HyperNova: Recursive arguments for customizable constraint systems

Abhiram Kothapalli
Carnegie Mellon University

Srinath Setty
Microsoft Research

Abstract.

We introduce *HyperNova*, a new recursive argument for proving incremental computations whose steps are expressed with CCS (Setty et al. ePrint 2023/552), a customizable constraint system that simultaneously generalizes Plonkish, R1CS, and AIR without overheads. HyperNova makes four contributions, each resolving a major problem in the area of recursive arguments.

First, it provides a folding scheme for CCS where the prover's cryptographic cost is a *single* multi-scalar multiplication (MSM) of size equal to the number of variables in the constraint system, which is optimal when using an MSM-based commitment scheme. The folding scheme can fold multiple instances at once, making it easier to build generalizations of IVC such as PCD. Second, when proving program executions on stateful machines (e.g., EVM, RISC-V), 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 ("a la carte" cost profile). Third, we show how to achieve zero-knowledge for "free" and *without* the need to employ *zero-knowledge* SNARKs: we use a folding scheme to "randomize" IVC proofs. This highlights a new application of folding schemes. Fourth, we show how to efficiently instantiate HyperNova over a cycle of elliptic curves. For this, we provide a general technique, which we refer to as CycleFold, that applies to all modern folding-scheme-based recursive arguments.

1 Introduction

Incrementally verifiable computation (IVC) [66] is a powerful cryptographic primitive that allows a prover to produce a proof of the correct execution of a "long running" computation in an incremental fashion. For example, it enables the following workflow: The prover takes as input a proof π_i proving the the first i steps of its computation and then update it to produce a proof π_{i+1} proving the correct execution of the first $i+1$ steps. Crucially, the prover's work to update the proof does not depend on the number of steps executed thus far, and the verifier's work to verify a proof does not grow with the number of steps executed thus far. IVC has received recent, renewed interest as it enables a wide variety of applications in decentralized settings including verifiable delay functions [9,70], succinct blockchains [44], rollups [71,45,53], verifiable state machines [58], and proofs of machine executions (e.g., EVM, RISC-V)

This is an extended version of a paper from CRYPTO 2024 [42]. Compared to an initial version, this version of the paper incorporates material from prior preprints [40,41]. Additionally, this version provides an approach to achieve zero-knowledge in folding-scheme-based recursive arguments without needing to use zkSNARKs.

Early realizations of IVC [66,4] rely on succinct non-interactive arguments of knowledge (SNARKs) [37,50,32,6]. At step i, the prover produces a SNARK proving that it has correctly applied a step of the specified computation using the output of step $i-1$ and that the SNARK verifier *represented as a circuit* has accepted a SNARK from step $i-1$ [7,4]. These works require representing the SNARK verifier as a circuit. To reduce the size of the SNARK verifier when encoded as a circuit, prior work [4] uses a *two-cycle of elliptic curves*.[1]

A flurry of works [13,19,10,18,43,40] reduce reliance on SNARKs to construct IVC, culminating in *folding schemes* [43], a primitive that simply reduces the task of checking two NP instances with the same "structure" (e.g., circuit description) into the task of checking a single NP instance. This primitive is sufficient to construct IVC, and is simpler and far more efficient than a SNARK.

1.1 An overview of the prior state-of-the-art: Nova

We first focus on prior work at the time this paper was written. We discuss additional related work in Appendix C. Following a preprint of this work, there are several follow-up works. Section 1.4 describes subsequent work.

Nova's computational model proves incremental computations where each step executes a non-deterministic circuit. To prove such computations, Nova uses a folding scheme for an NP-complete language to (recursively) transform the task of proving N steps of a computation into the task of proving a single step of the computation. It then applies a general-purpose zkSNARK (e.g., Spartan [57]) to prove that single step, obtaining zero-knowledge and additional succinctness.

Compared to employing a general-purpose zkSNARK, built from from polynomial IOPs and polynomial commitment schemes (e.g., Spartan [57], Plonk [30], Marlin [21], HyperPlonk [20]), to prove the entire incremental computation, Nova's approach is substantially cheaper (as long as each step is sufficiently large, to offset recursion overheads). Specifically, at each incremental step, Nova's prover incurs only *two* MSMs of size proportional to the size of the circuit proven. Whereas, general-purpose zkSNARKs need many more MSMs. For example, Marlin [21, Figure 1] reports 22 MSMs and many more FFTs of size proportional to the circuit size. In addition, by design, Nova's proof generation is incremental (i.e., it produces a proof for each step and then uses its recursion capabilities to produce a single proof), so it can be more easily distributed and parallelized than with a non-recursive zkSNARK where one must unroll program executions into monolithic circuits. The latter rules out applications where one cannot statically unroll program executions (e.g., VDF) or makes it inconvenient (e.g., program executions on machines such as EVM or RISC-V). As presented, Nova does not immediately support parallel proof generation, but there exists a generic compiler [7] to transform constructions such as Nova to support parallel proving.

[1] A 2-cycle of elliptic curves is a pair of elliptic curves (E_1, E_2) such that the scalar field of E_1 equals the base field of E_2 (i.e., the field over which points in E_2 are defined over) and vice versa (Section 8 provides details on how a 2-cycle of elliptic curves is used and how they help with concrete efficiency).

1.2 Open problems addressed by this work

Our work addresses several open problems in the prior state-of-the-art. We first discuss these problems and provide sufficient context about each of these.

(1) The need to fold customizable, high-degree constraint systems. In Nova, each step of an incremental computation is expressed with R1CS, an NP-complete problem that generalizes arithmetic circuit satisfiability [31,59,3]. In practice, when zkSNARKs (e.g., Plonk) are applied to prove program executions, practitioners use custom constraint systems (e.g., Plonkish) that are tailored to a particular classes of applications. Specifically, Plonkish constraints are multivariate high-degree polynomials. Whereas, R1CS is restricted to checking quadratic constraints in a specific form. These customizable, high-degree constraint systems are often more compact than equivalent R1CS. As a concrete example, a single iteration of MinRoot [36] can be represented with *one* degree-5 constraint in Plonkish [74]. Whereas, R1CS needs three constraints.

Sangria [51] shows that Nova can be adapted to handle Plonkish. However, the number of cross-terms that the prover must commit to increases linearly with the degree of the constraints d: The prover must incur $O(n \cdot d)$ cryptographic operations to commit to $O(d)$ cross-terms, where n is the number of constraints. As a result, in general, there are not significant benefits to employing high-degree constraints and use Sangria than use Nova with R1CS. Concretely, Sangria's prover applied to MinRoot with degree-5 constraints requires 5 scalar multiplications (and additional field work to compute cross-terms) per MinRoot iteration whereas Nova applied to MinRoot in R1CS requires 6 scalar multiplications.

A key question is whether one can build a recursive argument for Plonkish, with Nova-like performance characteristics. In particular, our goal is to prove CCS [60], a customizable constraint system that simultaneously generalizes Plonkish, R1CS, and AIR without overheads ([60] provides context on CCS, Plonkish, and AIR). Additionally, for any solution that handles high-degree constraints, we require the prover's cryptographic work to be independent of the degree of constraints supported. That is, the number of MSMs (or their sizes) performed by the prover must not depend on the degree of the supported constraints.

(2) The need to achieve an "a la carte" cost profile for proving machine executions. A classic approach to prove machine executions (e.g., program executions on EVM) is to employ a *universal circuit* (e.g., [2,5,55,33,47]) that can execute any instruction supported by the machine. To prove the correct execution of programs on the corresponding machine, it suffices to recursively prove, with an IVC scheme, repeated invocations of this circuit on an input program and memory state [4]. Unfortunately, the cost of proving a program's step is proportional to the size of the universal circuit (i.e., sum of sizes of circuits of all instructions supported by the machine)—even though the step invokes only one of the instructions.

Given the high costs imposed by universal circuits, designers of these machines aim to employ a minimal instruction set, to keep the size of the universal circuit

and thereby the cost of proving a program step minimal [5,3,33]. However, this is a not a panacea: for real applications, one needs to execute an enormous number of iterations of the minimal circuit (e.g., billions of iterations), making the prover's work largely untenable. This also means that emulating real programs that target *existing* virtual machines with rich instruction sets (e.g., EVM, RISC-V, Wasm) via a machine with a minimal instruction set would incur enormous costs.

An open question is whether one can achieve an *"a la carte" cost profile*, where the cost of proving a step of a program execution is proportional only to the size of the circuit representing the instruction invoked by the program step and independent of the circuit sizes of the uninvoked instructions.

(3) The need for providing zero-knowledge without needing zkSNARKs. Nova [43] shows how to efficiently achieve zero-knowledge for its IVC proofs by producing a zkSNARK proving the knowledge of valid IVC proofs. The zkSNARK scheme that is natively compatible with Nova is Spartan [57], which internally uses the sum-check protocol [46]. The most efficient way to achieve zero-knowledge in Spartan is to use the Cramer-Damgard transformation [25,69], where sum-check messages are committed with homomorphic commitments (e.g., Pedersen) and the sum-check verifier's checks are proven in zero-knowledge using Schnorr-type proofs. This means that the Spartan verifier must perform public key operations (e.g., group scalar multiplications), which are far too expensive especially in blockchain settings where the verifier is deployed on-chain.

An open question is whether one can leverage folding schemes to "blind" the IVC proof such that one can use a non-zk Spartan, where the verifier verifies the sum-check messages in plaintext (which are orders of magnitude more efficient).

(4) The need for an efficient instantiation over a cycle of elliptic curves. Folding schemes leverage additively homomorphic commitments, which are typically instantiated with elliptic curve groups. To realize IVC, the folding scheme's verifier must be represented as a circuit. The best known approach for this is the blueprint of [4], which leverages a cycle of elliptic curves. Nova's implementation [1] adapts BCTV's approach [4] to the context of folding-scheme-based recursive arguments and was recently proven secure [52].

Unfortunately, Nova's approach, like in [4], still requires representing a verifier (which happens to be the the non-interactive folding scheme verifier) as a circuit on *both* curves in the cycle of curves. For Nova [43], which is the only *fully* implemented folding-scheme-based approach to date, the circuit defined over the second curve in the cycle is $\approx 10,000$ multiplication gates (and more than 100,000 non-zero entries in R1CS matrices).

In practice, one often wants to use a "half"-pairing cycle E_1/E_2[2] (e.g., BN254 and Grumpkin, where only BN254 is pairing-friendly). The BN254/Grumpkin cycle is as efficient as non-pairing-friendly cycle of curves (e.g., Pasta) and also compatible with Ethereum for proof verification. In this setting, the part of the

[2] A 2-cycle of elliptic curves where only one of the curves is pairing-friendly.

IVC proof defined over E_1 can be compressed easily into a succinct proof with Spartan [57] using KZG-based commitment scheme [72,12]; the compressed proof can be verified with a logarithmic number of group scalar multiplications and two pairings. However, for E_2, the corresponding IVC proof must be verified with a circuit $\mathcal{C}$ defined over E_1 and then proven with a SNARK defined over E_1. Unfortunately, $|\mathcal{C}| > 70 \cdot 10^6$ gates (far too expensive to prove).[3]

An open question is whether one can substantially reduce the size of the circuit defined over the second curve in the cycle, which in turn reduces the size of $\mathcal{C}$.

1.3 A technical overview of results in this work

This subsection provides an overview of HyperNova, which resolves all the four open problems listed in the prior section.

(1) Multi-folding schemes and a multi-folding scheme for CCS. As noted earlier, HyperNova's target is to prove incremental computations where each step of the incremental computation is expressed with CCS [60]. However, if we naively build a folding scheme for CCS, perhaps for a "relaxed" variant of CCS (analogous to relaxed R1CS in Nova [43]), it will have the efficiency issues noted above for Sangria. To avoid those issues, HyperNova takes a different approach that involves leveraging the power of the sum-check protocol [46].

To construct HyperNova, we introduce a generalization of folding schemes, and we refer to it as *multi-folding schemes*. Recall that a folding scheme for a relation $\mathcal{R}$ is a protocol between a *prover* and *verifier* in which the prover and the verifier reduce the task of checking two instances in $\mathcal{R}$ with the same structure s into the task of checking a single instance in $\mathcal{R}$ with structure s. A multi-folding scheme is defined with respect to a pair of relations $(\mathcal{R}_1, \mathcal{R}_2)$ and constants (μ, ν), and it is an interactive protocol in which the prover and the verifier reduce the task of checking μ instances in $\mathcal{R}_1$ with structure s_1 and ν instances in $\mathcal{R}_2$ with structure s_2 into the task of checking a single instance in $\mathcal{R}_1$ with structure s_1—as long as $(\mathsf{s}_1, \mathsf{s}_2)$ satisfy a pre-defined predicate (e.g., that the two structures are equal). Below, we clarify how this generalization unlocks additional power for constructing IVC.

We also construct a multi-folding scheme for CCS. Our starting point is the observation that Spartan [57] (more specifically its generalization to handle CCS called SuperSpartan [60]) transforms the task of checking the satisfiability of a CCS instance into the task of checking if a multivariate polynomial g of total degree $d + 1$, where d is the degree of the CCS constraints, sums to zero over a suitable Boolean hypercube. Spartan then invokes the sum-check protocol [46] to prove that claim about g. At the end of the sum-check invocation, the prover and the verifier are left with checking certain claims. Fortunately, these claims concern a restricted form of CCS (we formalize this and refer to it as *linearized*

[3] $\mathcal{C}$ proves openings of two vector commitments of 10,000 bases, which costs $\approx 2 \cdot 10,000 \cdot 3,000 = 60 \cdot 10^6$ gates. Also, $\mathcal{C}$ evaluates 10^5 linear combinations, which require field emulation and we estimate it to be $10^5 \times 100 = 10 \cdot 10^6$ gates. So, $|\mathcal{C}| > 70 \cdot 10^6$ gates.

CCS). Note that Spartan proves those claims about the restricted form of CCS with an additional invocation of the sum-check protocol, followed by evaluations proofs of committed sparse multilinear polynomials.

While an "early stopping" version of Spartan (the one with a single invocation of the sum-check protocol) provides a reduction of knowledge [39] from CCS to linearized CCS, it is not a folding (or a multi-folding) scheme. So, our second idea is to redefine the polynomial g to additionally include claims from a running linearized CCS instance using a random challenge from the verifier. This is possible as long as the running instance and the CCS instance that is being folded share a *compatible* structure (e.g., the same CCS matrices).

The following theorem summarizes our result about the multi-folding scheme. Notably, our multi-folding scheme avoids commitments to cross-terms *altogether*.

Theorem 1 (A multi-folding scheme for CCS). *Construction 1 is a public-coin, multi-folding scheme that reduces the task of checking an arbitrary number of CCS instances and linearized CCS instances with the same structure into the task of checking a single linearized CCS instance with the same structure. For a single CCS instance with m constraints of degree d and q monomials, n witness variables, t CCS matrices, and N non-zero entries in CCS matrices, and a single linearized CCS instance, the efficiency characteristics are as follows.*

- *The prover time is $O(N + t \cdot m + q \cdot m \cdot d \cdot \log^2 d)$ finite field operations and O(1) group operations;*
- *The verifier time is $O(d \cdot \log m)$ finite field operations and O(1) group operations; and*
- *The communication complexity is $O(d \cdot \log m)$ finite field elements.*

Since the multi-folding scheme is public coin, we make it non-interactive in the random oracle model using the Fiat-Shamir transform [28] and heuristically instantiate it in the plain model using a concrete cryptographic hash function.

(2) Achieving "a la carte" costs with non-uniform IVC. We first introduce a generalization of IVC [66] to formally capture an "a la carte" cost profile. Consider a collection of $\ell + 1$ non-deterministic, polynomial-time computable functions $((F_1, \ldots, F_\ell), \varphi)$, where $\ell \geq 1$. Suppose that each function F_j $(1 \leq j \leq \ell)$ takes s inputs and produces s outputs, where $s > 0$; F_j can additionally take an arbitrary non-deterministic input. Furthermore, φ is a function that takes s inputs and an arbitrary non-deterministic input, and produces an element of $\mathbb{Z}^*_{\ell+1}$ (i.e., the set $\{1, \ldots, \ell\}$).

A non-uniform IVC (NIVC) scheme enables a prover to incrementally prove that it has performed an n-step computation with an initial input z_0 to produce an output z_n. In particular, at step i, the prover proves that it has applied F_j on input (z_{i-1}, ω_{i-1}) to produce an output z_i, where z_{i-1} is output of step $i-1$, ω_{i-1} is a (potentially secret) non-deterministic input from the prover for step i, and

$j = \varphi(z_{i-1}, \omega_{i-1})$. That is, φ selects one of the possible ℓ functions to apply at step i using inputs to step i. A bit more concisely, for a specified $((F_1, \ldots, F_\ell), \varphi)$ and (n, z_0, z_n), the prover proves the knowledge of a set of non-deterministic values $(\omega_0, \ldots, \omega_{n-1})$ and $(z_1, \ldots, z_{n-1})$ such that for all $i \in \{0, \ldots, n-1\}$, we have that $z_{i+1} = F_{\varphi(z_i, \omega_i)}(z_i, \omega_i)$. Crucially, the prover's work at step i is proportional only to $|F_j|$, where $j = \varphi(z_i, \omega_i)$, rather than $|F_1| + \ldots + |F_\ell|$.

