the correct execution of the i^{th} invocation of F^*. In the following, we provide the formal construction of F'.

Construction 35 *Let* $\mathsf{NIFS} = (\mathsf{G}, \mathsf{K}, \mathsf{P}, \mathsf{V})$ *be the non-interactive folding scheme defined by Construction 14. Consider a computation represented by a circuit* $\mathbf{z}_{i+1} = F(\mathbf{z}_i, \boldsymbol{\omega}_i)$ *that takes non-deterministic input, and a cryptographic hash function* hash*. We define the augmented circuit* F' *as follows:*

$F'(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, \mathbf{z}_0, \mathbf{z}_i), \boldsymbol{\omega}_i, r_i, r_{i+1})$*:*

If $i = 0$*, output* $\mathsf{hash}(\mathsf{vk}, 1, \mathbf{z}_0, F(\mathbf{z}_0, \boldsymbol{\omega}_0), r_{i+1})$*,*

otherwise,

(1) check if $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, \mathbf{z}_0, \mathbf{z}_i, \mathsf{U}_i, r_i)$*, where* $\mathsf{u}_i.\mathsf{x}$ *is the output of* u_i*,*

(2) check if $\mathsf{u}_i.y = 0$*,*

(3) compute $\mathsf{U}_{i+1} = \mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, com_{\mathbf{w}i})$ *where* $com_{\mathbf{w}i}$ *is the* $\mathsf{PolyCom}$ *commitment for the multilinear extension of* $\mathbf{w}_i$*, the witness of* u_i*.*

(4) output $\mathsf{hash}(\mathsf{vk}, i+1, \mathbf{z}_0, F(\mathbf{z}_i, \boldsymbol{\omega}_i), \mathsf{U}_{i+1}, r_{i+1})$*.*

Remark 36. For ease of presentation, we rearrange the inputs to resulting circuit F' takes to a public input vector $\mathbf{x}'$ and a private witness vector $\mathbf{w}'$, i.e., $\mathbf{x}' = (vk, \mathsf{U}_i, \mathsf{u}_i, (i, \mathbf{z}_0, \mathbf{z}_i), \boldsymbol{\omega}_i, r_i, r_{i+1})$ and $\mathbf{w}' = \boldsymbol{\omega}_i$. We also denote the output vector of F' by $\mathbf{y}'$, i.e., $\mathbf{y}' = \mathsf{hash}(\mathsf{vk}, i+1, \mathbf{z}_0, F(\mathbf{z}_i, \boldsymbol{\omega}_i), \mathsf{U}_{i+1}, r_{i+1})$. This way, we can decribe F' more compactly as $\mathbf{y}' = F'(\mathbf{x}', \mathbf{w}')$.

Remark 37. At first glance, the above construction appears to have a circular dependency. In particular, as described in Section 4.1, F^* is constructed by flattening F' and then adding extra circuitry. On the other hand, F' operates on $\mathsf{U_i}$ and $\mathsf{u_i}$, the committed satisfying instances of F^*. However, this actually does not create a circular dependency. This is because both $\mathsf{U_i}$ and $\mathsf{u_i}$ are simply three vectors of field elements plus one commitment (e.g. a group element if Pedersen's commitment is used). In the construction of F', running $\mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, com_{\mathbf{w}i})$ only involves well-defined field and group operations on $\mathsf{U_i} = (\boldsymbol{\beta}_i, \mathbf{X}_i, com_{\mathbf{W}i}, Y_i)$ and $\mathsf{u_i} = (\boldsymbol{\beta}_i, \mathbf{x}_i, com_{\mathbf{w}i}, y_i)$, and does not require the "structure" of F^*. This is similar to the original Nova construction, where F' only operates on the witnesses to the relaxed R1CS instances but not on the R1CS matrices (A, B, C), which represents the structure of F'.

B.2 Alternative Constructions of Coefficient Vector s

Alternative Strategy 1. Instead of using Formula 2, we can also generate $\mathbf{s}$ the following way to reduce the number of layers of the circuits to generate $\mathbf{s}$ to $\log_b n$ (e.g. $b = 10$). In this setup, the verifier randomly samples $\{\beta_0, \beta_1, .., \beta_{\log_b n}\}$ and sends them to the prover. Both the prover and verifier then calculate $\{\beta_{i,j} = \beta_i^j \mid i = 0, 1, .., \log_b n - 1; j = 0, 1, .., b-1\}$ and generate $\mathbf{s}$ using the following formula:

$$\mathbf{s} = \{ \prod_{i=0}^{\log_b n - 1} \beta_{i,j_i} | (j_0, j_1, .., j_{\log_b n - 1}) \in \{0, 1, .., b-1\}^{\log_b n - 1}\} \tag{13}$$

The circuit generating $\mathbf{s}$ defined above is a $(b \log_b n)$-degree circuit which has $\log_b n$ layers and can be implemented using $O(n)$ gates using the "evaluation tree" technique illustrated in the proof for Lemma 6.

Alternative Strategy 2. Alternatively, we can follow the technique in Section 3.5 of Protostar [11] for which the verifier only samples and sends the prover a single random value $\beta \in \mathbb{F}$. Then the prover calculates $2\sqrt{n}$ values from β as follows:

$$[\beta_i \leftarrow \beta^i]_{i=1}^{\sqrt{n}-1},\ [\beta'_j \leftarrow \beta^{j\sqrt{n}}]_{j=1}^{\sqrt{n}-1}$$

The prover sends these values back to the verifying, which can then generate a length-n vector $\mathbf{s}$ like below:

$$\mathbf{s} = \{\beta_i \cdot \beta'_j \mid i = 1, .., \sqrt{n} - 1; j = 1, .., \sqrt{n} - 1\} \tag{14}$$

The verifier also need to add a sub-circuit conduct $2\sqrt{n}$ degree-2 consistent checks $\beta_{i+1} - \beta_i \cdot \beta = 0$ and $\beta'_{i+1} - \beta'_i \cdot \beta'_1$ for $i = 1, .., \sqrt{n} - 2 = 0$ to make sure the prover generate vector $\mathbf{s}$ honestly.

The benefit of this construction is that each element of $\mathbf{s}$ is just a degree-2 polynomial of the inputs, unlike in Formula 2, where each element is a degree-$\log n$ polynomial of the inputs. Thus, it only increases the overall degree of the circuit by 2, and the number of layers by at most 2, i.e., $D^* = \bar{D} + 2$, and $d^* = \min\{\bar{d}, 1\} + 2 = \bar{d} + 2$. One of the downsides is that with this construction, the prover is required to keep track of an error vector representing the results of the $2\sqrt{n}$ consistency checks and needs to commit to this error vector for each folding round. This adds a group MSM operation of size $O(\sqrt{n})$ to the prover overhead.

B.3 SNARK Protocol for Validating an IVC Proof

In this section, we formally define the SNARK construction for a valid IVC proof.

Construction 38 (SNARK for a Valid IVC Proof) *Let* $\mathsf{IVC} = (\mathsf{G}, \mathsf{K}, \mathsf{P}, \mathsf{V})$ *denote the IVC scheme in Construction 15. Let* NIFS *denote the non-interactive folding scheme in Construction 14. Let* hash *denote a randomized cryptographic hash function that provides hiding. Assume a SNARK* SNARK *for circuit satisfiability that has the same public parameter generation algorithm. We construct a SNARK* $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ *for the IVC scheme as follows.*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$*: Output* $\mathsf{pp} \leftarrow \mathsf{SNARK.G}(1^\lambda)$.
- $\mathcal{K}(\mathsf{pp}, F) \to (\mathsf{pk}, \mathsf{vk})$*:*
  - *Compute* $(\mathsf{pk}_{\mathsf{IVC}}, \mathsf{vk}_{\mathsf{IVC}}) \leftarrow \mathsf{IVC.K}(\mathsf{pp}, F)$.
  - *generate circuit* $F^* \leftarrow \mathsf{NORMALIZE}(\mathsf{pp}, F)$ *as described in Section 4.1.*
  - *Compute* $(\mathsf{pk}_{\mathsf{SNARK}}, \mathsf{vk}_{\mathsf{SNARK}}) \leftarrow \mathsf{SNARK.K}(\mathsf{pp}, F^*)$.
  - *Output* $\mathsf{pk} \leftarrow (\mathsf{pk}_{\mathsf{IVC}}, \mathsf{pk}_{\mathsf{SNARK}})$ *and* $\mathsf{vk} \leftarrow (\mathsf{vk}_{\mathsf{IVC}}, \mathsf{vk}_{\mathsf{SNARK}})$.
- $\mathcal{P}(\mathsf{pk}, (i, \mathbf{z}_0, \mathbf{z}_n), \Pi_i) \to \pi$*:*
  - *If* $i = 0$*, output* $\perp$*;*
  - *Otherwise,*
    - *parse* Π_i *as* $((\mathsf{U}_i, \mathbf{W}_i), (\mathsf{u}_i, \mathbf{w}_i), r_i)$*;*
    - *compute* $(\mathsf{U}', \mathbf{W}') \leftarrow \mathsf{NIFS.P}(\mathsf{pk}_{\mathsf{IVC}}, ((\mathsf{U}_i, \mathbf{W}_i), (\mathsf{u}_i, \mathbf{w}_i)))$*;*
    - *compute* $\pi_{\mathsf{U}' \leftarrow \mathsf{zkSNAKR.P}(\mathsf{pk}_{\mathsf{SNARK}}, \mathsf{U}', \mathbf{W}')}$*;*
    - *output* $(\mathsf{U}_i, \mathsf{u}_i, r_i, \pi_{\mathsf{U}'})$.
- $\mathcal{V}(\mathsf{vk}, (i, \mathbf{z}_0, \mathbf{z}_n), \pi) \to \{0, 1\}$*:*
  - *If* $i = 0$*, check that* $\mathbf{z}_0 = \mathbf{z}_i$*;*
  - *otherwise,*
    - *parse* π *as* $(\mathsf{U}_i, \mathsf{u}_i, r_i, \pi_{\mathsf{U}'})$,
    - *check that* $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}_{\mathsf{IVC}}, i, \mathbf{z}_0, \mathbf{z}_i, \mathsf{U}_i, r_i)$,
    - *randomly samples* $\boldsymbol{\beta}_i \in \mathbb{F}^{\log n}$ *and set* $\mathsf{u}_i.\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_i$,
    - *compute* $\mathsf{U}' \leftarrow \mathsf{NIFS.V}(\mathsf{vk}_{\mathsf{IVC}}, \mathsf{U_i}, \mathsf{u_i})$*, and*
    - *check that* $\mathsf{SNARK.V}(\mathsf{vk}_{\mathsf{SNARK}}, \mathsf{U}', \pi_{\mathsf{U}'}) = 1$.