We then provide a generic compiler to construct a non-uniform IVC scheme from non-interactive multi-folding schemes such as the multi-folding scheme for CCS discussed above. The compiler requires the multi-folding scheme to satisfy certain requirements, which we formalize as *NIVC-compatibility*.

In more detail, suppose that the prover is provided with an NIVC proof π_i of i steps, which consists of a "fresh" instance u_i claiming the correct execution of step i, a collection of "running" instances (one for each function/instruction supported in NIVC) U_i claiming the correct execution of all prior $i-1$ steps, and the corresponding witnesses w_i and W_i. That is, $\pi_i = (\mathsf{U}_i, \mathsf{u}_i, \mathsf{W}_i, \mathsf{w}_i)$.

Then, the prover runs an augmented function, which, in addition to running a step of the incremental computation, runs a *verifier circuit*. The verifier circuit implements the verifier of the multi-folding scheme to fold u_i into an appropriate running instance in U_i to produce new running instances U_{i+1} that claims the correct execution of i steps. Alongside, the prover computes the corresponding folded witnesses W_{i+1}. The prover then produces a corresponding fresh instance u_{i+1} (and the corresponding witness w_{i+1}) that claims the correct execution of this augmented function; this fresh instance claims the correctness of the latest step of the incremental computation and that U_{i+1} was produced honestly. Together, $\pi_{i+1} = (\mathsf{U}_{i+1}, \mathsf{u}_{i+1}, \mathsf{W}_{i+1}, \mathsf{w}_{i+1})$ represents an NIVC proof of $i+1$ steps.

Remark 1. Because a multi-folding scheme folds an arbitrary number of running instances incoming instances into a single running instance, it affords a natural generalization of IVC [66] called proof-carrying data [7] using the approach of Bünz et al. [18]. We focus on IVC for its conceptual simplicity.

(3) Achieving zero-knowledge without zkSNARKs. To achieve a zero-knowledge argument of a valid NIVC proof without relying on zkSNARKs, we provide a new approach of *rerandomizing* the NIVC proof using folding schemes. In particular, given an NIVC proof $\pi_i = (\mathsf{U}_i, \mathsf{u}_i, \mathsf{W}_i, \mathsf{w}_i)$, the prover first folds the fresh instance u_i into an appropriate running instance. Next, the prover uses a folding scheme to fold in a random instance-witness pairs into $(\mathsf{U}_i, \mathsf{W}_i)$, effectively rerandomizing them. The prover then produces a randomized proof which consists of the rerandomized instance-witness pairs and the prover's messages in the folding scheme. A central challenge with this strategy is that the prover must prove that folding is done correctly without revealing the input random instances (and witnesses) used to randomize. To solve this, we have the prover execute the verifier's checks for the folding scheme inside a circuit and prove in zero-knowledge once again using a randomizing folding scheme.

The verifier at this point can directly check the randomized instances, or use a non-zero-knowledge SNARK if further succinctness is needed.

This approach highlights a new application of folding schemes. Until now, folding schemes were used to construct IVC [43]. Whereas, this result shows that folding schemes can be used to randomize folding-scheme-based IVC proofs (and NP instances!) prior to proving them with a non-zero-knowledge SNARK. Thus, we introduce a simpler, more efficient methodology (as opposed to that in [25]) for achieving zero-knowledge.

(4) Efficient instantiation over a two-cycle of elliptic curves. We provide a new approach, which we refer to as CycleFold, to efficiently instantiate HyperNova over a two-cycle of elliptic curves. In particular, we provide security proofs for this instantiation of HyperNova, but the approach and proofs generalize to other folding-scheme-based (N)IVC schemes.

CycleFold's starting point is the observation that folding-scheme-based recursive arguments can be efficiently instantiated *without* a cycle of elliptic curves—except for a few scalar multiplications in their verifiers (2 in Nova and 1 in HyperNova). Accordingly, CycleFold uses the second curve in the cycle to merely represent a *single* scalar multiplication (≈1,000 multiplication gates and ≈4,000 non-zero entries in R1CS matrices). CycleFold then folds invocations of this tiny circuit on the first curve in the cycle. This is more than an order of magnitude improvement over the prior state-of-the-art in terms of circuit sizes on the second curve. Furthermore, to achieve full succinctness for verification of proofs on a blockchain, $|\mathcal{C}|$ is similarly more than 10× smaller, and is now within the feasible range for proving with a SNARK defined over a pairing-friendly curve (§1.2).

Theorem 2 (HyperNova with CycleFold). *Given the multi-folding scheme for CCS (Construction 7) instantiated with the Pedersen commitment scheme, HyperNova (Construction 2) produces an NIVC scheme such that for step functions F_j for $j \in [\ell]$ that can be expressed in CCS with m_j constraints of degree d and q_j monomials, n_j witness variables, t_j CCS matrices, and N_j non-zero entries in CCS matrices, and control function φ that can be expressed in CCS with m constraints of degree d and q_φ monomials, n_φ witness variables, t_φ CCS matrices, and N_φ non-zero entries in the CCS matrices, the efficiency characteristics are as follows.*

- *The NIVC prover time for each step is a single MSM of size $O(n_\varphi + n_j)$ and $O((N_\varphi + N_j) + (t_\varphi + t_j) \cdot (m_\varphi + m_j) + (q_\varphi + q_j) \cdot (m_\varphi + m_j) \cdot d \cdot \log^2 d)$ finite field operations*
- *The verifier circuit size is $o(|\varphi| + 2 \cdot \mathsf{G} + (d \cdot \log m_j) \cdot \mathsf{F} + \log m_j \cdot \mathsf{R}_d + 2 \cdot \mathsf{H}_{\ell,t_j} + 2 \cdot \mathsf{M})$ on the first curve and G on the second curve in a cycle of elliptic curves.*

where G is the number of constraints required to encode a group scalar multiplication natively (i.e., without field emulation), H is the number of constraints required to encode a hash function, F is the number of constraints to encode field operations, R is the number of constraints to encode a cryptographic hash function

used for randomness, and M *is the number of constraints to encode to memory read/write over a memory of size* $O(\ell)$.

1.4 Subsequent works

We now discuss subsequent works that follow a preprint of this work.

(1) PCD. Zhou et al. [76] show that HyperNova naturally extends to provide a generalization of IVC called PCD [7,19,18].

(2) Protostar and ProtoGalaxy. Like HyperNova, Protostar [17] provides a folding scheme for high-degree constraints. Protostar achieves a similar prover efficiency as HyperNova. Although Protostar does not explicitly invoke the sum-check protocol, its folding procedure performs the same amount of commitment work and finite field operations, so it implicitly invokes a sum-check-like procedure.

- When folding two instances, Protostar's verifier circuit performs *three* group scalar multiplications whereas HyperNova does only *one*. On the other hand, HyperNova performs $O(d \cdot \log m)$ hashes where d is the degree of constraints and m is the number of constraints folded, and Protostar performs $O(d)$ hashes. When using SNARK-friendly hash functions (e.g., Poseidon), the hashing cost difference between HyperNova and Protostar is concretely small.
- HyperNova can fold $k > 2$ instances at once, which makes it easy to realize PCD [76]. On the other hand, Protostar folds only *two* instances at once. Extending it fold $k > 2$ instances at once blows up the degree of the polynomial involved exponentially in k [27, §1.2]. ProtoGalaxy [27] provides details of this, and avoids avoids this issue by essentially leveraging the sum-check protocol in a different way than in HyperNova. However, like HyperNova, it requires a logarithmic number of hashes in the verifier circuit [27, Table 1].
- Protostar does not describe how to instantiate it on a cycle of elliptic curves nor provide a zero-knowledge layer, whereas HyperNova includes both.
- Protostar describes how to integrate the logUp lookup argument [35] into IVC. One can easily integrate HyperNova with logUp as well as more recent lookup arguments (e.g., Lasso [61]). Since Lasso encodes lookups as sum-check instances, HyperNova can integrate with Lasso by including the lookup sum-check instances alongside HyperNova's sum-check instances for CCS.
- Protostar designs its folding scheme for special-sound protocols whereas HyperNova targets CCS. Both are equivalent as one can represent the verifier of the special-sound protocol as a CCS instance with standard transformations, for instance after making the special-sound protocol non-interactive with Fiat-Shamir transformation [65, §6]. Once that is done, a folding scheme for CCS can be applied. Note that Protostar too turns its special-sound protocols non-interactive protocols.

(3) KiloNova. KiloNova [75] extends HyperNova to achieve non-uniform PCD (a generalization of non-uniform IVC introduced in this paper). But, there are

fundamental problems. Its folding verifier's runtime is *linear* in the size of the NP instance (step-13 of Construction-1 folds CCS matrices), which makes it unsuitable for IVC, let alone PCD. In IVC, the folding-scheme verifier's runtime is $O(|F'|)$, where F' is an augmented circuit. This causes a well-known sizing issue because F' must include the folding verifier as a sub-circuit. KiloNova does separately suggest an optimization that may (inadvertently) resolve this issue. It is neither specified nor proven. Their scheme fundamentally relies on the ability to efficiently fold commitments to sparse CCS matrices, which is necessary for IVC/PCD. Unfortunately, the "natural" fix of committing to sparse CCS matrices such that they can be folded is an open problem: existing sparse polynomial commitments do *not* provide homomorphic commitments, or require (impractical) quadratic-sized parameters.

2 Preliminaries

We use λ to denote the security parameter and $\mathbb{F}$ to denote a finite field (e.g., the prime field $\mathbb{F}_p$ for a large prime p). We use $\mathsf{negl}(\lambda)$ to denote a negligible function in λ. We write $\Pr[X] \approx \epsilon$ to mean that $|\Pr[X] - \epsilon| = \mathsf{negl}(\lambda)$. Throughout the paper, the depicted asymptotics depend on λ, but we elide this for brevity. We write PPT to refer to probabilistic polynomial time algorithms. For relations $\mathcal{R}_1$ and $\mathcal{R}_2$ we let $\mathcal{R}_1 \times \mathcal{R}_2$ denote a new relation such that $((u_1, u_2), (w_1, w_2)) \in \mathcal{R}_1 \times \mathcal{R}_2$ if and only if $(u_1, w_1) \in \mathcal{R}_1$ and $(u_2, w_2) \in \mathcal{R}_2$. We write $\mathbb{F}^d[X_1, \ldots, X_n]$ to denote multivariate polynomials over field $\mathbb{F}$ in the variables $X_1, \ldots, X_n$ with degree bound d for each variable. We omit the superscript if there is no bound.

Appendix A provides additional preliminaries on multilinear polynomials, the sum-check protocol, commitment schemes, arguments of knowledge, and IVC.

Customizable constraint systems (CCS). CCS simultaneously generalizes R1CS, Plonkish, and AIR without overheads. We first provide an arithmetized variant of the original formulation. The definitions below are characterized by a finite field $\mathbb{F}$, but we leave this implicit.

Definition 1 (CCS [60]). *Consider size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ where $n > \ell$. Let $s = \log m$ and $s' = \log n$. We define the customizable constraint system (CCS) relation, $\mathcal{R}_{\mathsf{CCS}}$, over structure, instance, witness tuples as follows.*

An $\mathcal{R}_{\mathsf{CCS}}$ structure s consists of

- *a sequence of sparse multilinear polynomials in $s + s'$ variables $\widetilde{M}_1, \ldots, \widetilde{M}_t$ such that they evaluate to a non-zero value in at most $N = \Omega(m)$ locations over the Boolean hypercube $\{0,1\}^s \times \{0,1\}^{s'}$;*
- *a sequence of q multisets $[S_1, \ldots, S_q]$, where an element in each multiset is from the domain $\{1, \ldots, t\}$ and the cardinality of each multiset is at most d.*
- *a sequence of q constants $[c_1, \ldots, c_q]$, where each constant is from $\mathbb{F}$.*

An $\mathcal{R}_{\mathsf{CCS}}$ instance consists of public input and output vector $\mathsf{x} \in \mathbb{F}^\ell$. An $\mathcal{R}_{\mathsf{CCS}}$ witness consists of a multilinear polynomial $\widetilde{w}$ in $s' - 1$ variables. We have that

$(\mathsf{s}, \mathsf{x}, \widetilde{w}) \in \mathcal{R}_{\mathsf{CCS}}$ *if and only if for all* $x \in \{0,1\}^s$*,*

$$\sum_{i=1}^{q} c_i \cdot \left(\prod\nolimits_{j \in S_i} \left(\sum_{y \in \{0,1\}^{\log m}} \widetilde{M}_j(x, y) \cdot \widetilde{z}(y) \right) \right) = 0,$$

where $\widetilde{z}$ *is an* s'*-variate multilinear polynomial such that* $\widetilde{z}(x) = \widetilde{(w, 1, \mathsf{x})}(x)$ *for all* $x \in \{0,1\}^{s'}$*.*

In this work, we introduce *linearized CCS*, a variant of CCS that only contains the linear checks of CCS. We later show that we can fold a CCS instance into a linearized CCS instance to produce a new linearized CCS instance.

Definition 2 (Linearized CCS). *Consider size bounds* $m, n, N, \ell, t, q, d \in \mathbb{N}$ *where* $n = 2 \cdot (\ell + 1)$*. Let* $s = \log m$ *and* $s' = \log n$*. We define the linearized committed customizable constraint system (LCCS) relation,* $\mathcal{R}_{\mathsf{LCCS}}$*, over structure, instance, witness tuples as follows.*

An $\mathcal{R}_{\mathsf{LCCS}}$ *structure* s *consists of*

- *a sequence of sparse multilinear polynomials in* $s + s'$ *variables* $\widetilde{M}_1, \ldots, \widetilde{M}_t$ *such that they evaluate to a non-zero value in at most* $N = \Omega(m)$ *locations over the Boolean hypercube* $\{0,1\}^s \times \{0,1\}^{s'}$*;*
- *a sequence of* q *multisets* $[S_1, \ldots, S_q]$*, where an element in each multiset is from the domain* $\{1, \ldots, t\}$ *and the cardinality of each multiset is at most* d*.*
- *a sequence of* q *constants* $[c_1, \ldots, c_q]$*, where each constant is from* $\mathbb{F}$*.*

An $\mathcal{R}_{\mathsf{LCCS}}$ *instance is a tuple* $(u, \mathsf{x}, r, v_1, \ldots, v_t) \in (\mathbb{F}, \mathbb{F}^\ell, \mathbb{F}, \mathbb{F}^t)$*. An* $\mathcal{R}_{\mathsf{LCCS}}$ *witness consists of a multilinear polynomial* $\widetilde{w}$ *in* $s' - 1$ *variables. We have that* $(\mathsf{s}, (u, \mathsf{x}, r, v_1, \ldots, v_t), \widetilde{w}) \in \mathcal{R}_{\mathsf{LCCS}}$ *if and only if for all* $i \in [t]$

$$v_i = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r, y) \cdot \widetilde{z}(y)$$

where $\widetilde{z}$ *is an* s'*-variate multilinear polynomial such that* $z(x) = \widetilde{(w, u, \mathsf{x})}(x)$ *for all* $x \in \{0,1\}^{s'}$*.*

R1CS is an NP-complete problem implicit in QAPs [31]. For completeness, we formally define R1CS in Appendix A.4. Below, we recall its folding-friendly variant, relaxed R1CS [43]. We utilize relaxed R1CS for our zero-knowledge layer and our instantiation of HyperNova over a cycle of curves.

Definition 3 (Relaxed R1CS). *Consider a finite field* $\mathbb{F}$ *and a commitment scheme* Commit *over vectors over* $\mathbb{F}$*. Consider size bounds* $m, n, \ell \in \mathbb{N}$ *where*

$m > \ell$. We define the relaxed R1CS relation, $\mathcal{R}_{\mathsf{RR1CS}}$, over structure, instance, witness tuples as follows.[4]

A $\mathcal{R}_{\mathsf{RR1CS}}$ structure consists of matrices $A, B, C \in \mathbb{F}^{m \times m}$ with at most $n = \Omega(m)$ non-zero entries in each matrix. A $\mathcal{R}_{\mathsf{RR1CS}}$ instance is a tuple $(u, \mathsf{x}) \in (\mathbb{F}, \mathbb{F}^{\ell})$. A $\mathcal{R}_{\mathsf{RR1CS}}$ witness is a tuple $(E, W) \in (\mathbb{F}^{m}, \mathbb{F}^{m-\ell-1})$. We have that $((A, B, C), (u, \mathsf{x}), (E, W)) \in \mathcal{R}_{\mathsf{RR1CS}}$ iff for $Z = (W, \mathsf{x}, u)$, $AZ \circ BZ = u \cdot CZ + E$.