B.4 Multi-Fold Protocol Construcion

Here, we present the formal construction of the protocol for folding multiple instances at once:

Construction 39 (Interactive Multi-Fold Scheme (IMFS)**)** *Consider a finite field* $\mathbb{F}$ *and a succinct, hiding, and homomorphic polynomial commitment scheme for multilinear polynomials* PolyCom *over* $\mathbb{F}$*. We define the generator and encoder as follows:*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$*: output size bound for* $|F|$*, and commitment parameters* $\mathsf{pp}_{\mathbf{w}}$*.*
- $\mathcal{K}(\mathsf{pp}, F') \to (\mathsf{pk}, \mathsf{vk})$*: output* $\mathsf{pk} \leftarrow (\mathsf{pp}, F^*)$ *and* $\mathsf{vk} \leftarrow \perp$*.*

The verifier $\mathcal{V}$ *takes multiple committed instances* $\mathsf{u}_0 = (\boldsymbol{\beta}_0, \mathbf{x}_0, com_{\mathbf{w}0}, y_0)$ *and* $\{\mathsf{u}_i = (\perp, \mathbf{x}_i, com_{\mathbf{w}i}, 0) | i = 1, .., k-1\}$*. The prover* $\mathcal{P}$*, in addition to the two instances, takes witnesses to these instances* $\{(\mathbf{w}_i, r_{\mathbf{w}i}) | i = 0, 1, .., k-1\}$*. The prover and verifier proceed as follows:*

- $\mathcal{V}$*: Sample independent random field elements* $\boldsymbol{\beta}_1 = (\beta_0, .., \beta_k, .., \beta_{\log n - 1}) \in \mathbb{F}^{\log n}$*, and send them to* $\mathcal{P}$*.*
- $\mathcal{P}$*: Set* $\mathsf{u}_1.\boldsymbol{\beta} = \mathsf{u}_2.\boldsymbol{\beta} = .. = \mathsf{u}_{k-1}.\boldsymbol{\beta} = \boldsymbol{\beta}$*. Then, compute the point evaluation form of the polynomial* $y(t)$*, i.e., the evaluations of* $y(t) = F^*(\boldsymbol{\beta}(t), \mathbf{x}(t), \mathbf{w}(t))$ *as a function of* t *at* $t = k, k+1, .., kD^*$*. Here* $\mathbf{w}(t) = \sum_{j=0}^{k-1} \delta(t, j) \cdot \mathbf{w}_j$ *where* $\{\delta(t, j) | j = 0, 1, .., k\}$ *are the Lagrange basis polynomials.* $\boldsymbol{\beta}(t)$ *and* $\mathbf{x}(t)$ *are defined similarly. Send* $\{y(k), y(k+1), .., y(kD^*)\}$ *to* $\mathcal{V}$*.*
- $\mathcal{V}$*: Randomly sample* $\gamma \in \mathbb{F}$*, and send it to* $\mathcal{P}$*.*
- $\mathcal{P}$ *and* $\mathcal{V}$*: Output the folded instance* $(\boldsymbol{\beta}_\gamma, \mathbf{x}_\gamma, com_{\mathbf{w}\gamma}, y_\gamma)$ *as follows, where* $y(\gamma)$ *can be calculated efficiently by interpolating a degree-*(kD^*) *polynomial through points* $\{(0, y_0), (1, 0), .., (k-1, 0), (k, y(k)), .., (kD^*, y(kD^*))\}$ *via the barycentric evaluation method.*

$$\begin{aligned}
\boldsymbol{\beta}_\gamma &\leftarrow \sum_{j=0}^{k-1} \delta(\gamma, j) \cdot \boldsymbol{\beta}_j \\
\mathbf{x}_\gamma &\leftarrow \sum_{j=0}^{k-1} \delta(\gamma, j) \cdot \mathbf{x}_j \\
com_{\mathbf{w}\gamma} &\leftarrow \sum_{j=0}^{k-1} \delta(\gamma, j) \cdot com_{\mathbf{w}j} \\
y_\gamma &\leftarrow y(\gamma)
\end{aligned}$$

- P*: Output the folded witness* $(\mathbf{w}_\gamma, r_{\mathbf{w}})$*, where*

$$\mathbf{w}_\gamma \leftarrow \sum_{j=0}^{k-1} \delta(\gamma, j) \cdot \mathbf{w}_j$$
$$r_{\mathbf{w}\gamma} \leftarrow \sum_{j=0}^{k-1} \delta(\gamma, j) \cdot r_{\mathbf{w}j}$$

The above construction can be turned non-interactive using the Fiat-Shamir transformation:

Construction 40 (Non-Interactive Multi-Fold Scheme (NIMFS)) *Under the random oracle model, non-interactivity can be achieved using the strong Fiat-Shamir transformation. Let ρ denote a random oracle sampled during parameter generation and provided to all parties. Let $(\mathsf{G}, \mathsf{K}, \mathsf{P}, \mathsf{V})$ represented the interactive folding scheme defined above in Construction 39. A non-interactive folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ can be constructed as follows:*

- $\mathcal{G}(1^\lambda)$*: output* $\mathsf{pp} \leftarrow \mathsf{G}(1^\lambda)$.
- $\mathcal{K}(\mathsf{pp}, F)$*:* $\mathsf{vk} \leftarrow \rho(\mathsf{pp}, F)$ *and* $\mathsf{pk} \leftarrow (\mathsf{pp}, F^*, \mathsf{vk})$*; output* $(\mathsf{vk}, \mathsf{pk})$.
- $\mathcal{P}(\mathsf{pk}, \{(\mathsf{u}_i, \mathbf{w}_i) \mid i = 0, 1, .., k-1\}$*: Generate* $\log n$ *random values* $\boldsymbol{\beta} = \{\beta_i \in \mathbb{F} | i = 0, 1, .., \log n - 1\}$ *using the hash function ρ whose input is the concatenation of all messages up to that point. Set* $\mathsf{u}_1.\boldsymbol{\beta} = \mathsf{u}_2.\boldsymbol{\beta} = .. = \mathsf{u}_{k-1}.\boldsymbol{\beta} = \boldsymbol{\beta}$*. Then, compute the point evaluation form of the polynomial $y(t)$, i.e., the evaluations of $y(t) = F^*(\boldsymbol{\beta}(t), \mathbf{x}(t), \mathbf{w}(t))$ as a function of t at $t = k, k+1, .., kD^*$. Here $\mathbf{w}(t) = \sum_{j=0}^{k-1} \delta(t, j) \cdot \mathbf{w}_j$ where $\{\delta(t, j) | j = 0, 1, .., k\}$ are the Lagrange basis polynomials. $\boldsymbol{\beta}(t)$ and $\mathbf{w}(t)$ are defined similarly. Forward $\{y(k), y(k+1), .., y(kD^*)\}$ to* $\mathsf{IFS.P}$ *and* $\mathsf{IFS.V}$*; Generate randomness γ using the hash function ρ, and calculate $y_\gamma = y(\gamma)$ via the barycentric interpolation method and then output the resulting folded instance and witness.*
- $\mathcal{V}(\mathsf{vk}, \{\mathsf{u}_i \mid i = 0, 1, .., k-1\})$*: runs* $\mathsf{IFS.V}$ *with randomness $\boldsymbol{\beta} \in \mathbb{F}^{\log n}$ generated the same way as $\mathcal{P}$; Receive $\{y(k), y(k+1), .., y(kD^*)\}$ and generate γ the same way as $\mathcal{P}$. Calculate $y_\gamma = y(\gamma)$ using the barycentric interpolation method and output the resulting folded instance.*

B.5 Folding Scheme for Non-Uniform IVC

Non-uniform IVC aims at efficiently producing succinct proofs of correct execution of programs in a VM with a particular instruction set [11, 30]. The goal is to attain the "a la carte" cost profile, i.e., the cost for proving a single step of a program is proportional only to the size of the sub-circuit representing the instruction invoked by the program step, instead of the entire circuit representing all supported instructions. Below we illustrate how we can achieve this.

First, similar to [11, 30], we assume that the VM has I sub-circuits. Naturally, we can extend the definition of the committed circuit instance and the witness in Definition 11 to $(\mathbf{B}, \mathbf{X}, com_{\mathbf{W}}, \mathbf{Y})$ and $(\mathbf{W}, r_{\mathbf{W}})$, where the witness

$\mathbf{W} = (\mathbf{w}^{(0)}, .., \mathbf{w}^{(I-1)})$. Randomness $\mathbf{B}$, public input $\mathbf{X}$, commitments $com_{\mathbf{W}}$ and commitment randomness $r_{\mathbf{W}}$ are also expanded to I-element vectors, defined similarly to $\mathbf{W}$. For the output, we also expand y as defined in Formula 3 to a vector $\mathbf{Y} = \{y^{(i)} \mid i = 0, 1, .., I-1\}$, where:

$$y^{(i)} = \mathbf{s}^{(i)} \cdot \mathbf{y}^{(i)} \text{ for } i = 0, 1, .., I-1 \tag{15}$$

and $\mathbf{s}^{(i)}$ are derived from the randomness $\boldsymbol{\beta}^{(i)}$ similar to Formula 2.

In the context of IVC, let us denote the currently accumulated instance-witness pair as $(\mathbf{B}_{acc}, \mathbf{X}_{acc}, com_{\mathbf{W}acc}, \mathbf{Y}_{acc})$ and $(\mathbf{W}_{acc}, r_{\mathbf{W}acc})$. Also, note that during the program execution, at each step, only one sub-circuit is active. Hence, effectively, the satisfying instance-witness pair for each incremental computation can still be represented using the original format $(\boldsymbol{\beta}_{step}, \mathbf{x}_{step}, com_{\mathbf{w}step}, y_{step})$ and $(\mathbf{w}_{step}, r_{\mathbf{w}step})$ compactly. In order to fold it into the accumulated instance, we need to expand it first, so that the two instances to be folded together have the same format. To account for the fact that only one sub-circuit is active for each folding step, we assume there is a program counter $pc \in \{0, 1, .., I-1\}$ which equals the index of the currently active sub-circuit. We include in the input $\mathbf{X}$ a "one-hot" program counter vector $\mathbf{b} = (b^{(0)}, b^{(1)}, .., b^{(I-1)})$ where only the pc^{th} bit is 1 and all the others are 0. We then define the expanded witness for the step computation as:

$$\mathbf{W}_{step} = ((1 - b^{(i)}) \cdot \mathbf{W}_{acc}^{(i)} + b^{(i)} \cdot \mathbf{w}_{step} \mid i = 0, 1, .., I-1) \tag{16}$$

where $\mathbf{W}_{acc}^{(i)}$ is the witness vector to the i^{th} sub-circuit in the accumulated instance. In other words, $\mathbf{W}_{step}^{(i)} = \mathbf{w}_{step}$ if $i = pc$ and $\mathbf{W}_{step}^{(i)} = \mathbf{W}_{acc}^{(i)}$ otherwise. The intuition is that only the active sub-circuit needs to be folded, and the remaining should stay the same. Let us see how we can achieve this with the following folding procedure:

$$\begin{aligned}
\mathbf{B}_{acc} &\leftarrow (1-\gamma) \cdot \mathbf{B}_{acc} + \gamma \cdot \mathbf{B}_{step} \\
\mathbf{X}_{acc} &\leftarrow (1-\gamma) \cdot \mathbf{X}_{acc} + \gamma \cdot \mathbf{X}_{step} \\
com_{\mathbf{W}acc} &\leftarrow (1-\gamma) \cdot com_{\mathbf{W}acc} + \gamma \cdot com_{\mathbf{W}step}
\end{aligned}$$

Similarly, the prover updates the folded witness $(\mathbf{W}_{acc}, r_{\mathbf{W}acc})$ as

$$\begin{aligned}
\mathbf{W}_{acc} &\leftarrow (1-\gamma) \cdot \mathbf{W}_{acc} + \gamma \cdot \mathbf{W}_{step} \\
r_{\mathbf{W}acc} &\leftarrow (1-\gamma) \cdot r_{\mathbf{W}acc} + \gamma \cdot r_{\mathbf{W}step}
\end{aligned}$$

Let us analyze $\mathbf{W}_{acc}$ before and after folding to see the outcome of the above process. For $i \neq pc$, we have that $\mathbf{W}_{acc}^{(i)} \leftarrow (1-\gamma) \cdot \mathbf{W}_{acc}^{(i)} + \gamma \cdot \mathbf{W}_{step}^{(i)} = (1-\gamma) \cdot \mathbf{W}_{acc}^{(i)} + \gamma \cdot \mathbf{W}_{acc}^{(i)} = \mathbf{W}_{acc}^{(i)}$. In other words, the vector element remains unchanged. On the other hand, we have $\mathbf{W}_{acc}^{(pc)} \leftarrow (1-\gamma) \cdot \mathbf{W}_{acc}^{(pc)} + \gamma \cdot \mathbf{W}_{step}^{(pc)} = (1-\gamma) \cdot \mathbf{W}_{acc}^{(i)} + \gamma \cdot \mathbf{w}_{step}$, which is a random combination akin to the corresponding

folding formula in Construction 12. The same holds for $\boldsymbol{B}_{acc}$, $\mathbf{X}_{acc}$, $com_{\mathbf{W}acc}$, and $r_{\mathbf{W}acc}$.

Next, we turn our attention to the output vector $\mathbf{Y}$. For $i \neq pc$, per our analysis above, the public inputs and witness of the sub-circuit remain unchanged, therefore, $y^{(i)}$ stays the same. For $i = pc$, the prover and the verifier can simply compute the values of $y^{(i)}(\gamma)$ using the barycentric evaluation method similar to Construction 12. The formal construction of the folding scheme for non-uniform IVC is presented in Construction 41 in below. It is worth pointing out that the proof compression technique from Section 5.2 is compatible with Construction 41. This is because the verifier can generate the multilinear extension of $\mathbf{B}_{acc}$, $\mathbf{X}_{acc}$, and $\mathbf{Y}_{acc}$ on her own. And with $com_{\mathbf{W}}$, the verifier can validate the evaluation of the multilinear extension of $\mathbf{W}_{acc}$ at any random point, thanks to the additive homomorphic property of the PolyCom commitment.

Construction 41 *Consider a finite field $\mathbb{F}$ and a succinct, hiding, and homomorphic polynomial commitment scheme for multilinear polynomials* PolyCom *over $\mathbb{F}$. Assume circuit F hs $I+1$ sub-circuits $F^{(0)}$, $F^{(1)}$, .., $F^{(I)}$ where for each compuatation step, only one of the first I sub-circuits are activated. On the other hand, the last sub-circuit $F^{(I)}$ handles common computations and is always active for all steps. We define the generator and encoder as follows:*

- $\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}$*: output size bound for $|F|$, and commitment parameters* $\mathsf{pp}_{\mathbf{w}}$.
- $\mathcal{K}(\mathsf{pp}, F') \rightarrow (\mathsf{pk}, \mathsf{vk})$*: output* $\mathsf{pk} \leftarrow (\mathsf{pp}, F^*)$ *and* $\mathsf{vk} \leftarrow \perp$.

The verifier $\mathcal{V}$ takes an instance $\mathsf{U}_{acc} = (\mathbf{B}_{acc}, \mathbf{X}_{acc}, com_{\mathbf{W}acc}, \mathbf{Y}_{acc})$ which accumulates all previous computation steps, and $\mathsf{u}_{step} = (\perp, \mathbf{x}_{step}, com_{\mathbf{w}step}, 0)$ for the sub-circuit active for the current step. In addition, the verifier is given $\mathsf{u}_{com} = (\perp, \mathbf{x}_{com}, com_{\mathbf{w}com}, 0)$ for the common sub-circuit $F^{(I)}$ which is always active. The prover $\mathcal{P}$, in addition to the three instances, take witnesses to all three instances $(\mathbf{W}_{acc}, r_{\mathbf{W}acc})$, $(\mathbf{w}_{step}, r_{\mathbf{w}step})$, and $(\mathbf{w}_{com}, r_{\mathbf{w}com})$. The prover and verifier proceed as follows:

- $\mathcal{V}$*: Sample independent random vectors $\boldsymbol{\beta}_{step} = (\beta_0, .., \beta_k, .., \beta_{\log n - 1}) \in \mathbb{F}^{\log n}$ and $\boldsymbol{\beta}_{com} = (\beta'_0, .., \beta'_k, .., \beta'_{\log n - 1}) \in \mathbb{F}^{\log n}$ and send both $\boldsymbol{\beta}_{step}$ and $\boldsymbol{\beta}_{com}$ to $\mathcal{P}$.*
- $\mathcal{P}$ *and* $\mathcal{V}$*: Create the inputs to step instance $\mathbf{B}_{step}$ and $\mathbf{X}_{step}$ and witness commitment $com_{\mathbf{W}step}$ as follows:*

$$
\begin{aligned}
\mathbf{B}_{step} &= (\boldsymbol{\beta}^{(0)}, .., \boldsymbol{\beta}^{(I-1)}, \boldsymbol{\beta}_{com}) \\
&\quad \text{where } \boldsymbol{\beta}^{(i)} = (1 - b^{(i)}) \cdot \mathbf{B}^{(i)}_{acc} + b^{(i)} \cdot \boldsymbol{\beta}_{step} \ \text{ for } 0 \leq i \leq I-1 \\
\mathbf{X}_{step} &= (\mathbf{x}^{(0)}, .., \mathbf{x}^{(I-1)}, \mathbf{x}_{com}) \\
&\quad \text{where } \mathbf{x}^{(i)} = (1 - b^{(i)}) \cdot \mathbf{X}^{(i)}_{acc} + b^{(i)} \cdot \mathbf{x}_{step} \ \text{ for } 0 \leq i \leq I-1 \\
com_{\mathbf{W}step} &= (com^{(0)}_{\mathbf{W}}, .., com^{(I-1)}_{\mathbf{W}}, com_{\mathbf{W}com}) \\
&\quad \text{where } com^{(i)}_{\mathbf{W}} = (1 - b^{(i)}) \cdot com^{(i)}_{\mathbf{W}acc} + b^{(i)} \cdot com_{\mathbf{w}step} \ \text{ for } 0 \leq i \leq I-1
\end{aligned}
$$

- $\mathcal{P}$: *Create the witness to the step instance* $(\mathbf{W}_{step}, r_{\mathbf{W}step})$ *as follows:*

$$\begin{aligned}\mathbf{W}_{step} &= (\mathbf{w}^{(0)}, .., \mathbf{w}^{(I-1)}, \mathbf{w}_{com}) \\ &\quad \text{where } \mathbf{w}^{(i)} = (1-b^{(i)}) \cdot \mathbf{W}_{acc}^{(i)} + b^{(i)} \cdot \mathbf{w}_{step} \text{ for } 0 \leq i \leq I-1 \\ r_{\mathbf{W}step} &= (r_{\mathbf{W}}^{(0)}, .., r_{\mathbf{W}}^{(I-1)}, r_{\mathbf{w}com}) \\ &\quad \text{where } r_{\mathbf{W}}^{(i)} = (1-b^{(i)}) \cdot r_{\mathbf{W}acc}^{(i)} + b^{(i)} \cdot r_{\mathbf{W}step} \text{ for } 0 \leq i \leq I-1\end{aligned}$$

- $\mathcal{P}$: *Compute the point evaluation form of polynomial* $y^{(pc)}(t)$*, i.e. the evaluations of* $y^{(pc)}(t) = F^{(pc)*}(\boldsymbol{\beta}^{(pc)}(t), \mathbf{x}^{(pc)}(t), \mathbf{w}^{(pc)}(t))$ *as a function of* t *at* $t = 2, 3, .., D^{(pc)*}$*. Here* $\mathbf{w}^{(pc)}(t) = (1-t) \cdot \mathbf{w}_{acc}^{(pc)} + t \cdot \mathbf{w}_{step}^{(pc)}$*, and* $\boldsymbol{\beta}^{(pc)}(t)$ *and* $\mathbf{x}^{(pc)}(t)$ *are defined similarly. Send* $\{y^{(pc)}(2), y^{(pc)}(3), .., y^{(pc)}, (D^{(pc)*})\}$*, i.e. the point evaluation form of* $y^{(pc)}(t)$*, to* $\mathcal{V}$*.*
- $\mathcal{P}$: *Similarly, compute the point evaluation form of polynomial* $y^{(I)}(t)$ *at* $t = 2, 3, .., D^{(I)*}$*. Send* $\{y^{(I)}(2), y^{(I)}(3), .., y^{(I)}, (D^{(I)*})\}$*, i.e. the point evaluation form of* $y^{(I)}(t)$*, to* $\mathcal{V}$*.*
- $\mathcal{V}$: *Randomly sample* $\gamma \in \mathbb{F}$*, and send it to* $\mathcal{P}$*.*
- $\mathcal{P}$ *and* $\mathcal{V}$: *Update the accumulated instance as follows.*

$$\begin{aligned}\mathbf{B}_{acc} &\leftarrow (1-\gamma) \cdot \mathbf{B}_{acc} + \gamma \cdot \mathbf{B}_{step} \\ \mathbf{X}_{acc} &\leftarrow (1-\gamma) \cdot \mathbf{X}_{acc} + \gamma \cdot \mathbf{X}_{step} \\ com_{\mathbf{W}acc} &\leftarrow (1-\gamma) \cdot com_{\mathbf{W}acc} + \gamma \cdot com_{\mathbf{W}step} \\ \mathbf{Y}_{acc} &\leftarrow \mathsf{Update}(\mathbf{Y}_{acc})\end{aligned}$$

  Here is how $\mathsf{Update}(\mathbf{Y}_{acc})$ *works. First,* $y_{acc}^{(i)} \leftarrow (1-b^{(i)}) \cdot y_{acc}^{(i)} + b^{(i)} \cdot y^{(pc)}(\gamma)$ *for* $i = 0, 1, .., I-1$*. In this formula,* $y^{(pc)}(\gamma)$ *can be derived from the following points* $\{(0, y_{acc}^{(pc)}), (1, 0), (2, y^{(pc)}(2)), .., (D^{(pc)*}, y^{(pc)}(D^{(pc)*}))\}$ *efficiently via the barycentric evaluation method. Likewise,* $y_{acc}^{(I)} \leftarrow y^{(I)}(\gamma)$ *can be calculated efficiently from* $\{(0, y_{acc}^{(I)}), (1, 0), (2, y^{(I)}(2)), .., (D^{(I)*}, y^{(I)}(D^{(I)*}))\}$*. Note that since* $b^{(i)} = 0$ *for all* $i \neq pc$*, only the* pc^{th} *and the* I^{th} *element of* $\mathbf{Y}_{acc}$ *need to be updated. Other elements of* $\mathbf{Y}_{acc}$ *remain unchanged.*

- P: *Output the accumulated witness* $(\mathbf{W}_{acc}, r_{\mathbf{W}acc})$*, where*

$$\begin{aligned}\mathbf{W}_{acc} &\leftarrow (1-\gamma) \cdot \mathbf{W}_{acc} + \gamma \cdot \mathbf{W}_{step} \\ r_{\mathbf{W}acc} &\leftarrow (1-\gamma) \cdot r_{\mathbf{W}acc} + \gamma \cdot r_{\mathbf{W}step}\end{aligned}$$

Now let us analyze the efficiency of the above construction. At first glance, its time complexity increases with I, since $\mathbf{B}$, $\mathbf{X}$, $com_{\mathbf{W}}$, $r_{\mathbf{W}}$, and $\mathbf{Y}$ are all vectors of dimensions I. However, as demonstrated above, for each of these vectors, only

one element gets updated in each folding round. In practice, F might have a subcircuit which is always active for each computation step, e.g. for aggregating the results from each of the I sub-circuits and outputting to the user. We can simply add an extra element to $\mathbf{B}$, $\mathbf{X}$, $com_{\mathbf{W}}$, $\mathbf{Y}$, $\mathbf{W}$, and $r_{\mathbf{W}}$ to accommodate this always-on sub-circuit. Moreover, in the above we assume that $\mathbf{b}$ is indeed a one-hot vector, but this needs validation. This condition can be checked by verifying whether $b^{(i)} \cdot (1 - b^{(i)}) = 0$ holds for all $i = 0, 1, .., I-1$ and $\sum_{i=0}^{I-1} b^{(i)} = 1$. These checks and the $\mathbf{b}$ input can be absorbed into the always-on sub-circuit. In summary, the runtime of a folding step only depends on the size of the activated sub-circuit, plus this extra circuit for common computations. Thus, we achieve the "a la carte" cost profile. We have the following theorem regarding the properties of Construction 41 , whose proof can be found in Appendix C.4.