Instead of directly working with all of the above relations, we consider variants where a commitment to the witness is additionally presented in the instance. We generically refer to such relations as *committed relations*.

Definition 4 (Committed relation). *Consider a relation $\mathcal{R}$ over structure, instance, witness tuples where witnesses are in some space W. Consider a commitment scheme $\mathsf{com} = (\mathsf{Gen}, \mathsf{Commit})$ over message space W. We define the corresponding committed relation over public parameter, structure, instance, witness tuples characterized by com as follows.*

$$\mathcal{R}(\mathsf{com}) = \left\{ (\mathsf{pp}_{\mathsf{com}}, \mathsf{s}, (C, u), (w, r)) \;\middle|\; \begin{array}{l} (\mathsf{s}, u, w) \in \mathcal{R}, \\ C = \mathsf{Commit}(\mathsf{pp}_{\mathsf{com}}, w, r) \end{array} \right\}$$

We say relation $\mathcal{R}$ is the underlying relation for committed relation $\mathcal{R}(\mathsf{com})$.

Definition 5 ((Linearized) Committed CCS). *Consider an additively homomorphic polynomial commitment scheme, PC, for multilinear polynomials over a finite field $\mathbb{F}$. We define the committed CCS relation $\mathcal{R}_{\mathsf{CCCS}}$ as $\mathcal{R}_{\mathsf{CCS}}(\mathsf{PC})$ and the linearized committed CCS relation $\mathcal{R}_{\mathsf{LCCCS}}$ as $\mathcal{R}_{\mathsf{LCCS}}(\mathsf{PC})$.*

Definition 6 (Committed relaxed R1CS). *Consider a commitment scheme VC over vectors over field $\mathbb{F}$. We define the committed relaxed R1CS relation $\mathcal{R}_{\mathsf{CRR1CS}}$ as $\mathcal{R}_{\mathsf{RR1CS}}(\mathsf{VC}')$ where VC' commits to pairs of vectors by applying VC to each vector.*

3 Multi-folding schemes

Recall that a folding scheme [43] for a relation $\mathcal{R}$ is a protocol between a *prover* and *verifier* in which the prover and the verifier reduce the task of checking two instances in $\mathcal{R}$ with the same structure s into the task of checking a single instance in $\mathcal{R}$ with structure s.

We introduce a generalization of folding schemes, which we refer to as *multi-folding schemes*. A multi-folding scheme is defined with respect to a pair of relations $(\mathcal{R}_1, \mathcal{R}_2)$, a predicate compat, and size parameters μ and ν. It is an interactive protocol between a prover and a verifier in which the prover and the

[4] As formulated, any relaxed R1CS is naturally satisfiable by setting E appropriately. As shown by Kothapalli et al. [43] relaxed R1CS is augmented with honestly generated commitments to E, which sufficiently restricts the prover's choice of E.

verifier reduce the task of checking μ instances in $\mathcal{R}_1$ with structure s_1 and ν instances in $\mathcal{R}_2$ with structure s_2 into the task of checking a single instance in $\mathcal{R}_1$ with structure s_1—as long as s_1 and s_2 satisfy a predicate compat (e.g., compat might require that $\mathsf{s}_1 = \mathsf{s}_2$). Below, we formally define multi-folding schemes.

Definition 7 (Multi-folding schemes). *Consider relations $\mathcal{R}_1$ and $\mathcal{R}_2$ over public parameters, structure, instance, and witness tuples, a predicate compat that structures for instances in $\mathcal{R}_1$ and $\mathcal{R}_2$ must satisfy, and size parameters $\mu, \nu \in \mathbb{N}$. A multi-folding scheme for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu, \nu)$ is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic $\mathcal{K}$ denoted the generator, prover, verifier and encoder respectively with the following interface:*

- $\mathcal{G}(1^\lambda, N) \to \mathsf{pp}$*: on input security parameter λ and size bounds N, samples public parameters pp.*
- $\mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)) \to (\mathsf{pk}, \mathsf{vk})$*: on input pp, and structures s_1 and s_2 among the instances to be folded, outputs a prover key pk and a verifier key vk.*
- $\mathcal{P}(\mathsf{pk}, (\vec{\mathsf{u}}_1, \vec{\mathsf{w}}_1), (\vec{\mathsf{u}}_2, \vec{\mathsf{w}}_2)) \to (\mathsf{u}, \mathsf{w})$*: on input a vector of instances $\vec{\mathsf{u}}_1$ in $\mathcal{R}_1$ of size μ with structure s_1 and a vector of instances $\vec{\mathsf{u}}_2$ in $\mathcal{R}_2$ of size ν with structure s_2, and corresponding witness vectors $\vec{\mathsf{w}}_1$ and $\vec{\mathsf{w}}_2$ outputs a folded instance-witness pair (u, w) in $\mathcal{R}_1$ with structure s_1.*
- $\mathcal{V}(\mathsf{vk}, (\vec{\mathsf{u}}_1, \vec{\mathsf{u}}_2)) \to \mathsf{u}$*: on input a vector of instances $\vec{\mathsf{u}}_1$ and a vector of instances $\vec{\mathsf{u}}_2$ outputs a new instance u.*

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ denote the interaction between $\mathcal{P}$ and $\mathcal{V}$. We treat $\langle \mathcal{P}, \mathcal{V} \rangle$ as a function that takes as input $((\mathsf{pk}, \mathsf{vk}), (\vec{\mathsf{u}}_1, \vec{\mathsf{w}}_1), (\vec{\mathsf{u}}_2, \vec{\mathsf{w}}_2))$ and runs the interaction on prover input $(\mathsf{pk}, (\vec{\mathsf{u}}_1, \vec{\mathsf{w}}_1), (\vec{\mathsf{u}}_2, \vec{\mathsf{w}}_2))$ and verifier input $(\mathsf{vk}, (\vec{\mathsf{u}}_1, \vec{\mathsf{u}}_2))$. At the end of interaction $\langle \mathcal{P}, \mathcal{V} \rangle$ outputs (u, w) where u is the verifier's output folded instance, and w is the prover's output folded witness.

Let $\mathcal{R}^{(n)}$ be the relation such that $(\mathsf{pp}, \mathsf{s}, \vec{\mathsf{u}}, \vec{\mathsf{w}}) \in \mathcal{R}^{(n)}$ if and only if $(\mathsf{pp}, \mathsf{s}, \vec{\mathsf{u}}_i, \vec{\mathsf{w}}_i) \in \mathcal{R}$ for all $i \in [n]$. A multi-folding scheme for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu, \nu)$ satisfies the following requirements.

1. *Perfect Completeness: For all PPT adversaries $\mathcal{A}$, we have that*

$$\Pr\left[(\mathsf{pp}, \mathsf{s}_1, \mathsf{u}, \mathsf{w}) \in \mathcal{R}_1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N), \\ ((\mathsf{s}_1, \mathsf{s}_2), (\vec{\mathsf{u}}_1, \vec{\mathsf{u}}_2), (\vec{\mathsf{w}}_1, \vec{\mathsf{w}}_2)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ \mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = \mathsf{true}, \\ (\mathsf{pp}, \mathsf{s}_1, \vec{\mathsf{u}}_1, \vec{\mathsf{w}}_1) \in \mathcal{R}_1^{(\mu)}, (\mathsf{pp}, \mathsf{s}_2, \vec{\mathsf{u}}_2, \vec{\mathsf{w}}_2) \in \mathcal{R}_2^{(\nu)}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}_1, \mathsf{s}_2), \\ (u, w) \leftarrow \langle \mathcal{P}, \mathcal{V} \rangle ((\mathsf{pk}, \mathsf{vk}), (\vec{\mathsf{u}}_1, \vec{\mathsf{u}}_2), (\vec{\mathsf{w}}_1, \vec{\mathsf{w}}_2)) \end{array}\right] = 1.$$

2. *Knowledge Soundness: For any expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$ there is an expected polynomial-time extractor $\mathcal{E}$ such that*

$$\Pr_{\mathsf{r}}\left[\begin{array}{l|l}(\mathsf{pp},\mathsf{s}_1,\vec{\mathsf{u}}_1,\vec{\mathsf{w}}_1)\in\mathcal{R}_1^{(\mu)}, & \mathsf{pp}\leftarrow\mathcal{G}(1^\lambda,N),\\ (\mathsf{pp},\mathsf{s}_2,\vec{\mathsf{u}}_2,\vec{\mathsf{w}}_2)\in\mathcal{R}_2^{(\nu)} & ((\mathsf{s}_1,\mathsf{s}_2),(\vec{\mathsf{u}}_1,\vec{\mathsf{u}}_2),\mathsf{st})\leftarrow\mathcal{A}(\mathsf{pp},\mathsf{r}),\\ & \mathsf{compat}(\mathsf{s}_1,\mathsf{s}_2)=\mathsf{true},\\ & (\vec{\mathsf{w}}_1,\vec{\mathsf{w}}_2)\leftarrow\mathcal{E}(\mathsf{pp},\mathsf{r})\end{array}\right]\approx$$
$$\Pr_{\mathsf{r}}\left[\begin{array}{l|l}(\mathsf{pp},\mathsf{s}_1,u,w)\in\mathcal{R}_1 & \mathsf{pp}\leftarrow\mathcal{G}(1^\lambda,N),\\ & ((\mathsf{s}_1,\mathsf{s}_2),(\vec{\mathsf{u}}_1,\vec{\mathsf{u}}_2),\mathsf{st})\leftarrow\mathcal{A}(\mathsf{pp},\mathsf{r}),\\ & \mathsf{compat}(\mathsf{s}_1,\mathsf{s}_2)=\mathsf{true},\\ & (\mathsf{pk},\mathsf{vk})\leftarrow\mathcal{K}(\mathsf{pp},(\mathsf{s}_1,\mathsf{s}_2)),\\ & (u,w)\leftarrow\langle\mathcal{P}^*,\mathcal{V}\rangle((\mathsf{pk},\mathsf{vk}),(\vec{\mathsf{u}}_1,\vec{\mathsf{u}}_2),\mathsf{st})\end{array}\right]$$

where r *denotes an arbitrarily long random tape.*

A multi-folding scheme is secure in the random oracle model if the above requirements hold when all parties are provided access to a random oracle.

Definition 8 (Succinct). *A multi-folding scheme is succinct if the communication complexity and verifier time complexity is at most poly-logarithmic in the size of the structures and witnesses.*

Definition 9 (Non-interactive). *A multi-folding scheme $(\mathcal{G},\mathcal{K},\mathcal{P},\mathcal{V})$ is non-interactive if the interaction between $\mathcal{P}$ and $\mathcal{V}$ consists of a single message from $\mathcal{P}$ to $\mathcal{V}$. This single message is denoted as $\mathcal{P}$'s output and as $\mathcal{V}$'s input.*

Definition 10 (Public-coin). *A multi-folding scheme $(\mathcal{G},\mathcal{K},\mathcal{P},\mathcal{V})$ is called public-coin if all the messages sent from $\mathcal{V}$ to $\mathcal{P}$ are sampled uniformly.*

By applying the Fiat-Shamir transformation [28] we can transform a public-coin multi-folding scheme into a non-interactive multi-folding scheme in the random oracle model. We formally describe this transformation in Appendix B.

Lemma 1 (Fiat-Shamir transformation for multi-folding schemes). *Construction 3 transforms a public-coin multi-folding scheme for*

$$(\mathcal{R}_1,\mathcal{R}_2,\mathsf{compat},\mu,\nu)$$

into a non-interactive multi-folding scheme for $(\mathcal{R}_1,\mathcal{R}_2,\mathsf{compat},\mu,\nu)$ in the random oracle model.

4 A multi-folding scheme for CCS

This section describes a multi-folding scheme for CCS. Specifically, we provide a multi-folding scheme for $\mathcal{R}_1=\mathcal{R}_{\mathsf{LCCCS}}$ and $\mathcal{R}_2=\mathcal{R}_{\mathsf{CCCS}}$, with $\mathsf{compat}(\mathsf{s}_1,\mathsf{s}_2)$ requiring $\mathsf{s}_1=\mathsf{s}_2$. Our multi-folding scheme supports arbitrary values of μ and ν.

Overview. To introduce core ideas, we focus on the case of $\mu = \nu = 1$. Construction 1 formally describes the general case.

Consider structure $\mathsf{s}_1 = \mathsf{s}_2 = ([\widetilde{M}_1, \ldots, \widetilde{M}_t], [S_1, \ldots, S_q], [c_1, \ldots, c_q])$, and let $s = \log m$, and $s' = \log n$. We design a multi-folding scheme that reduces the verifier's task of checking a linearized committed CCS instance $(C_1, u, \mathsf{x}_1, r_x, v_1, \ldots, v_t)$ and a committed CCS instance (C_2, x_2) to the task of checking a new linearized committed CCS instance. In particular, the verifier's goal is to reduce the task of checking that a prover knows satisfying witnesses $\widetilde{w_1}$ and $\widetilde{w_2}$ such that for $\widetilde{z_1} = (\widetilde{w_1, u, \mathsf{x}_1})$ and $\widetilde{z_2} = (\widetilde{w_2, 1, \mathsf{x}_2})$ we have that

$$v_j = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r_x, y) \cdot \widetilde{z_1}(y) \tag{1}$$

for all $j \in [t]$ and

$$\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s}} \widetilde{M}_j(x, y) \cdot \widetilde{z_2}(y) \right) = 0 \tag{2}$$

for all $x \in \{0,1\}^s$.

The high-level strategy of the prover and verifier is to first encode the above claims as a claim about the evaluations of polynomials and then reduce this claim using the sum-check protocol. The resulting reduced claim is equivalent to checking two *compatible* linearized committed CCS instances. The compatibility ensures that we can reduce the task of checking both instances into the task of checking a single linearized CCS instance using a random linear combination.

In more detail, consider polynomials

$$H_j(x) \coloneqq \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_1(y) \tag{3}$$

and

$$G(x) \coloneqq \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_2(y) \right). \tag{4}$$

Then, checking $H_j(r_x) = v_j$ for all $j \in [t]$ implies checking Equation 1. Then, by Lemma 6, for $L_j(x) = \widetilde{eq}(r_x, x) \cdot H_j(x)$, this is equivalent to checking

$$v_j = \sum_{x \in \{0,1\}^{s}} L_j(x) \tag{5}$$

for all $j \in [t]$.

Similarly, checking $G(x) = 0$ for all $x \in \{0,1\}^s$ implies checking Equation 2. We define a corresponding Lagrange polynomial, $\sum_{x \in \{0,1\}^s} \widetilde{eq}(X, x) \cdot G(x)$, which

encodes each evaluation of G into its coefficients. Then checking that this Lagrange polynomial is the zero polynomial implies checking that $G(x) = 0$ for all $x \in \{0,1\}^s$. Then, for a random challenge $\beta \in \mathbb{F}$, by the Schwartz-Zippel lemma (Lemma 7), for $Q(x) = \widetilde{eq}(\beta, x) \cdot G(x)$, checking

$$0 = \sum_{x \in \{0,1\}^s} Q(x) \tag{6}$$

implies checking Equation 2 with high probability.

Equations 5 and 6 can be checked simultaneously with high probability by setting

$$g(x) \coloneqq \left(\sum_{j \in [t]} \gamma^j \cdot L_j(x) \right) + \gamma^{t+1} \cdot Q(x)$$

$$T \coloneqq \left(\sum_{j \in T} \gamma^j \cdot v_j \right) + \gamma^{t+1} \cdot 0$$

for some random challenge $\gamma \in \mathbb{F}$ and checking if

$$T = \sum_{x \in \{0,1\}^s} g(x). \tag{7}$$

Then, the prover and verifier run the sum-check protocol to reduce the task of checking Equation 7 to the task of checking

$$c = g(r'_x) \tag{8}$$

for some random point $r'_x \in \mathbb{F}^s$ chosen over the course of the sum-check protocol and a claimed evaluation $c \in \mathbb{F}$.

To assist the verifier in checking Equation 8, the prover computes claimed values for sums internal to polynomial g,

$$\sigma_i \leftarrow \sum_{y \in \{0,1\}^{s'}} \widetilde{M_i}(r'_x, y) \cdot \widetilde{z}_1(y) \tag{9}$$

$$\theta_i \leftarrow \sum_{y \in \{0,1\}^{s'}} \widetilde{M_i}(r'_x, y) \cdot \widetilde{z}_2(y), \tag{10}$$

for all $i \in [t]$, and sends them to the verifier.

Using these values, the verifier can check Equation 8. However, it must still check Equations 9 and 10, that is, that σ_i and θ_i were computed correctly for all $i \in [t]$.

We observe that because both of these equations are defined with respect to the same randomness r'_x. So, by linearity, the verifier can sample a random challenge ρ, and reduce the task of checking Equations 9 and 10 to the task of checking

$$\sigma_i + \rho \cdot \theta_i = \sum_{y \in \{0,1\}^{s'}} \widetilde{M_i}(r'_x, y) \cdot (\widetilde{z}_1(y) + \rho \cdot \widetilde{z}_2(y)) \tag{11}$$

for all $i \in [t]$.