Theorem 42. *Construction 41 achieves completeness and knowledge soundness. In terms of efficiency, the cost of proving a step of a program is proportional only to the size of the circuit invoked by the program step.*

C Proofs for the Folding and IVC Scheme

C.1 Proofs for the Folding Scheme

Below we prove Lemma 5, which characterizes the perfect completeness and knowledge soundness of the circuit normalization process from F' to F^*:

Proof. The proof of the first claim, i.e. the perfect completeness property, is straightforward and is omitted here. To prove the latter claim, i.e. the soundness property, let us first construct the following function of variables $\{\beta_0, .., \beta_k, .., \beta_{\log n-1}\} \in \mathbb{F}^{\log n}$:

$$f(\beta_0, .., \beta_{\log n-1}) = \sum_{i=1}^{n} (a_i \cdot \prod_{k=0}^{\log n-1} (i_k \cdot \beta_k + (1 - i_k) \cdot (1 - \beta_k))) = \sum_{i=1}^{n} a_i \cdot \tilde{\mathsf{eq}}(\boldsymbol{\beta}, \mathbf{i}) \tag{17}$$

where $\{a_1, .., a_i, .., a_n\} \in \mathbb{F}^n$ are constant field elements. Obviously, the function $f(\beta_0, .., \beta_k, .., \beta_{\log n-1})$ is a multilinear function of $\{\beta_0, .., \beta_k, .., \beta_{\log n-1}\} \in \mathbb{F}^{\log n}$. Based on the Schwartz-Zippel lemma for multilinear functions [44, 56], since $\{\tilde{\mathsf{eq}}(t, \mathbf{i}) \mid \mathbf{i} = 0, 1, .., n-1\}$ are the multilinear Lagrange basis, for independently chosen uniformly random values $\{\beta_0, .., \beta_k, .., \beta_{\log n-1}\}$, if $\{a_1, .., a_i, .., a_n\}$ has a non-zero element, the probability that $f(\beta_0, .., \beta_k, .., \beta_{\log n-1}) = 0$ is at most $\frac{\log n}{|\mathbb{F}|}$. Replacing $\{a_1, .., a_i, .., a_n\}$ with $\mathbf{y}$, the outputs of circuit $\bar{F}$, we can arrive at the conclusion that if $F^*(\boldsymbol{\beta}, \mathbf{x}, \mathbf{w}) = \mathbf{s} \cdot \mathbf{y} = 0$, the probability that any of the outputs of circuit $\bar{F}$ is non-zero is at most $\frac{\log n}{|\mathbb{F}|}$. Also, given the equivalence of circuit F' and $\bar{F}$ shown by Lemma 4 and that $n = |\mathbf{s}| \leq |F'|$, it follows that $F'(\mathbf{x}', \mathbf{w}') = \mathbf{y}'$ holds with probability at least $1 - \frac{\log n}{|\mathbb{F}|} \geq 1 - \frac{\log |F'|}{|\mathbb{F}|}$.

In the following, we prove Lemma 6, which states that the proofer can generate random vector $\mathbf{s}$ in $O_\lambda(n)$ time and $O_\lambda(n)$ memory.

Proof. The process for generating $\mathbf{s}$ is very similar to the proof of Lemma 3.8 in [48], which calculates n multilinear Lagrange basis polynomials for $\log n$ variables using an "evaluation tree" with $O(n)$ field multiplications/additions and $O(n)$ memory space. There are $\log n$ stages. In Stage 1, the prover computes and stores two values β_0 and $(1-\beta_0)$. In stage 2, the prover multiplies these two values with β_1 and $(1-\beta_1)$, respectively, and obtains and stores four values $\beta_0\beta_1$, $\beta_0(1-\beta_1)$, $(1-\beta_0)\beta_1$, and $(1-\beta_0)(1-\beta_1)$. In general, in Stage i, the prover computes and stores all such 2^i values from the $2^{(i-1)}$ values obtained in the previous stage using $O(2^i)$ field multiplications/additions until $i = \log n$. Hence, to generate vector $\mathbf{s} = (s_1, s_2, .., s_i, .., s_n)$ where s_i is defined by Formula (2), the algorithm takes $O_\lambda((2^1 + 2^2 + .. + 2^i + .. + 2^{\log n}) = O_\lambda(n)$ times and memory space.

Below we prove Lemma 7, which bounds the size and degree of the normalized circuit F^*.

Proof. Compared to F', the flattened circuit $\bar{F}$ has one more layer of gates, consisting of $n \leq |F'|$ subtraction gates. F^* adds n multiplication gates and an addition gate with n inputs. We also need a circuit to generate $\mathbf{s}$, which can be implemented using $O(n)$ gates with $O(n)$ gates according to the proof of Lemma 6. Moreover, the verifier needs to generate $\log n$ hashes to create $\boldsymbol{\beta} = (\beta_0, .., \beta_k, .., \beta_{\log n - 1}) \in \mathbb{F}^{\log n}$. This can be achieved with $O(\log n)$ gates. Finally, compared to F, F' only adds two hashes and the circuits to fold the inputs and witnesses, each has a size at most $O(|F|)$. Thus, $O(|F^*|) = O(\bar{F}) = O(|F'|) = O(|F|)$. Regarding the degrees, the degree of the circuit that generates $\mathbf{s}$ has degree $\log n$ since each s_i is a product of $\log n$ inputs. The inner product between $\mathbf{s}$ and the outputs of $\bar{F}$ increases its degree by $\log n$. Hence, we have $D^* = \bar{D} + \log n$. As for the depth of the circuit, the hash generating circuit can be flattened using the procedure in Section 4.1 if needed. Hence, the hash generation circuit does not increase the circuit depths and degree. On the other hand, the depth of the circuit generating $\mathbf{s}$ is $\log n$. Flattening this circuit can lead to extra complications as we will need to introduce more randomness to prove the correctness of its execution. If we do not flatten it, it has $\log n$ layers. Counting the multiplication and addition layer mentioned earlier, we have $d^* = min\{\bar{d}, \log n\} + 2$.

Now we prove Theorem 13, which states the completeness and knowledge soundness of the proposed folding scheme.

Proof. Let us first prove the perfect completeness property of the folding scheme, and then the knowledge soundness and public coin property.

Perfect Completeness. Given an adversarially chosen circuit F, according to Lemma 5, we can normalize F into an equivalent circuit F^*, that is, for any

satisfying witness of F, we can construct a corresponding satisfying witness for F^*. Now, consider two committed instances $\mathsf{u}_0 = (\boldsymbol{\beta}_0, \mathbf{x}_0, com_{\mathbf{w}0}, y_0)$ and $\mathsf{u}_1 = (\perp, \mathbf{x}_1, com_{\mathbf{w}1}, y_1)$ for F^*, where $\perp$ means that the $\boldsymbol{\beta}$ vector for u_1 is not set when the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ enter the protocol. Suppose that $\mathcal{P}$, in addition to these two instances, holds the satisfying witnesses to both instances $(\mathbf{w}_0, r_{\mathbf{w}0})$ and $(\mathbf{w}_1, r_{\mathbf{w}1})$.

Now suppose that $\mathcal{P}$ and $\mathcal{V}$ compute a folded instance $\mathsf{u}_\gamma = (\boldsymbol{\beta}_\gamma, \mathbf{x}_\gamma, com_{\mathbf{w}\gamma}, y_\gamma)$ and suppose that $\mathcal{P}$ computes a folded witness $(\mathbf{w}_\gamma, r_{\mathbf{w}\gamma})$. To prove completeness, we must show that $(\mathbf{w}_\gamma, r_{\mathbf{w}\gamma})$ is a satisfying witness of u_γ, which means the following must hold:

$$y_\gamma = F^*(\boldsymbol{\beta}_\gamma, \mathbf{x}_\gamma, \mathbf{w}_\gamma) \tag{18}$$

$$com_{\mathbf{w}\gamma} = \mathsf{PolyCom}(\mathsf{pp}, \mathbf{w}, r_{\mathbf{w}}) \tag{19}$$

For Equation 18, the left-hand side y_γ was the evaluation of $y(t)$ at $\gamma \in \mathbb{F}$, and the right-hand side is the evaluation of F^* at $(\boldsymbol{\beta}_\gamma, \mathbf{x}_\gamma, \mathbf{w}_\gamma)$. Below we show they are the equal. By construction, the prover first obtains the point evaluation form of $y(t)$, i.e. the evaluations of $y(t) = F^*(\boldsymbol{\beta}(t), \mathbf{x}(t), \mathbf{w}(t))$ as a function of t at $t = 0, 1, .., D$. Here $\mathbf{w}(t) = (1-t) \cdot \mathbf{w}_0 + t \cdot \mathbf{w}_1$ and $\mathbf{x}(t) = (1-t) \cdot \mathbf{x}_0 + t \cdot \mathbf{x}_1$. And $\boldsymbol{\beta}(t)$ is defined similarly, that is, $\boldsymbol{\beta}(t) = (1-t) \cdot \boldsymbol{\beta}_0 + t \cdot \boldsymbol{\beta}_1$, but note that $\boldsymbol{\beta}_1$ is randomly sampled by $\mathcal{V}$ during the protocol. Based on this, the verifier can obtain $y(\gamma)$ for the randomly sampled γ using the barycentric evaluation method. On the other hand, the right-hand side essentially first obtains $\boldsymbol{\beta}_\gamma = \boldsymbol{\beta}(t = \gamma)$, $\mathbf{x}_\gamma = \mathbf{x}(t = \gamma)$, and $\mathbf{w}_\gamma = \mathbf{w}(t = \gamma)$, and then evaluates F^* at $(\boldsymbol{\beta}_\gamma, \mathbf{x}_\gamma, \mathbf{w}_\gamma)$. The left-hand side and the right-hand side should reach the same result as $y_\gamma = y(\gamma) = F^*(\boldsymbol{\beta}(\gamma), \mathbf{x}(\gamma), \mathbf{w}(\gamma)) = F^*(\boldsymbol{\beta}_\gamma, \mathbf{x}_\gamma, \mathbf{w}_\gamma)$. This proves that Equation 18 holds.

Equation 19 obviously holds because of the additive homomorphism of the scheme $\mathsf{PolyCom}$. This thus proves the completeness property of the folding scheme.

Knowledge Soundness. For the knowledge soundness property, consider an adversarially chosen circuit F and two committed instances $\mathsf{u}_0 = (\boldsymbol{\beta}_0, \mathbf{x}_0, com_{\mathbf{w}0}, y_0)$ and $\mathsf{u}_1 = (\perp, \mathbf{x}_1, com_{\mathbf{w}1}, 0)$.

We will leverage the Forking Lemma (Lemma 1 in the Nova paper [33]) to prove the knowledge soundness property. That is, we prove that there exists a PPT algorithm $\mathcal{X}$ such that when given the public parameter pp, circuit F, and a tree of accepting transcripts and the corresponding folded instance-witness pairs, output a satisfying witness with probability $1 - \mathsf{negl}(\lambda)$.

Suppose $\mathcal{X}$ is provided with two transcripts (τ_0, τ_1) with the same initial message $\boldsymbol{\beta}_1 \in \mathbb{F}^{\log n}$ from $\mathcal{V}$. Note that a transcript τ_i for $i \in \{0, 1\}$ additionally comes with an accepting witness $\tau_i.(\mathbf{w}, r_{\mathbf{w}})$ and the verifier's randomness γ. Thus, $\mathcal{X}$ can retrieve $(\mathbf{w}_0, \mathbf{w}_1)$ such that

$$(1 - \tau_i.\gamma) \cdot \mathbf{w}_0 + (\tau_i.\gamma) \cdot \mathbf{w}_1 = \tau_i.\mathbf{w} \tag{20}$$

for $i \in \{0,1\}$. Using the same approach, $\mathcal{X}$ can interpolate for $(r_{\mathbf{w}0}, r_{\mathbf{w}1})$. Furthermore, recall that the two transcripts (τ_0, τ_1) contain the same $\boldsymbol{\beta}_1$ for instance u_1.