Conveniently, letting $C' \leftarrow C_1 + \rho \cdot C_2$, $u' \leftarrow u + \rho \cdot 1$, $\mathsf{x}' \leftarrow \mathsf{x}_1 + \rho \cdot \mathsf{x}_2$, and $v'_i \leftarrow \sigma_i + \rho \cdot \theta_i$ for all $i \in [t]$, checking Equation 11, is equivalent to checking that the prover knows a witness for the following linearized committed CCS instance

$$(C', u', \mathsf{x}', r'_x, v'_1, \ldots, v'_t),$$

thus completing the reduction. We formally describe our folding scheme below.

Construction 1 (A multi-folding scheme for CCS). Let $\mathsf{PC} = (\mathsf{Gen}, \mathsf{Commit})$ denote an additively-homomorphic commitment scheme for multilinear polynomials. We construct a multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu, \nu)$, where compat is defined as follows.

$\underline{\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) \to \{\mathsf{true}, \mathsf{false}\}}$: If $\mathsf{s}_1.\widetilde{M}_i = \mathsf{s}_2.\widetilde{M}_i$ for $i \in [t]$, then return true, otherwise return false.

Let $\mathsf{s}_1 = \mathsf{s}_2 = ([\widetilde{M}_1, \ldots, \widetilde{M}_t], [S_1, \ldots, S_q], [c_1, \ldots, c_q])$.

We define the generator and the encoder as follows.

$\underline{\mathcal{G}(1^\lambda, (m, N, \ell, t, q, d \in \mathbb{N})) \to \mathsf{pp}}$:

1. Let $n = 2 \cdot (\ell + 1)$
2. $\mathsf{pp}_{\mathsf{PC}} \leftarrow \mathsf{Gen}(1^\lambda, \log n - 1)$
3. Output $(m, n, N, \ell, t, q, d, \mathsf{pp}_{\mathsf{PC}})$

$\underline{\mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)) \to (\mathsf{pk}, \mathsf{vk})}$:

1. Let $\mathsf{pk} \leftarrow (\mathsf{pp}, \mathsf{s}_1)$ and $\mathsf{vk} \leftarrow \mathsf{pp}$
2. Output $(\mathsf{pk}, \mathsf{vk})$

The verifier $\mathcal{V}$ takes μ linearized committed CCS instances $\vec{\mathsf{u}}_1$ and ν committed CCS instances $\vec{\mathsf{u}}_2$. The prover in addition to these instances takes witnesses to all instances $\vec{\mathsf{w}}_1$ and $\vec{\mathsf{w}}_2$. We denote μ linearized committed CCS instance-witness pairs with $\mathcal{L}$ and use $\mathcal{L}_k$ (for $k \in [\mu]$) to index into the kth linearized committed CCS instance-witness pair. Similarly, we denote ν committed CCS instance-witness pairs $\mathcal{C}$ and use $\mathcal{C}_k$ (for $k \in [\nu]$) to index into the kth committed CCS instance-witness pair. Inside an instance-witness pair, we use ϕ to index into the instance and w to index into the witness.

Let $s = \log m$ and $s' = \log n$. Let $\widetilde{z}_{1,k} = \widetilde{(w, u, \mathsf{x})}$, where $w = \mathcal{L}_k.w$, $u = \mathcal{L}_k.\phi.u$, and $\mathsf{x} = \mathcal{L}_k.\phi.\mathsf{x}$. Similarly, let $\widetilde{z}_{2,k} = \widetilde{(w, 1, \mathsf{x})}$, where $w = \mathcal{C}_k.w$ and $\mathsf{x} = \mathcal{C}_k.\phi.\mathsf{x}$.

The prover and the verifier proceed as follows.

1. $\mathcal{V} \to \mathcal{P}$: $\mathcal{V}$ samples $\gamma \xleftarrow{\$} \mathbb{F}, \beta \xleftarrow{\$} \mathbb{F}^s$, and sends them to $\mathcal{P}$.
2. $\mathcal{V}$: Sample $r'_x \xleftarrow{\$} \mathbb{F}^s$.

3. $\mathcal{V} \leftrightarrow \mathcal{P}$: Run the sum-check protocol $c \leftarrow \langle \mathcal{P}, \mathcal{V}(r'_x) \rangle (g, s, d+1, T)$, where:

$$
\begin{aligned}
g(x) &\coloneqq \left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot L_{j,k}(x) \right) + \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot Q_k(x) \right) \\
L_{j,k}(x) &\coloneqq \widetilde{eq}(r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(x, y) \cdot \widetilde{z}_{1,k}(y) \right) \\
Q_k(x) &\coloneqq \widetilde{eq}(\beta, x) \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(x, y) \cdot \widetilde{z}_{2,k}(y) \right) \right) \\
T &\coloneqq \sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot \mathcal{L}_k.\phi.v_j
\end{aligned}
$$

4. $\mathcal{P} \rightarrow \mathcal{V}$: $\{\sigma_{j,k}\}$, where for all $j \in [t]$, $k \in [\mu]$:

$$\sigma_{j,k} = \sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z}_{1,k}(y)$$

   Similarly, $\{\theta_{j,k}\}$, where for all $j \in [t]$ and $k \in [\nu]$:

$$\theta_{j,k} = \sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z}_{2,k}(y)$$

5. $\mathcal{V}$: Compute $e_1 \leftarrow \widetilde{eq}(r_x, r'_x)$ and $e_2 \leftarrow \widetilde{eq}(\beta, r'_x)$, and check that

$$c = \left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot e_1 \cdot \sigma_{j,k} \right) + \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot e_2 \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \theta_{j,k} \right) \right)$$

6. $\mathcal{V} \rightarrow \mathcal{P}$: $\mathcal{V}$ samples $\rho \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{P}$.
7. $\mathcal{V}, \mathcal{P}$: Output the folded linearized committed CCS instance $(C, u, \mathsf{x}, r'_x, v_1, \ldots, v_t)$, where for all $j \in [t]$:

$$
\begin{aligned}
C &\leftarrow \textstyle\sum_{k \in [\mu]} \rho^k \cdot \mathcal{L}_k.\phi.C + \sum_{k \in [\nu]} \rho^{\mu+k} \cdot \mathcal{C}_k.\phi.C \\
u &\leftarrow \textstyle\sum_{k \in [\mu]} \rho^k \cdot \mathcal{L}_k.\phi.u + \sum_{k \in [\nu]} \rho^{\mu+k} \cdot 1 \\
\mathsf{x} &\leftarrow \textstyle\sum_{k \in [\mu]} \rho^k \cdot \mathcal{L}_k.\phi.\mathsf{x} + \sum_{k \in [\nu]} \rho^{\mu+k} \cdot \mathcal{C}_k.\phi.\mathsf{x} \\
v_j &\leftarrow \textstyle\sum_{k \in [\mu]} \rho^k \cdot \sigma_{j,k} + \sum_{k \in [\nu]} \rho^{\mu+k} \cdot \theta_{j,k}
\end{aligned}
$$

8. $\mathcal{P}$: Output the folded witness $\widetilde{w} \leftarrow \sum_{k \in [\mu]} \rho^k \cdot \mathcal{L}_k.w + \sum_{k \in [\nu]} \rho^{\mu+k} \cdot \mathcal{C}_k.w$.

Below, we adapt the proof of Kothapalli et al. [43] to prove the correctness of our multifolding scheme for CCS (Theorem 1).

Proof (intuition). We provide a formal proof in Appendix H.1. Our multi-folding scheme is an "early stopping" version of SuperSpartan [60] and the claimed efficiency follows from the analysis of costs for the first sum-check invocation in SuperSpartan [60, Theorem 1]. To prove knowledge soundness, we show there exists an expected polynomial-time extractor that can rewind the interaction between a verifier and a malicious prover to interpolate for witnesses $\vec{\mathsf{w}}_1$ and $\vec{\mathsf{w}}_2$. So long as the verifier does not abort, we have that $g(r'_x) = c$. Then, by the soundness of the sum-check protocol, we have that $\sum_{j\in[t],k\in[\mu]} \gamma^{(k-1)\cdot t+j} \cdot v_{j,k} + \sum_{k\in[\nu]} \gamma^{\mu\cdot t+k} \cdot 0 = \sum_{x\in\{0,1\}^s} g(x)$. By the Schwartz-Zippel lemma, we have that $v_{j,k} = \sum_{x\in\{0,1\}^s} L_{j,k}(x)$ for all $j \in [t]$ and $k \in [\mu]$ and $0 = \sum_{x\in\{0,1\}^s} Q_k(x)$ for $k \in [\nu]$. This in turn implies that $\widetilde{w_1}$ and $\widetilde{w_2}$ are satisfying. □

Lemma 2 (Efficiency). *Construction 1 is succinct.*

Proof. In Step 1 and Step 2, the verifier begins by sampling and sending random challenges, which takes work $O(\log m)$ work, where m is the number of CCS constraints. Next, in Step 3, the verifier verifies sum-check messages which requires $O(d \log m)$ work, where d is the degree of CCS constraints. In Step 5, the verifier computes $\widetilde{eq}(r_x, r'_x)$ and $\widetilde{eq}(\beta, r'_x)$, which requires $O(\log m)$ field operations, and performs $O(t \cdot \mu + \nu)$ field operations where t, μ, and ν are constants. Finally, in Step 7, the verifier computes $O(\mu + \nu)$ group scalar multiplication operations and $O((\mu + \nu) \cdot |\mathsf{x}|)$ field operations. Combining all these, the verifier's work and the space requirements are logarithmic in the number of constraints and linear in the degree of CCS constraints. Hence, the verifier's work is succinct. □

By applying the Fiat-Shamir transformation (Construction 3), and instantiating the random oracle with a hash function, we have the following.

Assumption 1 (Non-interactivity). There exists a non-interactive multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu, \nu)$ in the plain model.

5 Non-uniform incrementally verifiable computation

This section introduces *non-uniform IVC (NIVC)*, a generalization of IVC, where at each step of an incremental computation, the prover proves the satisfiability of a relation chosen from a set of possible relations (the choice of which relation to use is made by an additional designated relation), whereas in the standard IVC, there is only one possible relation. As a result of this generalization, the overall relation proven by non-uniform IVC can be a non-uniform circuit (i.e., circuits without repeating structure), which motivates its name.

As detailed in the introduction, non-uniform IVC implies proofs of program executions on machines with a pre-defined custom instruction set. In Section 6, we construct HyperNova, an efficient NIVC scheme.

In IVC, for a polynomial-time function F, the prover takes as input a claim (i, z_0, z) and a corresponding proof Π_i that proves the knowledge of witnesses $(\omega_0, \ldots, \omega_{i-1})$ such that by computing $z_{j+1} \leftarrow F(z_j, \omega_j)$ for all $j \in \{0, \ldots, i-1\}$ we have that $z = z_i$. Given a new witness ω_i, the prover computes a new proof Π_{i+1} of the same size, which proves the statement $(i+1, z_0, z_{i+1})$ for $z_{i+1} = F(z_i, \omega_i)$.

In NIVC, we extend IVC to handle a number of arbitrary polynomial-time functions $(F_1, \ldots, F_\ell)$. The choice of which function F_j for $j \in [\ell]$ is executed at a particular step in the incremental computation is handled by an additional polynomial-time function φ. More specifically, NIVC captures an incremental proof system for the following augmented statement: There exists $(\omega_0, \ldots, \omega_{i-1})$ such that on initial input z_0 and claimed output z, by computing $z_{j+1} \leftarrow F_{\varphi(z_j, \omega_j)}(z_j, \omega_j)$ for all $j \in \{0, \ldots, i-1\}$, we have that $z = z_i$.

Observe that if we fix $\ell = 1$ and that φ outputs 1, we recovers the definition of IVC [66]. This means that any NIVC scheme is also an IVC scheme.

Definition 11 (Non-uniform IVC). *A non-uniform incrementally verifiable computation (NIVC) 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, with the following interface:*

- $\mathcal{G}(1^\lambda, N) \to \mathsf{pp}$*: on input security parameter λ and size bounds N, samples public parameters* pp.
- $\mathcal{K}(\mathsf{pp}, ((F_1, \ldots, F_\ell), \varphi)) \to (\mathsf{pk}, \mathsf{vk})$*: on input public parameters* pp*, a control function φ, and functions $F_1, \ldots, F_\ell$ deterministically produces a prover key* pk *and a verifier key* vk.
- $\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \to \Pi_{i+1}$*: on input a prover key* pk*, a counter i, initial input z_0, claimed output after i applications z_i, a non-deterministic advice ω_i, and an NIVC proof Π_i attesting to z_i, produces a new proof Π_{i+1} attesting to $z_{i+1} = F_{\varphi(z_i, \omega_i)}(z_i, \omega_i)$.*
- $\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) \to \{0, 1\}$*: on input a verifier key* vk*, a counter i, an initial input z_0, a claimed output after i applications z_i, and an NIVC proof Π_i attesting to z_i, outputs 1 if Π_i is accepting, 0 otherwise.*

An NIVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ satisfies following requirements.

(i) Completeness: For any PPT adversary $\mathcal{A}$ we have that

$$\Pr\left[\, b = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N), \\ (((F_1, \ldots, F_\ell), \varphi), (i, z_0, z_i), (\omega_i, \Pi_i)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, ((F_1, \ldots, F_\ell), \varphi)), \\ \mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) = 1, \\ z_{i+1} \leftarrow F_{\varphi(z_i, \omega_i)}(z_i, \omega_i), \\ \Pi_{i+1} \leftarrow \mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i), \\ b \leftarrow \mathcal{V}(\mathsf{vk}, (i+1, z_0, z_{i+1}), \Pi_{i+1}) \end{array} \right] = 1$$

where $\ell \geq 1$ and φ produces an element in $\mathbb{Z}^*_{\ell+1}$. Moreover, φ and each F_j for $j \in \{1, \ldots, \ell\}$ are a polynomial-time computable function represented as arithmetic circuits.

(ii) Knowledge Soundness: Consider constant $n \in \mathbb{N}$. For all expected polynomial-time adversaries $\mathcal{P}^*$ there exists an expected polynomial-time extractor $\mathcal{E}$ such that

$$\Pr_{\mathsf{r}}\left[\begin{array}{l} z_n = z \text{ where} \\ z_{i+1} \leftarrow F_{\varphi(z_i,\omega_i)}(z_i, \omega_i) \\ \forall i \in \{0, \ldots, n-1\} \end{array} \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N), \\ (((F_1, \ldots, F_\ell), \varphi), (z_0, z), \mathit{\Pi}) \leftarrow \mathcal{P}^*(\mathsf{pp}, \mathsf{r}), \\ (\omega_0, \ldots, \omega_{n-1}) \leftarrow \mathcal{E}(\mathsf{pp}, \mathsf{r}) \end{array} \right] \approx$$

$$\Pr_{\mathsf{r}}\left[\mathcal{V}(\mathsf{vk}, (n, z_0, z), \mathit{\Pi}) = 1 \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N), \\ (((F_1, \ldots, F_\ell), \varphi), (z_0, z), \mathit{\Pi}) \leftarrow \mathcal{P}^*(\mathsf{pp}, \mathsf{r}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, ((F_1, \ldots, F_\ell), \varphi)) \end{array} \right]$$

where r denotes an arbitrarily long random tape.

(iii) Succinctness: The NIVC proof size is independent of the iteration count.

(iv) Efficiency: The prover's time complexity at any step i is linear in the size of the function applied at step i and the total number of functions ℓ.

6 HyperNova: NIVC from multi-folding schemes

We now describe HyperNova, a general compiler that takes a multi-folding scheme for an NP-complete relation with mild requirements and produces an NIVC scheme. For simplicity, we focus on constructing NIVC, but our construction extends naturally to provide a generalization of IVC to distributed computations called proof-carrying data (PCD) [22,7].

In Section 6.1, we provide an informal overview of HyperNova, instantiated with the multi-folding scheme for CCCS from Section 4. Next, in Section 6.2 we isolate the necessary properties for a general multi-folding scheme to be used to construct NIVC. We refer to multi-folding schemes that satisfy these properties as *NIVC-compatible*. We then prove that the folding scheme for CCCS is NIVC-compatible. In Section 6.3 we provide a formal construction of HyperNova.

6.1 Overview of HyperNova

We intentionally overlook certain minor complications. We then address these complications before providing a formal construction. For concreteness, we fix CCCS as the NP-complete relation.

Consider efficient functions $\{F_1, \ldots, F_\ell\}$ and φ. Recall that the NIVC statement (i, z_0, z_i) claims the knowledge of $(\omega_0, \ldots, \omega_{i-1})$ such that by computing $z'_{k+1} \leftarrow F_{\varphi(z'_k, \omega_k)}(z'_k, \omega_k)$ for all $k \in \{0, \ldots, i-1\}$ for $z'_0 = z_0$ we have that $z'_i = z_i$.

We now describe a single iterative step of the prover's work. That is, we explain how the prover can take a proof $\mathit{\Pi}_i$ for the NIVC statement (i, z_0, z_i) and efficiently produce an updated proof $\mathit{\Pi}_{i+1}$ for the NIVC statement $(i+1, z_0, z_{i+1})$. At a high level, instead of directly proving the knowledge of a satisfying witness to some

prescribed F_j for $j \in \{1, \ldots, \ell\}$ in each step, the prover proves the knowledge of a satisfying witness to an augmented function F'_j. The augmented function F'_j, in addition to running F_j, performs additional bookkeeping using a folding scheme to help verifiably update the NIVC proof.

At first glance, a straw-man approach is to have each F'_j take as input a CCCS instance that claims the correct execution of the latest iteration and then fold that instance into a running LCCCS instance using the folding scheme in Section 4 (this is the approach taken by Nova [43]). However, the folding scheme for CCCS requires that both instances have compatible structure (which requires that they represent the same computation in their matrices). In the case of standard IVC, as there is only one function that can be applied at each iterative step, this holds naturally. However, this is not the case for non-uniform IVC.

To address this, F'_j instead takes a list U_i of running instances, where $\mathsf{U}_i[j]$ attests to all prior iterations of F'_j up to $i-1$ steps. As such, checking all of U_i is equivalent to checking $i-1$ steps. In addition, F'_j takes as input a new instance u_i, which claims the correctness of the i'th step. Instead of directly checking this instance (which would be concretely expensive), F'_j folds u_i into the appropriate instance in U_i according to φ to produce a new list of running instances U_{i+1}. To claim the correctness of F'_j itself, the prover produces a new instance u_{i+1}.

We let the NIVC proof $\mathit{\Pi}_i$ contain the list U_i, the fresh instance u_i, and the corresponding witnesses. Thus, the prover can use parts of $\mathit{\Pi}_i$ as input to the appropriate function F'_j to produce U_{i+1} and u_{i+1}, and separately compute the corresponding witnesses. These terms together define $\mathit{\Pi}_{i+1}$. At the end of the iterative computation (or at any intermediate step, if necessary), the verifier can check i steps by checking proof $\mathit{\Pi}_i$ directly.

The prior description overlooks the following minor issues. Prior work [43] addresses these (except for the first one), and we now provide an overview of these in light of the above overview.

First, we describe how to update a proof $\mathit{\Pi}_i$ to produce a proof $\mathit{\Pi}_{i+1}$. However, we did not define a base case proof $\mathit{\Pi}_0$ and how the prover, the verifier, and each function F'_j handles the base case. At a high level, we have F'_j populate U with satisfying running instances in the base case.

Second, the non-interactive folding scheme's verifier run by F'_j needs additional advice generated by the non-interactive folding scheme's prover. To address this, the prover provides additional non-deterministic input to F'_j.

Finally, there is a subtle sizing issue in the above description: in each step, because U_{i+1} is produced as the public IO of $F'_{\mathsf{pc}_{i+1}}$, it must be contained in the public IO of instance u_{i+1}. In the next iteration, because u_{i+1} is folded into $\mathsf{U}_{i+1}[\mathsf{pc}_{i+1}]$, this means that $\mathsf{U}_{i+1}[\mathsf{pc}_{i+1}]$ is at least as large as U_i by the properties of the folding scheme. This means that the list of running instances grows in each step. To alleviate this issue, we have each F'_j only produce a hash of its outputs as

public output. In the subsequent step, the next augmented function takes as non-deterministic input a preimage to this hash.

6.2 NIVC-Compatible multi-folding schemes

Generalizing the above discussion, a multi-folding scheme for an arbitrary committed relation $\mathcal{R}_2$ can be used for NIVC if it satisfies the following properties: First, statements about the correct execution of an efficient function F can be encoded (and decoded) as statements in the underlying relation of $\mathcal{R}_2$. We refer to this property as *NP-completeness*. Second, structures and instances can be encoded (and decoded) independently of witnesses. We refer to this property as *partial functions*. Third, we must have that any efficient function F can be encoded as an $\mathcal{R}_2$ structure in a way that preserves the size of F. We refer to this property as *monotonicity*. Fourth, we must have that there exists a default satisfying instance-witness pair in $\mathcal{R}_1$ (this is required for the base case of our NIVC construction). We refer to this property as *default instances*. We formally define NIVC-compatibility as follows.

Definition 12 (NIVC-compatible multi-folding scheme). *Consider a relation $\mathcal{R}_1$, and a committed relation $\mathcal{R}_2$ over an underlying relation $\mathcal{R}'_2$. A succinct, non-interactive multi-folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ with deterministic $\mathcal{V}$ for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, 1, 1)$ is NIVC-compatible if it satisfies the following properties.*

1. *NP-completeness: There exists a deterministic polynomial-time efficiently invertible function enc such that for any arithmetic circuit F, input x, non-deterministic input w, and output y, for structure-instance-witness tuple $(\mathsf{s}_2, \mathsf{u}, \mathsf{w}) \leftarrow \mathsf{enc}(F, (x, y), w)$ we have that $(\mathsf{s}_2, \mathsf{u}, \mathsf{w}) \in \mathcal{R}'_2$ iff $F(x, w) = y$.*
2. *Partial functions: There exists deterministic, efficiently-invertible polynomial-time functions $\mathsf{enc}_{\mathsf{str}}$ and $\mathsf{enc}_{\mathsf{inst}}$ such that for any arithmetic circuit F, input x, non-deterministic input w, and output y, for $\mathcal{R}'_1$ and $\mathcal{R}'_2$ structures $(\mathsf{s}_1, \mathsf{s}_2) \leftarrow \mathsf{enc}_{\mathsf{str}}(F)$ and $\mathcal{R}'_2$ instance $\mathsf{u} \leftarrow \mathsf{enc}_{\mathsf{inst}}((x, y))$ we have that $(\mathsf{s}_2, \mathsf{u}, \mathsf{w}) = \mathsf{enc}(F, (x, y), w)$ for some $\mathcal{R}'_2$ witness w and that $\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = 1$.[5]*
3. *Monotonicity: For arithmetic circuits F and G, given $|F| \leq |G|$ we have that $|\mathsf{enc}_{\mathsf{str}}(F)| \leq |\mathsf{enc}_{\mathsf{str}}(G)|$. The term $|F|$ denotes the total number of gates in F and the term $|\mathsf{enc}_{\mathsf{str}}(F)|$ denotes the total number of constraints in $\mathsf{enc}_{\mathsf{str}}(F)$.*
4. *Default instances: There exists $(\mathsf{u}_\perp, \mathsf{w}_\perp)$ such that for any public parameters pp and structure s, we have that $(\mathsf{pp}, \mathsf{s}, \mathsf{u}_\perp, \mathsf{w}_\perp) \in \mathcal{R}_1$.*

Our multi-folding scheme for CCS (Construction 1) is NIVC-compatible.

Lemma 3 (NIVC-compatibility). *Construction 1 is NIVC-compatible.*

[5] Note that the required property on w is captured in the NP-completeness requirement.

Proof (Intuition). NP-completeness of $\mathcal{R}_{\mathsf{CCS}}$ follows from [60, Lemma 1], which reduces an NP-complete relation, R1CS, to $\mathcal{R}_{\mathsf{CCS}}$. Moreover, a structure, instance, and witness tuple in $\mathcal{R}_{\mathsf{CCS}}$ has enough information to reconstruct the original function F and its inputs and outputs. This implies invertibility of enc. Since CCS generalizes R1CS, we have that that the $\mathcal{R}_{\mathsf{CCCS}}$ structure (i.e., constraint matrices) depends only on the function F (and not inputs) and in turn can be used to reconstruct F and that the $\mathcal{R}_{\mathsf{CCCS}}$ instance depends only on the public inputs and outputs and can be used to reconstruct these values. This implies the partial function requirement. Moreover, monotonicity holds from the reasoning in Setty et al. [60]. Finally, we have that $\mathcal{R}_{\mathsf{LCCCS}}$ has default instances because for any public parameters and structure, we have that $(u = 0, \mathsf{x} = \vec{0}, r = 0, v_1 = 0, \ldots, v_t = 0)$ and $\widetilde{w} = 0$ is a satisfying instance-witness pair. We provide a formal proof in Appendix H.2. □

6.3 A compiler from NIVC-compatible folding schemes to NIVC

Construction 2 (NIVC from multi-folding schemes). Consider a relation $\mathcal{R}_1$ and a committed relation $\mathcal{R}_2$ for a commitment scheme $(\mathsf{Commit}, \mathsf{Gen})$. Let NIFS be an NIVC-compatible non-interactive multi-folding scheme for a single instance of $\mathcal{R}_1$ and $\mathcal{R}_2$. Let $(\mathsf{u}_\perp, \mathsf{w}_\perp)$ be a default instance-witness pair for $\mathcal{R}_1$ that satisfies any structure and public parameters. We construct an NIVC scheme as follows.

Consider a deterministic polynomial-time function φ and ℓ polynomial-time functions $(F_1, \ldots, F_\ell)$ that take non-deterministic input and a cryptographic hash function hash. We first define augmented functions F'_j for $j \in [\ell]$, where all input arguments are taken as non-deterministic advice, as follows.

$F'_j(\mathsf{vk}_{\mathsf{fs}}, \mathsf{U}_i, \mathsf{u}_i, \mathsf{pc}_i, (i, z_0, z_i), \omega_i, \pi) \to \mathsf{x}$:

1. Compute the next program counter $\mathsf{pc}_{i+1} \in [\ell] \leftarrow \varphi(z_i, \omega_i)$.
2. Compute the next output $z_{i+1} \leftarrow F_j(z_i, \omega_i)$.
3. If $i = 0$:
   (a) Check that $z_0 = z_i$ to ensure that the statement holds in the base case.
   (b) Set $\mathsf{U}_{i+1} \leftarrow (\mathsf{u}_\perp, \ldots, \mathsf{u}_\perp)$.
4. Otherwise:
   (a) Parse u_i as (C, u'_i), a commitment to the witness and the remainder.
   (b) Check that u'_i references U_i in the output of the prior iteration:

$$\mathsf{u}'_i \stackrel{?}{=} \mathsf{enc}_{\mathsf{inst}}(\mathsf{hash}(\mathsf{vk}_{\mathsf{fs}}, i, z_0, z_i, \mathsf{U}_i, \mathsf{pc}_i)).$$

   (c) Check that $1 \leq \mathsf{pc}_i \leq \ell$.

(d) Copy $\mathsf{U}_{i+1} \leftarrow \mathsf{U}_i$ and update $\mathsf{U}_{i+1}[\mathsf{pc}_i] \leftarrow \mathsf{NIFS}.\mathcal{V}(\mathsf{vk}_{\mathsf{fs}}[\mathsf{pc}_i], \mathsf{U}_i[\mathsf{pc}_i], \mathsf{u}_i, \pi)$.

5. Output $\mathsf{x} \leftarrow \mathsf{hash}(\mathsf{vk}_{\mathsf{fs}}, i+1, z_0, z_{i+1}, \mathsf{U}_{i+1}, \mathsf{pc}_{i+1})$.

Next, we define the NIVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ as follows.

$\underline{\mathcal{G}(1^\lambda, N) \to \mathsf{pp}}$: Output $\mathsf{NIFS}.\mathcal{G}(1^\lambda, N)$.

$\underline{\mathcal{K}(\mathsf{pp}, (\varphi, (F_1, \ldots, F_\ell))) \to (\mathsf{pk}, \mathsf{vk})}$:

1. Compute $(\mathsf{s}_{1,j}, \mathsf{s}_{2,j}) \leftarrow \mathsf{enc}_{\mathsf{str}}(F'_j)$ for all $j \in [\ell]$.
2. Compute $(\mathsf{pk}_{\mathsf{fs},j}, \mathsf{vk}_{\mathsf{fs},j}) \leftarrow \mathsf{NIFS}.\mathcal{K}(\mathsf{pp}, \mathsf{s}_{1,j}, \mathsf{s}_{2,j})$ for all $j \in [\ell]$.
3. Compute and output the prover and verifier keys.

$$\mathsf{vk} \leftarrow (\mathsf{pp}, (\mathsf{vk}_{\mathsf{fs},1}, \ldots, \mathsf{vk}_{\mathsf{fs},\ell}), (\mathsf{s}_{1,1}, \ldots, \mathsf{s}_{1,\ell}), (\mathsf{s}_{2,1}, \ldots, \mathsf{s}_{2,\ell}))$$
$$\mathsf{pk} \leftarrow ((\varphi, (F_1, \ldots, F_\ell)), (\mathsf{pk}_{\mathsf{fs},1}, \ldots, \mathsf{pk}_{\mathsf{fs},\ell}), \mathsf{vk})$$

$\underline{\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \mathit{\Pi}_i) \to \mathit{\Pi}_{i+1}}$:

1. Parse $\mathit{\Pi}_i$ as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \mathsf{pc}_i)$.
2. Compute the next program counter $\mathsf{pc}_{i+1} \in [\ell] \leftarrow \varphi(z_i, \omega_i)$.
3. If $i = 0$: Let $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi) \leftarrow ((\mathsf{u}_\perp, \ldots, \mathsf{u}_\perp), (\mathsf{w}_\perp, \ldots, \mathsf{w}_\perp), \perp)$.

   Otherwise: Copy $\mathsf{U}_{i+1} \leftarrow \mathsf{U}_i$ and $\mathsf{W}_{i+1} \leftarrow \mathsf{W}_i$, and update

   $$(\mathsf{U}_{i+1}[\mathsf{pc}_i], \mathsf{W}_{i+1}[\mathsf{pc}_i], \pi) \leftarrow \mathsf{NIFS}.\mathcal{P}(\mathsf{pk}[\mathsf{pc}_i], (\mathsf{U}_i[\mathsf{pc}_i], \mathsf{W}_i[\mathsf{pc}_i]), (\mathsf{u}_i, \mathsf{w}_i)).$$

4. Compute the output $y \leftarrow F'_{\mathsf{pc}_{i+1}}(\mathsf{vk}_{\mathsf{fs}}, \mathsf{U}_i, \mathsf{u}_i, \mathsf{pc}_i, (i, z_0, z_i), \omega_i, \pi)$.
5. Compute an instance-witness pair encoding the valid execution of $F'_{\mathsf{pc}_{i+1}}$:

   $$(\_, \mathsf{u}'_{i+1}, \mathsf{w}_{i+1}) \leftarrow \mathsf{enc}(F'_{\mathsf{pc}_{i+1}}, (\perp, y), (\mathsf{vk}_{\mathsf{fs}}, \mathsf{U}_i, \mathsf{u}_i, \mathsf{pc}_i, (i, z_0, z_i), \omega_i, \pi)).$$

6. Compute the committed instance: $\mathsf{u}_{i+1} \leftarrow (\mathsf{Commit}(\mathsf{pp}, \mathsf{w}_{i+1}), \mathsf{u}'_{i+1})$.
7. Output $\mathit{\Pi}_{i+1} \leftarrow ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}), \mathsf{pc}_{i+1})$

$\underline{\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \mathit{\Pi}_i) \to \{0,1\}}$:

1. If $i = 0$, output 1 if $z_i = z_0$ and 0 otherwise.
2. Parse $\mathit{\Pi}_i$ as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \mathsf{pc}_i)$.
3. Parse u_i as (C, u'_i). Check that $\mathsf{u}'_i = \mathsf{enc}_{\mathsf{inst}}(\mathsf{hash}(\mathsf{vk}_{\mathsf{fs}}, i, z_0, z_i, \mathsf{U}_i, \mathsf{pc}_i))$.
4. Check that $1 \leq \mathsf{pc}_i \leq \ell$.
5. Check $(\mathsf{pp}, \mathsf{s}_{1,j}, \mathsf{U}_i[j], \mathsf{W}_i[j]) \in \mathcal{R}_1$ for $j \in [\ell]$ and $(\mathsf{pp}, \mathsf{s}_{2,\mathsf{pc}_i}, \mathsf{u}_i, \mathsf{w}_i) \in \mathcal{R}_2$.

We formally prove the following lemma in Appendix H.3.

Lemma 4 (NIVC from multi-folding schemes). *Construction 2 takes a NIVC-compatible multi-folding scheme and produces an NIVC scheme.*

Proof (Intuition). To prove knowledge soundness, suppose that we have a malicious prover $\mathcal{P}_i^*$ that successfully produces a convincing NIVC proof Π_i of i iterations with non-negligible probability. Using $\mathcal{P}_i^*$, we construct a corresponding extractor $\mathcal{E}_{i-1}$ which can use this prover to extract a proof Π_{i-1} and the witness ω_{i-1} by the knowledge soundness of the underlying multi-folding scheme. This extractor then implies a corresponding prover $\mathcal{P}_{i-1}^*$ which can use this extractor to produce a successful proof Π_{i-1} with non-negligible property. Then, recursively repeating this process, we can derive the full list of witnesses. □

Below, we state the concrete efficiency characteristics of HyperNova instantiated with our multi-folding scheme for CCS (Construction 1), which is in turn is instantiated with the Pedersen commitment scheme.