Now we need to argue that $(\mathbf{w}_0, r_{\mathbf{w}0})$ and $(\mathbf{w}_1, r_{\mathbf{w}1})$ are indeed satisfying instances for u_0 and u_1, respectively.

We first show that the retrieved witnesses are valid openings to the corresponding commitments in the instance. For $i \in \{0,1\}$, because $\tau_i.\mathbf{w}$ and $\tau_i.r_{\mathbf{w}}$ are part of a satisfying witness, and because $\mathsf{PolyCom}$ is additive homomorphic, we should have

$$\begin{aligned}
&(1-\tau_i.\gamma)\cdot \mathsf{PolyCom}(\mathsf{pp}_{\mathbf{w}}, \mathbf{w}_0, r_{\mathbf{w}0}) + \tau_i.\gamma \cdot \mathsf{PolyCom}(\mathsf{pp}_{\mathbf{w}}, \mathbf{w}_1, r_{\mathbf{w}1}) \\
&= \mathsf{PolyCom}(\mathsf{pp}_{\mathbf{w}}, (1-\tau_i.\gamma)\cdot \mathbf{w}_0 + \tau_i.\gamma \cdot \mathbf{w}_1, (1-\tau_i.\gamma)\cdot r_{\mathbf{w}0} + \tau_i.\gamma \cdot r_{\mathbf{w}10}) \\
&= \mathsf{PolyCom}(\mathsf{pp}_{\mathbf{w}}, \tau_i.\mathbf{w}, \tau_i.r_{\mathbf{w}}) \\
&= com_{\tau_i.\mathbf{w}} \\
&= (1-\tau_i.\gamma)\cdot com_{\mathbf{w}0} + \tau_i.\gamma \cdot com_{\mathbf{w}1}
\end{aligned}$$

Interpolating, the following must hold:

$$\mathsf{PolyCom}(\mathsf{pp}_{\mathbf{w}}, \mathbf{w}_0, r_{\mathbf{w}_0}) = com_{\mathbf{w}0} \tag{21}$$
$$\mathsf{PolyCom}(\mathsf{pp}_{\mathbf{w}}, \mathbf{w}_1, r_{\mathbf{w}_1}) = com_{\mathbf{w}1} \tag{22}$$

Next, we need to show that

- $\mathbf{w}_0$ satisfies $F^*(\boldsymbol{\beta}_0, \mathbf{x}_0, \mathbf{w}_0) = y_0$, and
- $\mathbf{w}_1$ satisfies $F^*(\boldsymbol{\beta}_1, \mathbf{x}_1, \mathbf{w}_1) = 0$, where $\mathcal{X}$ can extract $\boldsymbol{\beta}_1$ from transcripts (τ_0, τ_1) as mentioned earlier.

To prove these claims, denote $f^*(t) = F^*(\boldsymbol{\beta}(t), \mathbf{x}(t), \mathbf{w}(t))$ where $\boldsymbol{\beta}(t) = (1-t)\cdot\boldsymbol{\beta}_0 + t\cdot\boldsymbol{\beta}_1$, and $\mathbf{x}(t)$ and $\mathbf{w}(t)$ are defined similarly. On the other hand, denote $y(t)$ as the degree-D^* polynomial obtained by interpolation through points $\{(0,y_0),(1,0),(2,y(2)),(3,y(3)),..,(D^*,y(D^*))\}$. If $f^*(t)$ and $y(t)$ are two different degree-D^* polynomials over field $\mathbb{F}$, the probability that for a random $\tau_i.\gamma$, $f^*(\tau_i.\gamma) = y(\tau_i.\gamma)$ is at most $\frac{D^*}{|\mathbb{F}|}$, according to the Schwartz-Zippel Lemma [44]. Since both (τ_0, τ_1) are both accepting transcripts, $F^*(\boldsymbol{\beta}(\tau_i.\gamma), \mathbf{x}(\tau_i.\gamma), \mathbf{w}(\tau_i.\gamma)) = f^*(\tau_i.\gamma) = y(\tau_i.\gamma)$ must hold for $i \in \{0,1\}$. This indicates that $f^*(t)$ and $y(t)$ are the same degree-D^* polynomial with probability $1 - \frac{D^*}{|\mathbb{F}|} = 1 - \mathsf{negl}(\lambda)$. Note that by construction, $y(0) = y_0$ and $y_1 = 0$. We can thus conclude that $F^*(\boldsymbol{\beta}_0, \mathbf{x}_0, \mathbf{w}_0) = F^*(\boldsymbol{\beta}(0), \mathbf{x}(0), \mathbf{w}(0)) = f^*(0) = y(0) = y_0$. Similarly, we must have $F^*(\boldsymbol{\beta}_1, \mathbf{x}_1, \mathbf{w}_1) = y(1) = 0$. With these two claims proved, the knowledge soundness properly follows from the Forking Lemma for folding schemes (Lemma 1 in [33]).

Public Coin. Obviously, the construction uses only public randomness. Hence, the public coin property holds.

C.2 Proofs for the Interstellar IVC Protocol

Below we prove Theorem 16, which claims the completeness, knowledge soundness, and effiency properties of the Interstellar IVC protocol presented in Construction 15.

Proof. Let us first prove the completeness property of the IVC protocol, and then knowledge soundness and efficiency.

Completeness. We will prove the completeness property by induction. Consider $(F, i+1, \mathbf{z}_0, \mathbf{z}_{i+1})$ and the corresponding inputs $(\mathbf{z}_i, \boldsymbol{\omega}_i)$ such that $\mathbf{z}_{i+1} = F(\mathbf{z}_i, \boldsymbol{\omega}_i)$. Let $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, and let $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F)$. Now, consider a proof Π_i such that $\mathcal{V}(\mathsf{vk}, i, \mathbf{z}_0, \mathbf{z}_i, \Pi_i) = 1$. We need to show that given $\Pi_{i+1} \leftarrow \mathcal{P}(\mathsf{pk}, (i, \mathbf{z}_0, \mathbf{z}_i), \boldsymbol{\omega}_i, \Pi_i)$, that $\mathcal{V}(\mathsf{vk}, i+1, \mathbf{z}_0, \mathbf{z}_{i+1}, \Pi_{i+1}) = 1$ holds with probability 1.

Base Case $(i = 0)$: Suppose that the prover is provided Π_0 and $\mathcal{V}(\mathsf{vk}, 0, \mathbf{z}_0, \mathbf{z}_0, \Pi_0) = 1$. By the based case of $\mathcal{P}$ and F^*, we have $\Pi_1 = ((\mathsf{U}_1, \mathbf{W}_1), (\mathsf{u}_1, \mathbf{w}_1), r_1)$ for some $(\mathsf{u}_1, \mathbf{w}_1)$, where $(\mathsf{U}_1, \mathbf{W}_1) = ((\boldsymbol{\beta} = \mathbf{0}, \mathbf{x} = \mathbf{0}, com_{\mathbf{0}}, y = 0), \mathbf{w}_0 = \mathbf{0})$ and $com_{\mathbf{0}}$ is the commitment to the zero vector $\mathbf{0}$. We note that in Equation 2, the index i starts from 1 instead of 0. This way, if $\boldsymbol{\beta}$ is a zero vector, $\mathbf{s}$ will also be a zero vector, which indcates $y = \mathbf{s} \cdot \mathbf{y} = \mathbf{0} \cdot \mathbf{y} = 0$. Thus, $(\mathsf{U}_1, \mathbf{W}_1)$ as defined above is a satisfying instance-witness pair to start with. In addition, $(\mathsf{u}_1, \mathbf{w}_1)$ satisfies F^* by construction. Moreover, by definition $\mathsf{u}_1.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, 1, \mathbf{z}_0, F(\mathbf{z}_0, \boldsymbol{\omega}_0), \mathsf{u}_\perp, r_1)$. Then, we have $\mathcal{V}(\mathsf{pp}, 1, \mathbf{z}_0, \mathbf{z}_1, \Pi_1) = 1$ by the construction of $\mathcal{V}$.

Inductive Step $(i \geq 1)$: Assume that for $\Pi_i = ((\mathsf{U}_i, \mathbf{W}_i), (\mathsf{u}_i, \mathbf{w}_i), r_i)$, we have $\mathcal{V}(\mathsf{vk}, i, \mathbf{z}_0, \mathbf{z}_i, \Pi_i) = 1$, and that $\mathcal{P}$ outputs $\Pi_{i+1} = ((\mathsf{U}_{i+1}, \mathbf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathbf{w}_{i+1}), r_{i+1})$. By construction, we have $(\mathsf{U}_{i+1}, \mathbf{W}_{i+1}) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}, (\mathsf{U}_i, \mathbf{W}_i), (\mathsf{u}_i, \mathbf{w}_i))$. Thus, by the completeness of the underlying folding scheme, we have that $(\mathsf{U}_{i+1}, \mathbf{W}_{i+1})$ is a satisfying instance-witness pair. In addition, we have $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, \mathbf{z}_0, \mathbf{z}_i, \mathsf{U}_i, r_i)$. Therefore, $\mathcal{P}$ can construct a satisfying instance-witness pair $(\mathsf{u}_{i+1}, \mathbf{w}_{i+1})$ that represents the correct execution of F' on input $(\mathsf{U}_i, \mathsf{u}_i, (i, \mathbf{z}_0, \mathbf{z}_i), \boldsymbol{\omega}_i, r, r_{i+1})$, where $r_{i+1} \in \mathbb{F}$ is a field element randomly sampled by $\mathcal{P}$. By construction, we have

$$\begin{aligned}\mathsf{u}_{i+1}.\mathsf{x} &= \mathsf{hash}(\mathsf{vk}, i+1, \mathsf{u}_i.\mathbf{z}_0, F(\mathbf{z}_i, \boldsymbol{\omega}_i), \mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i), r_{i+1}) \\ &= \mathsf{hash}(\mathsf{vk}, i+1, \mathsf{u}_i.\mathbf{z}_0, \mathbf{z}_{i+1}, \mathsf{U}_{i+1}, r_{i+1})\end{aligned}$$

by the completeness of the underlying folding scheme. Finally, with the above $\mathsf{u}_{i+1}.\mathsf{x}$ and that $(\mathsf{U}_i, \mathbf{W}_i)$ and $(\mathsf{u}_i, \mathbf{w}_i)$ are both satisfying instance-witness pairs as proved above, $\mathcal{V}(\mathsf{vk}, i+1, \mathbf{z}_0, \mathbf{z}_{i+1}, \Pi_{i+1}) = 1$ must hold according to the definition of $\mathcal{V}$ in Construction 15. This proves the completeness of the Interstellar IVC protocol.

Knowledge Soundness. The following proof largely follows the knowledge soundness proof of Lemma 10 in [33] with some adaptation for our IVC protocol. Here we prove the knowledge soundness property through induction. Consider

an expected polynomial time adversary $\mathcal{P}^*$ that outputs a function F on input pp, and let $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F)$. Suppose given a constant n, $\mathcal{P}^*$ additionally outputs $(\mathbf{z}_0, \mathbf{z}, \mathit{\Pi})$ such that $\mathcal{V}(\mathsf{vk}, n, \mathbf{z}_0, \mathbf{z}, \mathit{\Pi}) = 1$ with probability ϵ. Now we need to construct an expected polynomial time extractor $\mathcal{E}$ which takes input $(\mathsf{pp}, \mathbf{z}_0, \mathbf{z})$, outputs $(\boldsymbol{\omega}_0, .., \boldsymbol{\omega}_{n-1})$ such that we have $\mathbf{z}_n = \mathbf{z}$ with probability $\epsilon - \mathsf{negl}(\lambda)$ by computing $\mathbf{z}_i \leftarrow F(\mathbf{z}_{i-1}, \boldsymbol{\omega}_{i-1})$.

We show by induction that $\mathcal{E}$ can construct an expected polynomial time extractor $\mathcal{E}_i(\mathsf{pp})$ that outputs $((\mathbf{z}_i, .., \mathbf{z}_{n-1}), (\boldsymbol{\omega}_i, .., \boldsymbol{\omega}_{n-1}), \mathit{\Pi}_i)$ such that for all $j \in \{i+1, .., n\}$, we have $\mathbf{z}_j = F(\mathbf{z}_{j-1}, \boldsymbol{\omega}_{j-1})$ and $\mathcal{V}(\mathsf{vk}, i, \mathbf{z}_0, \mathbf{z}_i, \mathit{\Pi}_i) = 1$ for $\mathbf{z}_n = \mathbf{z}$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Then, in the end, when $i = 0$, $\mathcal{V}$ checks $\mathbf{z}_0 = \mathbf{z}_i$, and the values $(\boldsymbol{\omega}_0, .., \boldsymbol{\omega}_{n-1})$ retrieved by $\mathcal{E}_0(\mathsf{pp})$ are such that $\mathbf{z}_{i+1} = F(\mathbf{z}_i, \boldsymbol{\omega}_i)$ holds for all $i \geq 1$ given $\mathbf{z}_n = \mathbf{z}$. In the following, we will use the following strategy. First, we assume the existence of $\mathcal{E}_i$ which satisfies the inductive hypothesis. Then, we construct $\mathcal{E}_{i-1}$ by letting $\mathcal{E}_i$ construct an adverarial non-interactive folding scheme $\tilde{\mathcal{P}}_{i-1}$, which guarantees an extractor $\tilde{\mathcal{E}}_{i-1}$ for the non-interactive folding scheme. Next, we use $\tilde{\mathcal{E}}_{i-1}$ to construct $\mathcal{E}_{i-1}$ for the induction.

Base Case $(i = n)$: Let $\mathcal{E}_n(\mathsf{pp}; \rho)$ output $(\perp, \perp, \mathit{\Pi}_n)$ where $\mathit{\Pi}_n$ is the output of $\mathcal{P}^*(\mathsf{pp}; \rho)$. By assumption, $\mathcal{E}_n$ should succeed with probability ϵ in expected polynomial time.

Inductive step $(1 \leq i < n)$: Suppose $\mathcal{E}$ can construct an expected polynomial time extractor $\mathcal{E}_i$ that outputs $((\mathbf{z}_i, .., \mathbf{z}_{n-1}), (\boldsymbol{\omega}_i, .., \boldsymbol{\omega}_{n-1}))$, and $\mathit{\Pi}_i$ that satisfies the inductive hypothesis. To construct an extractor $\mathcal{E}_{i-1}$, $\mathcal{E}$ first constructs an adversary $\tilde{P}_{i-1}$ for the non-interactive folding schemes as follows:

$\tilde{\mathcal{P}}_{i-1}(\mathsf{pp}; \rho)$:
(1) Let $((\mathbf{z}_i, .., \mathbf{z}_{n-1}), (\boldsymbol{\omega}_i, .., \boldsymbol{\omega}_{n-1}), \mathit{\Pi}_i) \leftarrow \mathcal{E}_i(\mathsf{pp}; \rho)$.
(2) Parse $\mathit{\Pi}_i$ as $((\mathsf{U}_i, \mathbf{W}_i), (\mathsf{u}_i, \mathbf{w}_i), r_i)$.
(3) Parse $\mathbf{w}_i$, the witness input to F^* to retrieve $(\mathsf{U}_{i-1}, \mathsf{u}_{i-1}, r_{i-1})$.
(4) Output $(\mathsf{U}_{i-1}, \mathsf{u}_{i-1})$ and $(\mathsf{U}_i, \mathbf{W}_i)$.

By the inductive hypothesis, we have $\mathcal{V}(\mathsf{vk}, i, \mathbf{z}_0, \mathbf{z}_i, \mathit{\Pi}_i) = 1$, where $\mathit{\Pi}_i \leftarrow \mathcal{E}_i(\mathsf{pp})$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Therefore, by the verifier check, $(\mathsf{u}_i, \mathbf{w}_i)$ and $(\mathsf{U}_i, \mathbf{W}_i)$ are satisfying instance-witness pairs, and that $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, \mathbf{z}_0, \mathbf{z}_i, \mathsf{U}_i, r_i)$. Because $\mathcal{V}$ randomly samples $\boldsymbol{\beta}_i \in \mathbb{F}^{\log n}$ and set $\mathbf{u}_i.\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_i$, if $(\mathsf{u}_i, \mathbf{w}_i)$ passes the check, then $\mathbf{w}_i$ is indeed a satisfying assignment for F^* (not just a trivially satisfying witness). Moreover, according to Lemma 5, if F^* is satisfied, then F' is also satisfied with probability at least $1 - \frac{\log |F'|}{|\mathbb{F}|} = 1 - \mathsf{negl}(\lambda)$.

Next, by the construction of F^* and the binding property of the hash function, we have $\mathsf{U}_i = \mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}_{i-1}, \mathsf{u}_{i-1})$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Thus, by the union bound, $\tilde{\mathcal{P}}_{i-1}$ succeeds in producing an accepting folded instance-witness pair $(\mathsf{U}_i, \mathbf{W}_i)$ for instances $(\mathsf{U}_{i-1}, \mathsf{u}_{i-1})$ with probability $1 - (1 - (1 - \mathsf{negl}(\lambda))) - (1 - (\epsilon - \mathsf{negl}(\lambda))) = \epsilon - 2 \cdot \mathsf{negl}(\lambda) = \epsilon - \mathsf{negl}(\lambda)$ in expected polynomial time. Then, by the knowledge soundness of the underlying folding scheme there exists an extractor $\tilde{\mathcal{E}}_{i-1}$ that outputs $(\mathbf{w}_{i-1}, \mathbf{W}_{i-1})$ such that $(\mathsf{u}_{i-1}, \mathbf{w}_{i-1})$ and $(\mathsf{U}_{i-1}, \mathbf{W}_{i-1})$ satisfy F' with probability $\epsilon - \mathsf{negl}(\lambda)$ in expected polynomial time.

Then, with $\tilde{\mathcal{P}}_{i-1}$ and $\tilde{\mathcal{E}}_{i-1}$, $\mathcal{E}$ constructs an expected polynomial time $\mathcal{E}_{i-1}$ as follows:

$\tilde{\mathcal{E}}_{i-1}(\mathsf{pp};\rho)$:

(1) $((\mathsf{U}_{i-1},\mathsf{u}_{i-1}),(\mathsf{U}_i,\mathbf{W}_i)) \leftarrow \tilde{\mathcal{P}}_{i-1}(\mathsf{pp};\rho)$;

(2) Retrieve $((\mathbf{z}_i,..,\mathbf{z}_{n-1}),(\boldsymbol{\omega}_i,...,\boldsymbol{\omega}_{n-1}),\Pi_i,r_{i-1})$ from the state of $\tilde{\mathcal{P}}_{i-1}$;

(3) Parse $\Pi_i.\mathbf{w}_i$ to retrieve $\mathbf{z}_i - 1$ and $\boldsymbol{\omega}_{i-1}$;

(4) Let $(\mathbf{w}_{i-1},\mathbf{W}_{i-1}) \leftarrow \tilde{\mathcal{E}}_{i-1}(\mathsf{pp})$;

(5) Let $\Pi_{i-1} \leftarrow ((\mathsf{U}_{i-1},\mathbf{W}_{i-1}),(\mathsf{u}_{i-1},\mathbf{w}_{i-1}),r_{i-1})$;

(6) Output $((\mathbf{z}_{i-1},..,\mathbf{z}_{n-1}),(\boldsymbol{\omega}_{i-1},...,\boldsymbol{\omega}_{n-1}),\Pi_{i-1})$.

Let us first reason that the outputs $(\mathbf{z}_{i-1},..,\mathbf{z}_{n-1})$ and $(\boldsymbol{\omega}_{i-1},...,\boldsymbol{\omega}_{n-1})$ are valid. By the inductive hypothesis, we already have that for all $j \in \{i+1,..,n\}$, $\mathbf{z}_j = F(\mathbf{z}_{j-1},\boldsymbol{\omega}_{j-1})$ and that $\mathcal{V}(\mathsf{vk},i,\mathbf{z}_0,\mathbf{z}_i,((\mathsf{U}_i,\mathbf{W}_i),(\mathsf{u}_i,\mathbf{w}_i),r_i)) = 1$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Because $\mathcal{V}$ checks that

$$\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk},i,\mathbf{z}_0,\mathbf{z}_i,\mathsf{U}_i,r_i) \tag{23}$$

by the construction of F^*, the binding property of the hash function, and the union bound, we have $F(\mathbf{z}_{i-1},\boldsymbol{\omega}_{i-1}) = \mathbf{z}_i$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Next, we argue that Π_{i-1} is valid. Because $(\mathbf{u}_i,\mathbf{w}_i)$ satisfies F', and $(\mathsf{U}_{i-1},\mathsf{u}_{i-1})$ were retrieved from $\mathbf{w}_i$, by the binding property of the hash function, and by Equation 23, we have $\mathsf{u}_{i-1}.\mathsf{x} = \mathsf{hash}(\mathsf{vk},i-1,\mathbf{z}_0,\mathbf{z}_{i-1},\mathsf{U}_{i-1},r_{i-1})$ and $\mathsf{u}_{i-1}.\boldsymbol{\beta} \neq \mathbf{0}$. Additionally, when $i = 1$, by the base case check we have $\mathbf{z}_{i-1} = \mathbf{z}_0$. Because $\tilde{\mathcal{E}}_{i-1}$ succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$. We have $\mathcal{V}(\mathsf{vk},i-1,\mathbf{z}_0,\mathbf{z}_{i-1},\Pi_{i-1}) = 1$ with probability $\epsilon - \mathsf{negl}(\lambda)$. This thus proves the knowledge soundness of the Interstellar IVC protocol.

Efficiency. In Construction 14, regarless of the PolyCom we employ, the prover first needs to perform $O(\log n)$ hashes to generate $\boldsymbol{\beta}_1 = (\beta_0,..,\beta_k,..,\beta_{\log n-1}) \in \mathbb{F}^{\log n}$. Since $n \leq |F|$, the prover performs at most $O(\log|F|)$ hashes. Then, the prover evaluates $F^*(\boldsymbol{\beta}(t),\mathbf{x}(t),\mathbf{w}(t))$ at D^*-1 points $(2,3,..,D^*)$ to obtain $\{y(2),y(3),..,y(D^*)\}$. This requires $O(D^*|F^*|)$ field operations. According to Lemma 7, $O(|F^*|) = O(F'|)$. Hence $O(D^*|F^*|) = O(D^*|F|)$. Next, one more hash is performed to generate γ. With γ, the prover can use the barycentric method to evaluate $y(\lambda)$, where the online processing takes $O_\lambda(D^*)$ field operations. Then, the prover needs to calculate $\boldsymbol{\beta}_\lambda$ and $\mathbf{x}_\gamma$, each of which takes at most $O(|F|)$ field operations. In summary, these calculations require $O(\log|F|)$ hashes and $O(D^*|F|)$ field operations.

Besides that, the prover also calculates $com_{\mathbf{w}\gamma}$ and commits to $\mathbf{w}_1$. The time complexity of these two operations depends on the PolyCom scheme we choose. If we instantiate PolyCom with the either the Bulletproofs or Dory commitment scheme, generating $com_{\mathbf{w}\gamma} \leftarrow (1-\gamma)\cdot com_{\mathbf{w}0} + \gamma \cdot com_{\mathbf{w}1}$ involves one single group scalar multiplication. On the other hand, committing to $\mathbf{w}_1$ requires a single gourp MSM of size $|\mathbf{w}|$.

C.3 Proofs for the IVC SNARK Protocol

Below we prove Theorem 17, which claims the completeness and knowledge soundness of Construction 38.