Optimization. As an immediate optimization, we have the verifier circuit inside F'_j for $j \in [\ell]$ use standard memory checking techniques to verifiably read and write the appropriate running instance into externalized memory rather than directly passing all running instances through each step of F'_j. This provides asymptotic improvements: For instance the verifier circuits work is $O(\log \ell)$ hashes when using Merkle trees [49,16], and $O(1)$ hashes and elliptic curve hash-to-curve and point additions when using a multiset-CRHF-based memory [8,23,58]. Then, we have an NIVC scheme with an a-la-carte cost profile, where the cost of each recursive step only scales with the particular function executed at that step.

Theorem 3 (HyperNova). *Given the multi-folding scheme in Construction 1 instantiated with the Pedersen commitment scheme, Construction 2 produces an NIVC scheme such that for step functions F_j for $j \in [\ell]$ that can be expressed in CCS with m_j constraints of degree d and q_j monomials, n_j witness variables, t_j CCS matrices, and N_j non-zero entries in the CCS matrices, and control function φ that can be expressed in CCS with m constraints of degree d and q_φ monomials, n_φ witness variables, t_φ CCS matrices, and N_φ non-zero entries in the CCS matrices, the efficiency characteristics are as follows: The NIVC prover time for a step proving the correct execution of F_j is a single MSM of size $O(n_\varphi + n_j)$ and $O((N_\varphi + N_j) + (t_\varphi + t_j) \cdot (m_\varphi + m_j) + (q_\varphi + q_j) \cdot (m_\varphi + m_j) \cdot d \cdot \log^2 d)$ finite field operations. The verifier circuit size is $o(|\varphi| + 1 \cdot \mathsf{G} + 2 \cdot \mathsf{H}_{\ell,t_j} + d \cdot \log m_j \cdot \mathsf{F} + \log m_j \cdot \mathsf{R}_d + 2 \cdot \mathsf{M})$, where $|\varphi|$ denotes the size of the constraint system for encoding φ in the verifier circuit, G is the number of constraints required to encode a group scalar multiplication, H_{ℓ,t_j} is the number of constraints required to encode hash (which depends on ℓ and t_j), F is the number of constraints to encode field operations, R_d is the number of constraints to encode the RO ρ, and M is the number of constraints to encode to memory read/write over $O(\ell)$ elements.*

Proof. The prover time complexity follows from Theorem 1. As for the verifier circuit size, on input instances U and u, $\mathsf{NIFS.V}$ computes $\mathsf{U}.C \leftarrow \mathsf{U}.C + \rho \cdot u.C$,

which costs a single group scalar multiplication. Verifying the non-interactive sum-check proof in the non-interactive multi-folding scheme proof requires the verifier to perform $O(d \cdot \log m_\varphi + m_j)$ finite field operations and $O(\log m_\varphi + m_j)$ calls to the RO to obtain challenges in the sum-check protocol. By construction, the verifier circuit calls φ once and makes two additional calls to hash. Finally, two memory operations are required to read and write a running instance. $\square$

7 HyperNova's zero-knowledge and succinctness layer

In HyperNova, NIVC proofs are linear in the sizes of circuits for each supported function and may reveal information about the secret witnesses in each step of execution. This section describes how to provide a zero-knowledge argument of a valid NIVC proof. Formally, our goal is to design a zero-knowledge argument (Definition 23) for the following relation. We achieve this without employing zkSNARKs (solving the problem motivated earlier in Section 1.2). For additional succinctness, one can employ a non-zk SNARK.

Definition 13 (Proof of Valid NIVC Proof). *Let* NIVC *denote the NIVC scheme described in Construction 2. We define the relation* $\mathcal{R}_{\mathsf{VNIVC}}$ *over public parameter, structure, instance, and witness tuples as follows.*

$$\mathcal{R}_{\mathsf{VNIVC}} = \left\{ (\mathsf{pp}, (F_1, \ldots, F_\ell, \varphi), (i, z_0, z_i), \mathit{\Pi}) \;\middle|\; \begin{array}{l} \mathsf{vk} \leftarrow \mathsf{NIVC}.\mathcal{K}(\mathsf{pp}, (F_1, \ldots, F_\ell, \varphi)), \\ \mathsf{NIVC}.\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \mathit{\Pi}) = 1 \end{array} \right\}$$

Recall that an NIVC proof consists of running instances U and the corresponding witnesses W, the latest instance u and the corresponding witness w, and the latest index pc. To check a statement (i, z_0, z_i), the NIVC verifier checks the list of running instances U against witnesses W, checks the latest instance u against witness w with respect to F'_{pc}, and checks that u references (i, z_0, z_i) and U.

A straw-man solution is to simply run a zkSNARK proving that the NIVC verifier accepts some proof $\mathit{\Pi}$ with respect to a prescribed verifier key vk. However, this is prohibitively expensive, as it would involve a universal circuit that checks all running instance-witness pairs internally (including the task of checking if the provided witnesses are valid openings of commitments in the instances). Also, as noted in Section 1.2, this entails significant verifier costs in some settings.

Achieving zero-knowledge. To avoid zkSNARKs, our central idea is to instead *rerandomize* an NIVC proof using a much more efficient folding scheme.

We formalize this construction and prove its properties in Appendix D (Construction 6). Here, we provide an overview.

To ensure that an NIVC proof $\mathit{\Pi}$ does not reveal any secret information, the prover does the following: First, to hide the last instruction pc, the prover verifiably folds (u, w) into $(\mathsf{U}[\mathsf{pc}], \mathsf{W}[\mathsf{pc}])$ without revealing any of the involved terms. Next, the prover verifiably folds in randomized instances $(\mathsf{U}_\mathsf{r}, \mathsf{W}_\mathsf{r})$ into (U, W) to produce a new set of randomized running instances $(\mathsf{U}', \mathsf{W}')$, that reveal

no information about (U, W) or $(\mathsf{U}_\mathsf{r}, \mathsf{W}_\mathsf{r})$, but can be checked in place of the original instance-witness pairs. Finally, the prover produces a randomized proof, which consists of $(\mathsf{U}', \mathsf{W}')$ along with proofs of correct folding.

The central challenge with the above strategy is that the prover must verifiably fold the instance-witness pairs in the first two steps without revealing any information about the randomizing instances (and corresponding witnesses) to the verifier. Due to this constraint, the prover cannot directly engage in a folding scheme with the verifier. Instead, the prover executes the verifier's end of the folding scheme in an auxiliary circuit blind which takes as secret input $(\mathsf{u}, \mathsf{U}, \mathsf{pc})$ and randomized instances U_r. blind performs the standard checks on u before folding in u into $\mathsf{U}[\mathsf{pc}]$ and then folds each of the randomized instances U_r into U to produce and output the randomized running instances U'. The prover then produces a corresponding instance-witness pair $(\mathsf{u}_{\mathsf{blind}}, \mathsf{w}_{\mathsf{blind}})$ that attests to the correct execution of the blind circuit itself.

Remark 2. If we are interested in randomizing an IVC proof, specifically Nova's IVC proof, as opposed to an NIVC proof, then the blind circuit can be avoided as there is no need to hide pc. We describe this idea further in Appendix D.4.

Several problems remain. First, we must ensure that there actually exists a method to sample $(\mathsf{U}_\mathsf{r}, \mathsf{W}_\mathsf{r})$. Moreover, we must ensure that the folding scheme used to randomize (U, W) satisfies the following property: Given one of the input instance-witness pairs is randomly sampled, we must have that the output folded instance-witness pair is indistinguishable from random. We refer to a folding scheme that satisfies this property as a *randomizing folding scheme* (Definition 28) and argue that the folding scheme for committed CCS is randomizing (Lemma 8). Then, we can ensure that $(\mathsf{U}', \mathsf{W}')$ reveals no information.

Second, the prover cannot directly reveal the instance-witness pair $(\mathsf{u}_{\mathsf{blind}}, \mathsf{w}_{\mathsf{blind}})$ attesting to the correct execution of blind as $\mathsf{w}_{\mathsf{blind}}$ will implicitly contain pc and several other sensitive terms. Seemingly, we can use a randomizing folding scheme again, where the prover samples $(\mathsf{u}_{\mathsf{rb}}, \mathsf{w}_{\mathsf{rb}})$ folds it into $(\mathsf{u}_{\mathsf{blind}}, \mathsf{w}_{\mathsf{blind}})$ and only reveals the randomized instance-witness pair $(\mathsf{u}'_{\mathsf{blind}}, \mathsf{w}'_{\mathsf{blind}})$ as well as an (interactive) proof of correct folding π_{blind}. However, this may not be sufficient because π_{blind} may itself reveal information about $\mathsf{w}_{\mathsf{blind}}$ even if $(\mathsf{u}'_{\mathsf{blind}}, \mathsf{w}'_{\mathsf{blind}})$ does not. To account for this, we require a folding scheme with a slightly stronger property, in which the transcript (and output) can be simulated so long as one of the inputs is random. We refer to a folding scheme that satisfies this stronger property as a *hiding folding scheme*. Unfortunately, the folding scheme for committed CCS, as presented, is not a hiding folding scheme as the interaction may reveal information about the witness. To remedy this, we instead use the folding scheme underlying Nova, which we demonstrate satisfies the required hiding property. Then, we can ensure that $(\mathsf{u}'_{\mathsf{blind}}, \mathsf{w}'_{\mathsf{blind}})$ and π_{blind} reveal no information about $\mathsf{w}_{\mathsf{blind}}$.

The use of Nova's folding scheme rather than HyperNova's in the zero-knowledge layer does not pose efficiency problems: the zero-knowledge layer is applied only once in the "end" before externalizing NIVC proofs. Furthermore, the work performed inside blind consists only of the folding scheme verifier (which is quite efficient to represent in R1CS, the computational model of Nova's folding scheme).

Altogether, the prover's final blinded proof consists of the blinded running instance-witness pairs $(\mathsf{U}', \mathsf{W}')$, an instance $\mathsf{u}_{\mathsf{blind}}$ attesting to the correct execution of blind, a randomized instance-witness pair $(\mathsf{u}'_{\mathsf{blind}}, \mathsf{w}'_{\mathsf{blind}})$, and an (interactive) proof π_{blind} attesting that checking $(\mathsf{u}'_{\mathsf{blind}}, \mathsf{w}'_{\mathsf{blind}})$ implies checking $\mathsf{u}_{\mathsf{blind}}$.

Achieving non-interactivity and succinctness. Appendix D demonstrates that the above construction is *honest-verifier* zero-knowledge (i.e., zero-knowledge only if the verifier behaves honestly in the interaction). One can heuristically make it zero-knowledge and non-interactive by employing the Fiat-Shamir transformation in a standard manner.

For some applications, further succinctness may be required. In such a situation, the prover can succinctly prove the knowledge of a randomized proof $((\mathsf{U}', \mathsf{W}'), \mathsf{u}_{\mathsf{blind}}, (\mathsf{u}'_{\mathsf{blind}}, \mathsf{w}'_{\mathsf{blind}}), \pi_{\mathsf{blind}})$ by using a SNARK to prove each instance in U' and $\mathsf{u}'_{\mathsf{blind}}$. This is sufficient as the remainder of the blinded proof is constant-sized. This approach of randomizing first, then adding a succinctness layer affords two benefits. First, there is no need to use a *zero-knowledge* SNARK as the randomizing step ensures that the randomized proof reveals no sensitive information. Second, this SNARK can be independently used on each of the instances in the randomized proof, as opposed to a universal circuit. This avoids having to simulate the SNARK verifier inside a circuit.

8 HyperNova over a two-cycle of curves with CycleFold

This section describes how to instantiate HyperNova over a cycle of elliptic curves, which unlocks a concretely-efficient construction that can be implemented. We motivate a cycle of elliptic curves below, but we refer to prior works [4,52] for more details. We focus on HyperNova, but our approach is generic and applies to other folding-scheme-based IVC schemes. It also improves upon prior approach that was proposed in the context of SNARK-based IVC [4].

8.1 Prior approaches and downsides for using them for HyperNova

We first recall the 2-cycle approach to instantiate SNARK-based recursive arguments in [4]. We then describe how an implementation of Nova [1,52] adapts this approach to the context of folding-scheme-based recursive arguments.

The 2-cycle approach in [4]. The starting point for [4] is a pairing-based SNARK (e.g., [54,5]) instantiated over a pairing-friendly elliptic curve E. The proof system can prove constraint systems defined over E's scalar field. Furthermore, verifying a proof requires a handful of pairing operations, which are naturally represented as operations over E's base field.

Let (Π_1, Π_2) denote two SNARK schemes (such as [54,5]) defined respectively over (E_1, E_2). In particular, Π_1 can "natively" (i.e., without field emulation) prove constraint systems (e.g., R1CS) defined over the scalar field of E_1 and Π_2 can prove constraint systems defined over the scalar field of E_2.[6] Naturally, proofs produced by Π_1 can be efficiently verified by a constraint system supported by Π_2 and vice versa. This is because the algorithm to verify proofs produced by Π_1 involves operations over E_1's base field, which, by design, equals the scalar field of E_2. (When the fields do not match, one would need to emulate arithmetic of the desired field using another field, which entails significant costs in terms of the number of gates necessary to perform basic operations such as additions and multiplications over the desired field.) In other words, the constraint system supported by Π_2 can efficiently encode the SNARK verifier of Π_1.

To realize IVC, at step i, in [4], the prover proceeds as follows (for ease of exposition, we ignore the base case of $i = 0$).

1. Using Π_1, the prover produces a SNARK $\pi_i^{(1)}$ that proves that it has executed the step i of the desired computation and has successfully verified a SNARK $\pi_{i-1}^{(2)}$ from step $i - 1$.
2. Using Π_2, the prover produces a SNARK $\pi_i^{(2)}$ that it knows a SNARK $\pi_i^{(1)}$ and has successfully verified it.

Note that $\pi_i^{(2)}$ is the IVC proof at the end of step i. At step $i + 1$, the prover starts with $\pi_i^{(2)}$ and repeats the above procedure for the $(i + 1)$th step of the computation. A key take-away is that this approach requires representing the SNARK verifier as a circuit on both curves in the cycle.

Nova's instantiation over a 2-cycle of elliptic curves. The Nova library [1] adapts [4]'s blueprint to the context of folding schemes, and obtains a concretely-efficient implementation of Nova [43]. Its approach is to essentially replace "SNARK verifier" with a "non-interactive folding scheme verifier". Specifically, an NP instance defined over the scalar field of the first curve can be efficiently folded using a circuit defined over the scalar field of the second curve and vice versa. Different from [4], Nova's IVC proof is a set of instances and witnesses defined over *both* curves in the cycle rather than a single SNARK. Nova additionally uses the public IO of circuits to track folded NP instances. A recent work [52] provides a rigorous and detailed description of Nova's instantiation on a 2-cycle of elliptic curves and proves its security. This work also exposes a vulnerability in the original implementation (which is now fixed). Overall, Nova's approach, like in [4], still requires representing a verifier (which happens to be the non-interactive folding scheme verifier) as a circuit on *both* curves in the cycle of curves. For Nova [43], which provides the most efficient folding scheme verifier in

[6] [4] uses cycles of elliptic curves where both curves are pairing-friendly as they use pairing-based SNARKs to realize IVC. Unfortunately, such cycles of pairing-friendly elliptic curves require field sizes to be much larger than ordinary elliptic curves to achieve a "standard" 128 bits of security.

the literature, the circuit defined over the second curve in the cycle is $\approx 10,000$ multiplication gates.

Additional downsides in the context of HyperNova. If the approach in Nova's implementation [1,52] is applied to HyperNova to instantiate HyperNova over a 2-cycle of elliptic curves, it requires significant non-native arithmetic. In particular, HyperNova's verifier circuit on the scalar field of E_2 must verify a sum-check proof produced on the scalar field of E_1. This involves representing operations over the scalar field of E_1 in a circuit defined over the scalar field of E_2. Since the two scalar fields are different, this would require field emulation, which is concretely expensive (e.g., thousands of constraints for each field multiplication and verifying a sum-check proof requires $O(d \cdot \log m)$ field operations).

8.2 CycleFold's approach

CycleFold's starting point is the observation that folding-scheme-based recursive arguments (e.g., Nova, HyperNova) can be efficiently instantiated *without* a cycle of elliptic curves—except for a few elliptic scalar multiplication operations (2 in Nova, 1 in HyperNova) in their verifier circuits that must be handled with "wrong" field arithmetic (or non-native arithmetic). We further observe that this scalar multiplication operation can be *verifiably delegated* to the second curve with the following approach. We first represent the desired scalar multiplication operation as a circuit over the scalar field of the second curve. Crucially, this avoids non-native arithmetic for computing the scalar multiplication operation (as there is no need for field emulation). Then, by employing Nova's folding scheme verifier on the first curve, we fold that scalar multiplication circuit satisfiability instance into a running instance. Figure 1 depicts CycleFold's approach.