Proof. Completeness and knowledge soundness both hold due to the completeness and knowledge soundness of the underlying SNARK and the non-interactive folding scheme in Construction 14. The protocol achieves succinctness due to the succinctness of the commitment scheme used by the non-interactive folding scheme, the succinctness of the hash function, and the succinctness of the underlying non-interactive argument.

Below we prove Theorem 18, which claims the existence of a SNARK in the random oracle model for the circuit satisfiability problem for circuit F.

Proof. We adopt the GKR-based Libra Polynomial IOP [51] for the circuit F^* and replace the polynomial oracles with the selected $\mathsf{PolyCom}$ scheme, which the prover and verifier use to simulate ideal queries to a committed oracle. Since the Libra polynomial IOP is a public coin protocol, we can apply the Fiat-Shamir transformation to achieve non-interactivity in the random oracle model.

In terms of efficiency, in the Libra polynomial IOP, the prover runs in linear time $O_\lambda(|F^*|) = O_\lambda(|F|)$ for any circuit (including non-uniform circuits). Also, as explained by Remark 19, there is an improvement which we can apply to make the IOP verifier's online run time and proof size both in $O_\lambda(d^* \cdot \log |F|)$ even for non-uniform circuits, where d^* is the number of layers in F^*. Combining the polynomial IOP run time with various $\mathsf{PolyCom}$ schemes, we have the following conclusions:

- If we adopt Bulletproofs based $\mathsf{PolyCom}_{\mathsf{BP}}$, the commitment to the witness $\mathbf{w}$ takes $O_\lambda(|\mathbf{w}|)$ time, and produces an $O_\lambda(1)$-sized commitment. The prover time for generating an $O_\lambda(\log |\mathbf{w}|)$-sized evaluation proof is $O_\lambda(|\mathbf{w}|)$, which takes the verifier $O_\lambda(|\mathbf{w}|)$ time to verify. Since $|\mathbf{w}| \leq |F|$, combining the prover and verifer runtime for the IOP protocol, we have that the prover runs in time $O_\lambda(|F|)$; the proof length is $O_\lambda(d^* \log |F|)$; and the verifier online run time is in $O_\lambda(d^* \log |F| + |\mathbf{w}|)$.
- If we adopt Dory based $\mathsf{PolyCom}_{\mathsf{Dory}}$, the commitment to the witness $\mathbf{w}$ takes $O_\lambda(|\mathbf{w}|)$ time, and produces an $O_\lambda(1)$-sized commitment. The prover time for generating an $O_\lambda(\log |\mathbf{w}|)$-sized evaluation proof is $O_\lambda(|\mathbf{w}|)$, which takes the verifier $O_\lambda(\log |\mathbf{w}|)$ time to verify. Since $|\mathbf{w}| \leq |F|$, combining the prover and verifer runtime for the IOP protocol, we have that the prover runs in time $O_\lambda(|F|)$; the proof length is $O_\lambda(d^* \log |F|)$; and the verifier online run time $O_\lambda(d^* \log |F|)$.

C.4 Proofs for the Interstellar+ Protocol

Below we prove Theorem 22, which claims the completeness, knowledge soundness, and efficiency of Construction 21.

Proof. **Completeness**. Since the satisfying witness-instance pair of G^* and that of F^* have the same format, we can follow the steps of the completeness proof of Theorem 13. The details are omitted here, but it is worth mentioning that $v_i + \tau = 0$ could render $\frac{1}{v_i+\tau}$ undefined. Thus, similar to Protostar, we set $h_i = 0$ if $\frac{1}{v_i+\tau} = 0$. $\{g_i \mid i = 0, .., P-1\}$ are handled in a similar fashion. In this way, the lookup checks 5 and 6 can always be satisfied with a correct witness.

Knowledge Soundness. For knowledge soundness we need to prove that given a tree of accepting transcripts, there is an polynomial time extractor that can extract $\mathbf{w}_0$, $\mathbf{w}_1$, $r_{\mathbf{w}0}$, $r_{\mathbf{w}1}$, $\mathbf{m}_1$, τ_1, $\mathbf{h}_1$, $\mathbf{g}_1$, and $\boldsymbol{\beta}_1$, such that:

- $(\mathbf{w}_0, r_{\mathbf{w}0})$ and $(\mathbf{w}_1, r_{\mathbf{w}1})$ are witnesses of $\mathsf{u}_0 = (\boldsymbol{\beta}_0, \mathbf{x}_{A0}, com_{\mathbf{w}0}, y_0)$ and $\mathsf{u}_1 = (\beta_1, \mathbf{x}_{A1}, com_{\mathbf{w}1}, 0)$, respectively, with probability $1 - \mathsf{negl}(\lambda)$. Here $\mathbf{x}_{A1} = (\mathbf{x}_1, \mathbf{p}_1, \tau_1, \mathbf{m}_1, \mathbf{h}_1, \mathbf{g}_1)$.
- The lookup check per Lemma 20 $\sum_{i=0}^{l-1} \frac{1}{X+v_i} = \sum_{i=0}^{P-1} \frac{m_i}{X+p_i}$ holds with probability $1 - \mathsf{negl}(\lambda)$.

Once these two claims are proven, we can apply the forking lemma [33] to prove the knowledge soundness.

The first claim can be proved similarly to the knowledge soundness proof for Theorem 13. Note that in Construction 21, $\mathbf{w}_\gamma$ and $r_{\mathbf{w}\gamma}$ are sent after from the tree of accepting transcripts, thus we can pick two transcripts with the same ancestors containing $\mathbf{m}_1$, τ_1, $\mathbf{h}_1$, $\mathbf{g}_1$, and $\boldsymbol{\beta}_1$. Then, using the same interpolation method as in the knowledge soundness proof for Theorem 13, we can solve a set of linear equations to obtain $(\mathbf{w}_0, r_{\mathbf{w}0})$ and $(\mathbf{w}_1, r_{\mathbf{w}1})$ which satisfy u_0 and u_1, respectively, with probability $1 - \mathsf{negl}(\lambda)$. The details are omitted here.

For the second claim, we can prove it using a strategy similar to the proof for Protostar's Lemma 5 [11]. From the transcript tree, we take $2(l + P)$ transcripts that all have the same $\mathbf{v}_1$ and $\mathbf{m}_1$ as the initial messages, but different $(\tau_1, \mathbf{h}_1 \in \mathbb{F}^l, \mathbf{g}_1 \in \mathbb{F}^P)$ as the follow-up messages. By the pigeonhole principle, there must exist a subset $S \subseteq \{0, 1, 2(l + P) - 1\}$ transcripts such that $|S| = l + P$ and $w_{1i} + \tau_1 \neq 0$ for all $i \in \{0, 1, .., l-1\}$ and $j \in S$ and $p_{1i} + \tau_1 \neq 0$ for all $i \in \{0, 1, .., P-1\}$ and $j \in S$. For these transcripts, we have $h_{1i} = \frac{1}{v_{1i}+\tau_1}$ and $g_{1i} = \frac{m_{1i}}{p_{1i}+\tau_1}$ with probability $1 - \mathsf{negl}(\lambda)$. The probability is not strictly 1 because the verifier circuit checks conditions 5 and 6 indirectly by checking whether $y = \mathbf{s} \cdot \mathbf{y} = 0$, which introduces a $\mathsf{negl}(\lambda)$ soundness error (see Lemma 5). Now, let us define the following polynomial that has a degree $l + P - 1$:

$$p(X) = \prod_{k=1}^{l}(X + w_{1k}) \cdot \prod_{j=1}^{P}(X + p_{1j}) \cdot (\sum_{i=0}^{l-1} \frac{1}{X + v_i} = \sum_{i=0}^{P-1} \frac{m_i}{X + p_i})$$

We have shown that there are $l + P$ values of τ_1 for which $p(X = \tau_1) = 0$. Since $p(X)$ has degree $l + P - 1$, $p(X)$ must be a zero polynomial. Therefore, $\sum_{i=0}^{l-1} \frac{1}{X+v_i} = \sum_{i=0}^{P-1} \frac{m_i}{X+p_i}$ holds with probability $1 - \mathsf{negl}(\lambda)$. According

to Lemma 20, we must have $\{v_i | i = 0, 1, .., l-1\} \subset \mathbf{p}$ holds with probability $1 - \mathsf{negl}(\lambda)$.

Finally, applying the forking lemma, we can conclude Construction 21 is knowledge sound.

Next, we prove Theorem 23, which claims that the protocol for folding multiple instances at once achieves the completeness and knowledge soundness properly, and provides its runtime efficiency.

Proof. **Completeness**. As the interactive protocol for folding multiple instances at once in Construction 39 is a natural extension of Construction 12, we can simply follow the completeness proof of Theorem 13. The details are omitted here.

Knowledge Soundness. Similar to the completeness property, we can also follow the proving stratgy of the knowledge soundness proof for Theorem 13. The only caveat is that $y(t)$ now has degree kD^*, and the Schwartz-Zippel error becomes $\frac{kD^*}{|\mathbb{F}|}$. As a result, our construction requires $\frac{kD^*}{|\mathbb{F}|}$ in $O(\mathsf{negl}(\lambda))$, which places an upper bound on k. The other caveat is that we reuse the same $\boldsymbol{\beta}$ for u_1, u_2, .., u_{k-1}. We note that by construction, $\mathsf{u}_1.y = \mathsf{u}_2.y = .. = \mathsf{u}_{k-1}.y = 0$, meaning that $F^*(\boldsymbol{\beta}, \mathsf{u}_i.\mathbf{x}, \mathsf{u}_i.\mathbf{w}) = 0$ holds for $i = 1, 2, .., k-1$. Per Lemma 5, this indicates for each $i \in \{1, 2, .., k-1\}$, $F'(\mathbf{x}'_i, \mathbf{w}'_i) = \mathbf{y}'_i$ holds with probability at least $1 - \frac{\log |F'|}{|\mathbb{F}|}$. By the union bound, the probability that $F'(\mathbf{x}'_i, \mathbf{w}'_i) = \mathbf{y}'_i$ holds for all $= 1, 2, .., k-1$ is at least $1 - \frac{k \log |F'|}{|\mathbb{F}|}$, which is $1 - \mathsf{negl}(\lambda)$ if $\frac{k}{|\mathbb{F}|}$ is in $O(\mathsf{negl}(\lambda))$. The remainder of the proof are largely the same as the knowledge soundness proof for Theorem 13 with linear combinations such as $(1-t)\cdot\mathbf{w}_0 + t\cdot\mathbf{w}_1$ replaced by its generalization $\sum_{j=0}^{k-1} \delta(t, j) \cdot \mathbf{w}_j$. The details are omitted here.

Public Coin. Obviously, the construction uses only public randomness. Hence, the public coin property holds.

Constant Folded Instance Size. Obviously, the folded instance still has the same size as u_0, whose size is in $O_\lambda(|F^*|) = O_\lambda(|F|)$ (Lemma 7). Therefore, the size of the folded instance is a constant.

Efficiency. In Construction 40, regarless of the $\mathsf{PolyCom}$ we employ, the prover first needs to perform $O(\log n)$ hashes to generate $\{\beta_i \in \mathbb{F} | i = 0, 1, .., \log n - 1\}$. Since $n \leq |F|$, the prover performs at most $O(\log |F|)$ hashes. Then, the prover evaluates $F^*(\boldsymbol{\beta}(t), \mathbf{x}(t), \mathbf{w}(t))$ at $D^* - 1$ points $(k, k+1, .., kD^*)$ to obtain $\{y(k), y(k+1), .., y(kD^*)\}$. This requires $O(kD^*|F|)$ field operations. Per Lemma 7, $O(|F^*|) = O(|F|)$. Hence $O(kD^*|F^*|) = O(kD^*|F|)$. Next, one more hash is performed to generate γ. With γ, the prover can use the barycentric method to evaluate $y(\lambda)$, where the online processing takes $O(kD^*)$ field operations. Then, the prover needs to calculate $\boldsymbol{\beta}_\gamma$, $\mathbf{x}_\gamma$, and $\mathbf{w}_\gamma$. Again, using the barycentric evaluation method, generating each of these vectors takes at most $O(k|F|)$

field operations. In summary, these calculations require $O(\log |F|)$ hashes and $O(kD^*|F|)$ field operations.