Note that CycleFold can be viewed as employing a cycle of elliptic curves at a *different* level of abstraction than [4] or its adaptation in Nova [43,1,52]. Specifically, with CycleFold, the cycle of elliptic curves is used at the level of a folding scheme. In particular, the specific way the cycle of elliptic curves is used ensures that the folding scheme verifier *can* be efficiently represented as a circuit with a *single* curve in the cycle. Accordingly, the resulting IVC scheme nor its proof of security has to reason about the cycle of elliptic curves. Indeed, when we apply CycleFold to HyperNova, we apply it at the level of a folding scheme.

A preliminary design. CycleFold employs a 2-cycle of curves (E_1, E_2), but it instantiates a folding-scheme-based recursive argument as if there is only a *single* elliptic curve E_1. This means that the folding-scheme verifier is represented as a circuit, say C_{V}, on the scalar field of E_1. For the case of HyperNova, C_{V} performs finite field and hash operations, and a *single* scalar multiplication (more precisely, a scalar multiplication followed by a point addition). The finite field and hashing operations in C_{V} are over E_1's scalar field so they are represented efficiently in E_1's scalar field. However, the scalar multiplication and point addition operations require arithmetic over E_1's base field. Naively, one can perform those operations with non-native arithmetic inside C_{V}. Unfortunately, this strategy will result in C_{V} containing a million multiplication gates or more.

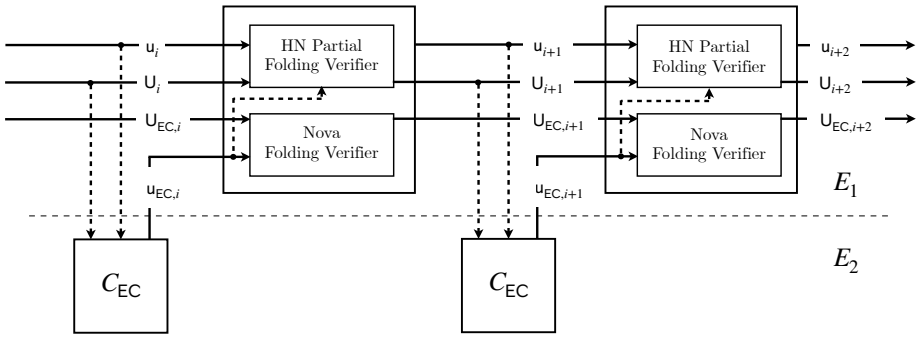

Fig. 1. Two incremental steps in HyperNova's recursive argument instantiated with CycleFold. u_i attests to the computation at step i and U_i attests to all prior steps of the computation. $\mathsf{U}_{\mathsf{EC},\mathsf{i}}$ attests to all prior steps of the outsourced elliptic curve operations. C_{EC} is a circuit which computes the outsourced elliptic curve operations on E_2. u_i and U_i are parsed to retrieve inputs for circuit C_{EC} (represented with a dotted line). $\mathsf{u}_{\mathsf{EC},i}$ represents the correct execution of C_{EC}. The main computation on each step additionally runs the HyperNova folding scheme verifier (which folds claims regarding the main computation) by taking as auxiliary advice the result of the elliptic curve operation (read from $\mathsf{u}_{\mathsf{EC},i}$). The main computation additionally runs the Nova folding scheme verifier which folds claims about the outsourced elliptic curve operation. u_{i+1} represents the correctness of the latest step and $(\mathsf{U}_i, \mathsf{U}_{\mathsf{EC},i})$ represents the correctness of all prior steps and outsourced computations.

We now discuss how CycleFold avoids the non-native arithmetic to compute a scalar multiplication and a point addition—without using the 2-cycle approach of [4] or its adaptation in Nova [43,1,52].

A "co-processor" circuit over the scalar field of $\mathbf{E_2}$. CycleFold creates a circuit C_{EC} defined over the scalar field of the second curve in the cycle E_2 (e.g., on Grumpkin). C_{EC} performs the desired scalar multiplication and a point addition operation. Furthermore, the public IO of C_{EC} contains the inputs and outputs of the scalar multiplication and point addition operation. Since C_{EC} is defined over the scalar field of E_2, which is the base field of E_1 since (E_1, E_2) is a 2-cycle of elliptic curves. As a result, C_{EC} does not require non-native arithmetic to compute the desired scalar multiplication and point addition. In particular, the size C_{EC} is concretely small (e.g., with ≈1,000–1,500 multiplication gates).

Closing the loop. Instead of performing a scalar multiplication and a point addition with non-native arithmetic (which as noted above is untenable), the verifier circuit C_{V} takes as non-deterministic input, among other things, a circuit

satisfiability instance u_{EC} (i.e., the public IO and a commitment to a purported satisfying witness to an instance of C_{EC}). In addition to performing the rest of folding scheme verifier's work, C_{V} consumes the claimed output from the public IO of u_{EC} after checking that inputs to C_{EC} match its desired inputs. C_{V} then folds u_{EC} into a running instance, using Nova's folding scheme.

Appendix E provides a formal construction of HyperNova over a cycle of elliptic curves and proves its correctness. We summarize our result with the following theorem. For simplicity, we formalize the case folding a single fresh instance into a single running instance. However, our construction can be naturally generalized for an arbitrary number of instances, as in Construction 1.

Theorem 4 (A multi-folding scheme for CCS over cycles). *Construction 7 is a public-coin multi-folding scheme for* $(\mathcal{R}_1 = \mathcal{R}_{\mathsf{LCCCS}} \times \mathcal{R}_{\mathsf{CRR1CS}}, \mathcal{R}_2 = \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu = 1, \nu = 1)$ *with perfect completeness and knowledge soundness. For a CCS instance with* m *constraints of degree* d *and* q *monomials,* n *witness variables,* t *CCS matrices, and* N *non-zero entries in CCS matrices, and a linearized CCS instance with the same structure, the efficiency characteristics are as follows: The prover time is* $O(N + t \cdot m + q \cdot m \cdot d \cdot \log^2 d)$ *finite field operations and* $O(1)$ *group operations. The verifier time is* $O(d \cdot \log m)$ *finite field operations and* $O(1)$ *group operations. The communication complexity is* $O(d \cdot \log m)$ *finite field elements.*

Proof (Intuition). Completeness, knowledge-soundness, and efficiency hold by similar reasoning as the proof of Theorem 1 and the properties of the folding scheme for relaxed R1CS [43]. We provide a formal proof in Appendix H.4 □

By leveraging Construction 7 made non-interactive in the plain model via the Fiat-Shamir transformation (Construction 3), we get Theorem 2, which follows from [43, Lemma 4] and Theorem 4.

Acknowledgments

We thank Justin Drake, Ariel Gabizon, Bryan Parno, Carlos Pérez, Drew Stone, Justin Thaler, Ioanna Tzialla, and the anonymous CRYPTO and Eurocrypt reviewers for helpful conversations and comments on a prior version of this paper. We thank Tohru Kohrita for pointing out a discrepancy in an earlier version of the knowledge soundness definition for multi-folding schemes. We thank Wilson Nguyen for pointing out an issue with the simulator's description in an earlier version of the proof of Lemma 9. While at Carnegie Mellon University, Abhiram Kothapalli was supported by a fellowship from Protocol Labs, NSF Grant No. 1801369 and 190099, and by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

1. Nova: Recursive SNARKs without trusted setup. `https://github.com/Microsoft/Nova`
2. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.: Fast reductions from RAMs to delegatable succinct constraint satisfaction problems: Extended abstract. In: ITCS (2013)
3. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: CRYPTO (Aug 2013)
4. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: CRYPTO (2014)
5. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von Neumann architecture. In: USENIX Security (2014)
6. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: ITCS (2012)
7. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKs and proof-carrying data. In: STOC (2013)
8. Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. In: FOCS (1991)
9. Boneh, D., Bünz, B., Fisch, B.: A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712 (2018)
10. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo Infinite: Recursive zk-SNARKs from any Additive Polynomial Commitment Scheme. In: CRYPTO (2021)
11. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: EUROCRYPT (2016)
12. Bootle, J., Chiesa, A., Hu, Y., Orrú, M.: Gemini: Elastic snarks for diverse environments. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 427–457 (2022)
13. Bowe, S., Grigg, J., Hopwood, D.: Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021 (2019)
14. Bowe, S., Grigg, J., Hopwood, D.: Halo2 (2020), `https://github.com/zcash/halo2`
15. Braun, B.: Compiling computations to constraints for verified computation. Tech. rep., UT Austin Honors thesis HR-12-10 (Dec 2012)
16. Braun, B., Feldman, A.J., Ren, Z., Setty, S., Blumberg, A.J., Walfish, M.: Verifying computations with state. In: SOSP (2013)
17. Bünz, B., Chen, B.: Protostar: Generic efficient accumulation/folding for special sound protocols. Cryptology ePrint Archive, Paper 2023/620 (2023)
18. Bünz, B., Chiesa, A., Lin, W., Mishra, P., Spooner, N.: Proof-carrying data without succinct arguments. In: CRYPTO (2021)
19. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Proof-carrying data from accumulation schemes. In: TCC (2020)
20. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In: EUROCRYPT (2023)
21. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: EUROCRYPT (2020)
22. Chiesa, A., Tromer, E.: Proof-carrying data and hearsay arguments from signature cards. In: Innovations in Computer Science (ICS) (2010)

23. Clarke, D., Devadas, S., Dijk, M.V., Gassend, B., Edward, G., Mit, S.: Incremental multiset hash functions and their application to memory integrity checking. In: ASIACRYPT (2003)
24. Cormode, G., Mitzenmacher, M., Thaler, J.: Practical verified computation with streaming interactive proofs. In: ITCS (2012)
25. Cramer, R., Damgård, I.: Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free? In: CRYPTO. pp. 424–441 (1998)
26. Eagen, L., Fiore, D., Gabizon, A.: cq: Cached quotients for fast lookups. Cryptology ePrint Archive (2022)
27. Eagen, L., Gabizon, A.: Protogalaxy: Efficient protostar-style folding of multiple instances. Cryptology ePrint Archive, Paper 2023/1106 (2023)
28. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO. pp. 186–194 (1986)
29. Gabizon, A., Williamson, Z.J.: plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive (2020)
30. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. ePrint Report 2019/953 (2019)
31. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: EUROCRYPT (2013)
32. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: STOC. pp. 99–108 (2011)
33. Goldberg, L., Papini, S., Riabzev, M.: Cairo – a Turing-complete STARK-friendly CPU architecture. Cryptology ePrint Archive (2021)
34. Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT (2016)
35. Haböck, U.: Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive (2022)
36. Khovratovich, D., Maller, M., Tiwari, P.R.: MinRoot: candidate sequential function for Ethereum VDF. Cryptology ePrint Archive, Paper 2022/1626 (2022)
37. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: STOC (1992)
38. Kosba, A., Papadopoulos, D., Papamanthou, C., Song, D.: MIRAGE: succinct arguments for randomized algorithms with applications to universal zk-SNARKs. In: USENIX Security (2020)
39. Kothapalli, A., Parno, B.: Algebraic reductions of knowledge. In: CRYPTO (2023)
40. Kothapalli, A., Setty, S.: SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive (2022)
41. Kothapalli, A., Setty, S.: CycleFold: CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. Cryptology ePrint Archive, Paper 2023/1192 (2023)
42. Kothapalli, A., Setty, S.: HyperNova: Recursive arguments for customizable constraint systems. In: CRYPTO (2024)
43. Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. In: CRYPTO (2022)
44. Labs, O.: Mina cryptocurrency (2020), `https://minaprotocol.com`
45. Lee, J., Nikitin, K., Setty, S.: Replicated state machines without replicated execution. In: S&P (2020)
46. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. In: FOCS (Oct 1990)

47. Lurk: https://github.com/lurk-lang
48. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updateable structured reference strings. In: CCS (2019)
49. Merkle, R.C.: A digital signature based on a conventional encryption function. In: CRYPTO (1988)
50. Micali, S.: CS proofs. In: FOCS (1994)
51. Mohnblatt, N.: Sangria: a folding scheme for PLONK. https://geometry.xyz/notebook/sangria-a-folding-scheme-for-plonk (2023)
52. Nguyen, W., Boneh, D., Setty, S.: Revisiting the Nova proof system on a cycle of curves. Cryptology ePrint Archive, Paper 2023/969 (2023)
53. Ozdemir, A., Wahby, R.S., Boneh, D.: Scaling verifiable computation using efficient set accumulators. In: USENIX Security (2020)
54. Parno, B., Gentry, C., Howell, J., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: S&P (May 2013)
55. RISC ZERO: https://www.risczero.com/
56. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. J. ACM **27**(4) (1980)
57. Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: CRYPTO (2020)
58. Setty, S., Angel, S., Gupta, T., Lee, J.: Proving the correct execution of concurrent services in zero-knowledge. In: OSDI (Oct 2018)
59. Setty, S., Braun, B., Vu, V., Blumberg, A.J., Parno, B., Walfish, M.: Resolving the conflict between generality and plausibility in verified computation. In: EuroSys (Apr 2013)
60. Setty, S., Thaler, J., Wahby, R.: Customizable constraint systems for succinct arguments. Cryptology ePrint Archive (2023)
61. Setty, S., Thaler, J., Wahby, R.: Unlocking the lookup singularity with lasso. In: EUROCRYPT (2024)
62. Setty, S., Vu, V., Panpalia, N., Braun, B., Blumberg, A.J., Walfish, M.: Taking proof-based verified computation a few steps closer to practicality. In: USENIX Security (Aug 2012)
63. Solberg, T.: A brief history of lookup arguments. https://github.com/ingonyama-zk/papers/blob/main/lookups.pdf (2023)
64. Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: CRYPTO (2013)
65. Thaler, J.: Proofs, arguments, and zero-knowledge. http://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html (2020)
66. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: TCC. pp. 552–576 (2008)
67. Vu, V., Setty, S., Blumberg, A.J., Walfish, M.: A hybrid architecture for verifiable computation. In: S&P (2013)
68. Wahby, R.S., Setty, S., Ren, Z., Blumberg, A.J., Walfish, M.: Efficient RAM and control flow in verifiable outsourced computation. In: NDSS (2015)
69. Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: S&P (2018)
70. Wesolowski, B.: Efficient verifiable delay functions. In: EUROCRYPT. pp. 379–407 (2019)
71. WhiteHat, B., Gluchowski, A., HarryR, Fu, Y., Castonguay, P.: Roll_up / roll_back snark side chain ~17000 tps. https://ethresear.ch/t/roll-up-roll-back-snark-side-chain-17000-tps/3675 (Oct 2018)

72. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In: S&P (2017)
73. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vRAM: Faster verifiable RAM with program-independent preprocessing. In: S&P (2018)
74. Zhang, Z.: Origami: Fold a Plonk for Ethereum's VDF. Cryptology ePrint Archive (2023)
75. Zheng, T., Gao, S., Guo, Y., Xiao, B.: Kilonova: Non-uniform pcd with zero-knowledge property from generic folding schemes. Cryptology ePrint Archive, Paper 2023/1579 (2023)
76. Zhou, Z., Zhang, Z., Dong, J.: Proof-carrying data from multi-folding schemes. Cryptology ePrint Archive, Paper 2023/1282 (2023)

A Additional Preliminaries

A.1 Polynomials and low-degree extensions

We adapt this subsection from prior work [57]. We start by recalling several facts about polynomials.

Definition 14 (Multilinear polynomial). *A multivariate polynomial is called a multilinear polynomial if the degree of the polynomial in each variable is at most one.*

Definition 15 (Low-degree polynomial). *A multivariate polynomial g over a finite field $\mathbb{F}$ is called low-degree polynomial if the degree d of g in each variable is exponentially smaller than $|\mathbb{F}|$ (i.e., $d = O(\log |\mathbb{F}|)$).*

Low-degree extensions (LDEs). Suppose $g : \{0,1\}^\ell \to \mathbb{F}$ is a function that maps ℓ-bit elements into an element of $\mathbb{F}$. A *polynomial extension* of g is a low-degree ℓ-variate polynomial, denoted $\widetilde{g}$, such that $\widetilde{g}(x) = g(x)$ for all $x \in \{0,1\}^\ell$.

A *multilinear* polynomial extension (or simply, a multilinear extension, or MLE) is a low-degree polynomial extension where the extension is a multilinear polynomial (i.e., the degree of each variable in $\widetilde{g}$ is at most one). Given a function $Z : \{0,1\}^\ell \to \mathbb{F}$, the multilinear extension of Z is the unique multilinear polynomial $\widetilde{Z} : \mathbb{F}^\ell \to \mathbb{F}$. It can be computed as follows.

$$\begin{aligned}\widetilde{Z}(x_1, \ldots, x_\ell) &= \sum_{e \in \{0,1\}^\ell} Z(e) \cdot \prod_{i=1}^{\ell} (x_i \cdot e_i + (1 - x_i) \cdot (1 - e_i)) \\ &= \sum_{e \in \{0,1\}^\ell} Z(e) \cdot \widetilde{eq}(x, e) \\ &= \langle (Z(0), \ldots, Z(2^\ell - 1)), (\widetilde{eq}(x, 0), \ldots, \widetilde{eq}(x, 2^\ell - 1)) \rangle\end{aligned}$$

Note that $\widetilde{eq}(x, e) = \prod_{i=1}^{\ell} (e_i \cdot x_i + (1 - e_i) \cdot (1 - x_i))$, which is the MLE of the following function:

$$eq(x, e) = \begin{cases} 1 & \text{if } x = e \\ 0 & \text{otherwise} \end{cases}$$

For any $r \in \mathbb{F}^\ell$, $\widetilde{Z}(r)$ can be computed in $O(2^\ell)$ operations in $\mathbb{F}$ [67,64].