Besides that, the prover also calculates $com_{\mathbf{w}\gamma}$ and commits to $\{\mathbf{w}_j \mid j = 1, 2, .., k\}$). If we instantiate $\mathsf{PolyCom}$ with the either the Bulletproofs or Dory commitment scheme, committing to $\{\mathbf{w}_j \mid j = 1, 2, .., k\}$) requires k group MSMs, each of size $|\mathbf{w}|$. On the other hand, generating $com_{\mathbf{w}\gamma} \leftarrow \sum_{j=0}^{k-1} \delta(\gamma, j) \cdot com_{\mathbf{w}j}$ involves k group scalar-multiplications and additions since each $com_{\mathbf{w}j}$ is a group element and $\delta(\gamma, j)$ is a constant.

Below we prove Theorem 42, which claims the completeness, knowledge soundness, and efficiency of Construction 41.

Proof. **Completeness**. The completeness proof basically follows the completeness proof for Theorem 13, with the argument applied to each individual sub-circuit. The proof is straightforward and the details are omitted here.

Knowledge Soundness. Similar to completeness, the knowledge soundness proof follows that of Theorem 13, with the argument applied to each individual sub-circuit. The only minor caveat is that if $i \neq pc$ and $i \neq I$, since the prover will reuse $\mathbf{w}_{acc}^{(i)}$ for $\mathbf{w}_{step}^{(i)}$, we have

$$\begin{aligned}\mathbf{w}^{(i)} &= (1-\gamma) \cdot \mathbf{w}_{acc}^{(i)} + \gamma \cdot \mathbf{w}_{step}^{(i)} \\ &= (1-\gamma) \cdot \mathbf{w}_{acc}^{(i)} + \gamma \cdot \mathbf{w}_{acc}^{(i)} \\ &= \mathbf{w}_{acc}^{(i)}\end{aligned}$$

Hence $\mathbf{w}_{step}^{(i)} = \mathbf{w}_{acc}^{(i)} = \mathbf{w}^{(i)}$ can be extracted directly from the transcript without solving a system of linear equations as in Formula 20. For $i = pc$ or $i = I$, we can still solve a Formula 20-like set of linear equations to obtain $\mathbf{w}_{step}^{(i)}$ and $\mathbf{w}_{acc}^{(i)}$.

Efficiency. As already analyzed in Appendix B.5, for $\mathbf{B}$, $\mathbf{X}$, $com_{\mathbf{W}}$, $\mathbf{Y}$, $\mathbf{W}$, and $r_{\mathbf{W}}$, only two elements get updated in each folding round. In particular, for $\mathbf{Y}$, only $y^{(pc)}(\gamma)$ and $y^{(I)}(\gamma)$ need to be calculated. Thus, the runtime of a folding step only depends on the size of the activated sub-circuit, plus the extra circuit for common computations. We therefore conclude that we can achieve the "a la carte" cost profile.

C.5 Details and Proofs for Collaborative Folding/IVC

Below we present the construction of the non-interactive collaborative folding scheme:

Construction 43 (Non-Interactive Collaborative Folding Scheme (NICFS)**)** *Under the random oracle model, non-interactivity can be achieved using the strong Fiat-Shamir transformation. Let ρ denote a random oracle sampled during parameter generation and provided to all parties. Let $(\mathsf{G}, \mathsf{K}, \mathsf{PC}, \mathsf{PS}, \mathsf{V})$ represented the interactive folding scheme defined above in Construction 26. A non-interactive folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{PC}, \mathcal{PS}, \mathcal{V})$ can be constructed as follows:*

- $\mathcal{G}(1^\lambda)$*: output* $\mathsf{pp} \leftarrow \mathsf{G}(1^\lambda)$.
- $\mathcal{K}(\mathsf{pp}, F)$*:* $\mathsf{vk} \leftarrow \rho(\mathsf{pp}, F)$ *and* $\mathsf{pk} \leftarrow (\mathsf{pp}, F^*, \mathsf{vk})$*; output* $(\mathsf{vk}, \mathsf{pk})$.
- $\mathcal{PC}(\mathsf{pk}, (\mathsf{u}_0, \{\mathbf{w}_0^j | j \in [k]\}), (\mathsf{u}_1, \{\mathbf{w}_1^j | j \in [k]\}))$*: Generate* $\log n$ *random values* $\beta_0 \leftarrow \rho(\mathsf{vk}, \mathsf{u}_0, \mathsf{u}_1)$, $\beta_1 \leftarrow \rho(\mathsf{vk}, \mathsf{u}_0, \mathsf{u}_1, \beta_0)$, .., $\beta_{\log n-1} \leftarrow \rho(\mathsf{vk}, \mathsf{u}_0, \mathsf{u}_1, \beta_0, .., \beta_{\log n-2})$. *Run* $\mathsf{CIFS.PC}$ *with these random values to create* com_y *and forward it to* $\mathcal{V}$. *Generate randomness* $\gamma \leftarrow \rho(\mathsf{vk}, \mathsf{u}_0, \mathsf{u}_1, \beta_0, .., \beta_{\log n-2}, com_y)$, *and forward it to* $\mathcal{PC}$ *to compute* $y(0)$, $y(1)$, *and* $y(\gamma)$ *and generate their corresponding evaluation proofs* π_{y0}, π_{y1}, *and* $\pi_{y\gamma}$ *via MPC. Forward* $(y(0), \pi_{y0})$, $(y(1), \pi_{y1})$ *and* $(y(\gamma), \pi_{y\gamma})$ *to* $\mathcal{V}$, *and then output the resulting folded instance and witness.*
- $\mathcal{V}(\mathsf{vk}, \mathsf{u}_0, \mathsf{u}_1)$*: runs* $\mathsf{CIFS.V}$ *with randomness* $(\beta_0, .., \beta_{\log n-1})$ *generated the same way as* $\mathcal{PC}$*; Receive* com_y *and generate* γ *the same way as* $\mathcal{P}$. *Receive* $(y(0), \pi_{y0})$, $(y(1), \pi_{y1})$ *and* $(y(\gamma), \pi_{y\gamma})$ *to* $\mathcal{V}$ *from* $\mathcal{PC}$. *Perform verification on these values and proofs. Output the resulting folded instance.*

Below we present the proof for Theorem 27, which claims that the interactive collaborative folding scheme achieves the completeness, knowledge soundness, and m-zero-knowledge property:

Proof. Completeness proof directly follows that of Theorem 13 and the completeness proof of the collaborative MSM protocol [24] and BGW protocol. We omit the details.

To prove knowledge soundness, we first follow the soundness proof of Theorem 13 to show that the extractor algorithm $\mathcal{X}$ can retrieve $\{\mathbf{w}_0^j, \mathbf{w}_1^j\}$ for each $j \in [k]$ by solving a similar set of linear equations, and the same apply to $\{com_{\mathbf{w}0}^j, com_{\mathbf{w}1}^j | j \in [k]\}$. Next we need to show that $\mathbf{w}_0$ satisfies $F^*(\boldsymbol{\beta}_0, \mathbf{x}_0, \{\mathbf{w}_0^j | j \in [k]\}) = y_0$, and that $\mathbf{w}_1$ satisfies $F^*(\boldsymbol{\beta}_1, \mathbf{x}_1, \{\mathbf{w}_1^j | j \in [k]\}) = y_1$. For these claims we can apply the same Schwartz-Zippel-based argument as in the soundness proof for Theorem 13. The only caveat is that instead of $\{y(0), y(1), .., y(D^*)\}$ as in Construction 12, in Construction 26, the verifier receives the commitment to $y(t)$, and the evaluation results at points $0, 1, \gamma$ and the corresponding evaluation proof. However, if we use an extractable polynomial commitment scheme, then knowledge soundness would hold, since $\mathcal{X}$ can extract $\{y(0), y(1), .., y(D^*)\}$ with high probability and thus we can fall back to the soundness proof for Theorem 13. Moreover, in the MPC setting with honest-majority and the corrupted servers are semi-honest, the servers only compute $y(t)$ and the zero-knowledge polynomial commitment and proofs. They do not help cheating clients hide inconsistency because evaluation proofs are knowledge-sound and verification is public. Thus an extractor $\mathcal{X}$ that rewinds the public-coin ICFS transcript, for example to obtain two accepting runs with different γ if needed, succeeds exactly as in the single-prover folding analysis.

To construct a simulator Sim_{cf} in order to prove the m-zero-knowledge property, we break the interaction messages among $\mathcal{V}$, $\mathcal{PC}$, and $\mathcal{PS}$ into two views, namely the view between $\mathcal{PC}$ and $\mathcal{PS}$, and the view between $\mathcal{PC}$ and $\mathcal{V}$:

- For the view between $\mathcal{PC}$ and $\mathcal{PS}$, given the assumption that at most $m < \frac{n}{2} - D^*$ servers are corrupted, and the corrupted servers are semi-honest, Sim_{cf} can employ an MPC simulator to simulate the view, using similar arguments as zkSaaS [24], since the only MPC computations in our protocol are the collaborative MSM, polynomial commitment, and the BGW protocol, same as in zkSaaS [24].
- The view between $\mathcal{PC}$ and $\mathcal{V}$ consists of the following messages: $\mathcal{V}$ first sends $\mathcal{PC}$ $(\beta_0, .., \beta_k, .., \beta_{\log n-1}) \in \mathbb{F}^{\log n}$. Next, $\mathcal{PC}$ sends com_y to $\mathcal{V}$. $\mathcal{V}$ then replies with random value $\gamma \in \mathbb{F}$. In response, $\mathcal{PC}$ sends $(y(0), \pi_{y0})$, $(y(1), \pi_{y1})$, $(y(\gamma), \pi_{y\gamma})$ to $\mathcal{V}$. To simulate this view, the simulator Sim_{cf} just needs to run the protocol to the end so that $\mathcal{V}$ receives $\gamma \in \mathbb{F}$ and $y(\gamma)$. She then constructs an alternative degree-D^* univariate polynomial $y'(t)$ which agrees with $y(t)$ only at 0, 1, and γ, that is, $y'(0) = y(0)$, $y'(1) = y(1)$, and $y'(\gamma) = y(\gamma)$. Next, she rewinds the protocol to the second step, and send com'_y, the commitment to $y'(t)$ to $\mathcal{V}$ instead. $\mathcal{V}$ proceeds and send the same $\gamma \in \mathbb{F}$ back to $\mathcal{PC}$. Then, the simulator sends back $(y'(0), \pi'_{y0})$, $(y'(1), \pi'_{y1})$, $(y'(\gamma), \pi'_{y\gamma})$. To prove zero-knowledge, we note that com'_y is indistinguishable from com_y given the hiding property of the commitment scheme. Next, since $y'(0) = y(0)$, $y'(1) = y(1)$, and $y'(\gamma) = y(\gamma)$, the evaluation results are identical in the real and simulated view. Finally, due to the zero-knowledge property of the commitment scheme for $y'(t)$, π'_{y0}, π'_{y1}, and $\pi'_{y\gamma}$ are indistinguishable from π_{y0}, π_{y1}, and $\pi_{y\gamma}$. And since $\pi'_{y\gamma}$ is the correct evaluation proof of $y'(\gamma)$, and likewise for π'_{y0} and π'_{y1}, they will pass the validity checks imposed by the verifier. Therefore, we can conclude that the simulated view between $\mathcal{PC}$ and $\mathcal{V}$ is indistinguishable from the real view for honest verifiers.

In summary, since the two simulated views are indistinguishable from the corresponding real views, we can claim that honest verifier zero-knowledge can be achieved under the assumptions.