Dense representation for multilinear polynomials. Since the MLE of a function is unique, it offers the following method to represent any multilinear polynomial. Given a multilinear polynomial $g : \mathbb{F}^\ell \to \mathbb{F}$, it can be represented uniquely by the list of tuples L such that for all $i \in \{0,1\}^\ell$, $(\mathsf{to\text{-}field}(i), g(i)) \in L$ if and only if $g(i) \neq 0$, where $\mathsf{to\text{-}field}$ is the canonical injection from $\{0,1\}^\ell$ to $\mathbb{F}$. We denote such a representation of g as $\mathsf{DenseRepr}(g)$.

Definition 16. *A multilinear polynomial g in ℓ variables is a sparse multilinear polynomial if $|\mathsf{DenseRepr}(g)|$ is sub-linear in 2^ℓ. Otherwise, it is a dense multilinear polynomial.*

As an example, suppose $g : \mathbb{F}^{2s} \to \mathbb{F}$. Suppose $|\mathsf{DenseRepr}(g)| = O(2^s)$, then g is a sparse multilinear polynomial because 2^s is sublinear in 2^{2s}.

A.2 The sum-check protocol

Suppose there is an ℓ-variate low-degree polynomial, g, where the degree of each variable in g is at most d. Suppose that a verifier $\mathcal{V}$ is interested in checking a claim of the following form by an untrusted prover $\mathcal{P}$:

$$T = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_\ell \in \{0,1\}} g(x_1, x_2, \ldots, x_\ell)$$

Of course, given g, $\mathcal{V}$ can deterministically evaluate the above sum and verify whether the sum is T. But, this computation takes time exponential in ℓ. Lund et al. [46] describe the sum-check protocol that requires far less computation on $\mathcal{V}$'s behalf, but provides a probabilistic guarantee. In the protocol, $\mathcal{V}$ takes as input randomness $r \in \mathbb{F}^\ell$ and interacts with $\mathcal{P}$ over a sequence of ℓ rounds. At the end of this interaction, $\mathcal{V}$ outputs a claim about the evaluation $g(r)$. Let $\langle \mathcal{P}, \mathcal{V}(r) \rangle$ denote the interaction between the prover and verifier with verifier randomness r. We treat $\langle \mathcal{P}, \mathcal{V}(r) \rangle$ as a function that takes prover and verifier input (g, ℓ, d, T) and outputs the claimed evaluation to be checked.

Lemma 5 (The sum-check protocol [46]). *Let g be an ℓ-variate polynomial with degree at most d in each variable. Then, the sumcheck protocol satisfies the following properties.*

1. *Completeness: If $T = \sum_{x \in \{0,1\}^\ell} g(x)$, then for all $r \in \mathbb{F}^\ell$,*

$$\Pr\left[\langle \mathcal{P}, \mathcal{V}(r) \rangle(g, \ell, d, T) = g(r) \right] = 1.$$

2. *Soundness: If $T \neq \sum_{x \in \{0,1\}^\ell} g(x)$, then for any $\mathcal{P}^\star$*

$$\Pr_r\left[\langle \mathcal{P}^\star, \mathcal{V}(r) \rangle(g, \ell, d, T) = g(r) \right] \leq \ell \cdot d / |\mathbb{F}|.$$

3. *Succinctness: The communication cost is $O(\ell \cdot d)$ elements of $\mathbb{F}$.*

Lemma 6 (Sums over evaluations). *Consider size $\ell \in \mathbb{N}$. For multilinear polynomial $P \in \mathbb{F}[X_1, \ldots, X_\ell]$ we have that*

$$P(X) = \sum_{x \in \{0,1\}^\ell} \widetilde{eq}(X, x) \cdot P(x).$$

where $\widetilde{eq}$ is a multilinear extension of eq, which takes as inputs two values in $\{0,1\}^\ell$ returns 1 if its inputs are equal and 0 otherwise.

Proof. Let $Q(X) = \sum_{x \in \{0,1\}^\ell} \widetilde{eq}(X, x) \cdot P(x)$ By the definition of $\widetilde{eq}$, we have that

$$P(x) = Q(x)$$

for all $x \in \{0,1\}^\ell$. However, because $P \in \mathbb{F}[X_1, \ldots, X_\ell]$ is multilinear it is completely determined by 2^ℓ evaluation points. The same holds for Q. Because P and Q agree on 2^ℓ points, they must be the same polynomial. □

Lemma 7 (Schwartz-Zippel [56]). *let $g : \mathbb{F}^\ell \to \mathbb{F}$ be an ℓ-variate polynomial of total degree at most d. Then, on any finite set $S \subseteq \mathbb{F}$,*

$$\Pr_{x \leftarrow S^\ell}[g(x) = 0] \leq d/|S|.$$

A.3 Commitment Schemes

Definition 17 (Commitment Scheme). *A commitment scheme is defined by polynomial-time algorithm $\mathsf{Gen} : \mathbb{N}^2 \to P$ that produces public parameters given the security parameter and size parameter, a deterministic polynomial-time algorithm $\mathsf{Commit} : P \times M \times R \to C$ that produces a commitment in C given a public parameters, message, and randomness tuple such that binding holds. That is, for any PPT adversary $\mathcal{A}$, given $\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda, n)$, and given $((m_1, r_1), (m_2, r_2)) \leftarrow \mathcal{A}(\mathsf{pp})$ we have that*

$$\Pr[(m_1, r_1) \neq (m_2, r_2) \wedge \mathsf{Commit}(\mathsf{pp}, m_1, r_1) = \mathsf{Commit}(\mathsf{pp}, m_2, r_2)] \approx 0.$$

The commitment scheme is deterministic if Commit does not use its randomness.

Definition 18 (Hiding). *The commitment scheme $(\mathsf{Gen}, \mathsf{Commit})$ is hiding if for any PPT adversary $\mathcal{A}$, given $\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda, n)$, $((m_1, r_1), (m_2, r_2)) \leftarrow \mathcal{A}(\mathsf{pp})$, and $C_i \leftarrow \mathsf{Commit}(\mathsf{pp}, m_i, r_i)$ for $i \in \{1, 2\}$ we have that*

$$\Pr[\mathcal{A}(\mathsf{pp}, C_1) = 1] \approx \Pr[\mathcal{A}(\mathsf{pp}, C_2) = 1].$$

Definition 19 (Homomorphic). *The commitment scheme $(\mathsf{Gen}, \mathsf{Commit})$ is homomorphic if the message space M, randomness space R, and commitment space C are groups and for all $n \in \mathbb{N}$, and $\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda, n)$, we have that for any $m_1, m_2 \in M$ and $r_1, r_2 \in R$*

$$\mathsf{Commit}(\mathsf{pp}, m_1, r_1) + \mathsf{Commit}(\mathsf{pp}, m_2, r_2) = \mathsf{Commit}(\mathsf{pp}, m_1 + m_2, r_1 + r_2).$$

Definition 20 (Succinct Commitments). *A commitment scheme $(\mathsf{Gen}, \mathsf{Commit})$, over message space M and commitment space R, provides succinct commitments if for all $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$, and any $m \in M$ and $r \in R$, we have that $|\mathsf{Commit}(\mathsf{pp}, m, r)| = O_\lambda(\mathsf{polylog}(|m|))$.*

Definition 21 (Multilinear Polynomial Commitment Scheme). *A multilinear polynomial commitment scheme over polynomial ring $\mathbb{F}^1[X_1, \ldots, X_n]$ is a*

commitment scheme $(\mathsf{Gen}, \mathsf{Commit})$ *over message space* $\mathbb{F}^1[X_1, \ldots, X_n]$*, equipped with an argument of knowledge (Definition 23) for relation* $\mathcal{R}_{\mathsf{polyeval}}$ *defined as follows*

$$\mathcal{R}_{\mathsf{polyeval}} = \left\{ (\mathsf{pp}, (C, x, y), (P, r)) \middle| \begin{array}{l} P \in \mathbb{F}^1[X_1, \ldots, X_n], \\ P(x) = y, \\ C = \mathsf{Commit}(\mathsf{pp}, P, r) \end{array} \right\}.$$

A.4 Rank-1 constraint satisfiability (R1CS)

R1CS is an NP-complete problem implicit in the work of GGPR [31]. Below, we recall its definition.

Definition 22 (R1CS). *Consider a finite field* $\mathbb{F}$*. Let the public parameters consist of size bounds* $m, n, \ell \in \mathbb{N}$ *where* $m > \ell$*. The R1CS structure consists of sparse matrices* $A, B, C \in \mathbb{F}^{m \times m}$ *with at most* $n = \Omega(m)$ *non-zero entries in each matrix. An instance* $\mathsf{x} \in \mathbb{F}^{\ell}$ *consists of public inputs and outputs and is satisfied by a witness* $W \in \mathbb{F}^{m-\ell-1}$ *if* $(A \cdot Z) \circ (B \cdot Z) = C \cdot Z$*, where* $Z = (W, \mathsf{x}, 1)$*.*

A.5 Arguments of Knowledge

Definition 23 (Argument of Knowledge). *Consider relation* $\mathcal{R}$ *over public parameters, structure, instance, and witness tuples. A reduction of knowledge for* $\mathcal{R}$ *is defined by PPT algorithms* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *and deterministic algorithm* $\mathcal{K}$*, denoting the generator, the prover, the verifier and the encoder respectively with the following interface.*

- $\mathcal{G}(\lambda, N) \to \mathsf{pp}$*: Takes as input security parameter* λ *and size parameters* N*. Outputs public parameters* pp*.*
- $\mathcal{K}(\mathsf{pp}, \mathsf{s}) \to (\mathsf{pk}, \mathsf{vk})$*: Takes as input public parameters* pp *and structure* s*. Outputs prover key* pk *and verifier key* vk
- $\mathcal{P}(\mathsf{pk}, \mathsf{u}, \mathsf{w}) \to \perp$*: Takes as input public parameters* pp*, and an instance-witness pair* (u, w)*. Interactively proves that* $(\mathsf{pp}, \mathsf{s}, \mathsf{u}, \mathsf{w}) \in \mathcal{R}$*.*
- $\mathcal{V}(\mathsf{pk}, \mathsf{u}) \to \{0, 1\}$*: Takes as input public parameters* pp*, and an instance* u*. Interactively checks* u*.*

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ *denote the interaction between* $\mathcal{P}$ *and* $\mathcal{V}$*. We treat* $\langle \mathcal{P}, \mathcal{V} \rangle$ *as a function that takes as input* $((\mathsf{pk}, \mathsf{vk}), \mathsf{u}, \mathsf{w})$ *and runs the interaction on prover input* $(\mathsf{pk}, \mathsf{u}, \mathsf{w})$ *and verifier input* $(\mathsf{pp}, \mathsf{u})$*. At the end of the interaction,* $\langle \mathcal{P}, \mathcal{V} \rangle$ *outputs the verifier's decision. An argument of knowledge* $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ *satisfies the following conditions.*

(i) Completeness: For any PPT *adversary* $\mathcal{A}$*, given* $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, N)$*,* $(\mathsf{s}, \mathsf{u}, \mathsf{w}) \leftarrow \mathcal{A}(\mathsf{pp})$ *such that* $(\mathsf{pp}, \mathsf{s}, \mathsf{u}, \mathsf{w}) \in \mathcal{R}$ *and* $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$ *we have that*

$$\langle \mathcal{P}, \mathcal{V} \rangle ((\mathsf{pk}, \mathsf{vk}), \mathsf{u}, \mathsf{w}) = 1$$

(ii) Knowledge Soundness: For any expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$, there exists an expected polynomial-time extractor $\mathcal{E}$ such that given $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, N)$, $(\mathsf{s}, \mathsf{u}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$, we have that

$$\Pr[(\mathsf{pp}, \mathsf{s}, \mathsf{u}, \mathcal{E}(\mathsf{pp}, \mathsf{u}, \mathsf{st})) \in \mathcal{R}_1] \approx \Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), \mathsf{u}, \mathsf{st}) = 1].$$

Definition 24 (Succinctness). *An argument of knowledge is succinct if the communication complexity and the verifier time complexity is at most poly-logarithmic in the size of the structure and witness.*

Definition 25 (Non-Interactivity). *An argument of knowledge is non-interactive if the interaction consists of a single message from the prover to the verifier. In this case, we denote this single message as the output of the prover, and as an input to the verifier.*

Definition 26 (Zero-knowledge). *An argument of knowledge $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for relation $\mathcal{R}$ satisfies zero-knowledge if for any PPT adversary $\mathcal{V}^*$ there exists an EPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$ for $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N)$, $(\mathsf{s}, (u, w), \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$*

$$\left\{ \mathsf{st}_2 \,\middle|\, \mathsf{st}_2 \leftarrow \langle \mathcal{P}, \mathcal{V}^*(\mathsf{st}_1) \rangle((\mathsf{pk}, \mathsf{vk}), u, w) \right\} \cong \left\{ \mathsf{st}_2 \,\middle|\, \mathsf{st}_2 \leftarrow \mathcal{S}(\mathsf{pp}, \mathsf{s}, u, \mathsf{st}_1) \right\}$$

where st_2 denotes the output of $\mathcal{V}^*$ after interaction. An argument of knowledge satisfies honest-verifier zero-knowledge (HVZK) if it satisifes zero-knowledge under an honest (but curious) verifier that behaves according to the interactive protocol but produces arbitrary output on the side.

A.6 Incrementally Verifiable Computation

Definition 27 (Incrementally verifiable computation (IVC)). *An incrementally verifiable computation (IVC) scheme is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic $\mathcal{K}$ denoting the generator, the prover, the verifier, and the encoder respectively, with the following interface*

- $\mathcal{G}(1^\lambda, N) \to \mathsf{pp}$*: on input security parameter λ and size bounds N, samples public parameters* pp*.*
- $\mathcal{K}(\mathsf{pp}, F) \to (\mathsf{pk}, \mathsf{vk})$*: on input public parameters* pp*, and polynomial-time function F, deterministically produces a prover key* pk *and a verifier key* vk*.*
- $\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \to \Pi_{i+1}$*: on input a prover key* pk*, a counter i, an initial input z_0, a claimed output after i iterations z_i, a non-deterministic advice ω_i, and an IVC proof Π_i attesting to z_i, produces a new proof Π_{i+1} attesting to $z_{i+1} = F(z_i, \omega_i)$.*
- $\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) \to \{0, 1\}$*: on input a verifier key* vk*, a counter i, an initial input z_0, a claimed output after i iterations z_i, and an IVC proof Π_i attesting to z_i, outputs 1 if Π_i is accepting, and 0 otherwise.*

An IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ satisfies the following requirements.

1. *Perfect Completeness: For any PPT adversary $\mathcal{A}$*

$$\Pr\left[\mathcal{V}(\mathsf{vk}, (i+1, z_0, z_{i+1}), \Pi_{i+1}) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N), \\ F, (i, z_0, z_i, \Pi_i) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F), \\ z_{i+1} \leftarrow F(z_i, \omega_i), \\ \mathcal{V}(\mathsf{vk}, i, z_0, z_i, \Pi_i) = 1, \\ \Pi_{i+1} \leftarrow \mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \end{array}\right] = 1$$

   where F is a polynomial-time computable function represented as an arithmetic circuit.

2. *Knowledge Soundness: Consider constant $n \in \mathbb{N}$. For all expected polynomial-time adversaries $\mathcal{P}^*$ there exists an expected polynomial-time extractor $\mathcal{E}$ such that*

$$\Pr_{\mathsf{r}}\left[\begin{array}{l} z_n = z \text{ where} \\ z_{i+1} \leftarrow F(z_i, \omega_i) \\ \forall i \in \{0, \ldots, n-1\} \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N), \\ (F, (z_0, z_i), \Pi) \leftarrow \mathcal{P}^*(\mathsf{pp}, \mathsf{r}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F), \\ \mathcal{V}(\mathsf{vk}, (n, z_0, z), \Pi) = 1, \\ (\omega_0, \ldots, \omega_{n-1}) \leftarrow \mathcal{E}(\mathsf{pp}, \mathsf{r}) \end{array}\right] \approx 1$$

   where r denotes an arbitrarily long random tape. Moreover, F is a polynomial-time computable function represented as an arithmetic circuit.

3. *Succinctness: The size of an IVC proof Π is independent of the number of iterations n.*

B Achieving non-interactivity for multi-folding schemes

Construction 3 (Fiat-Shamir transformation for multi-folding schemes). Consider a public-coin multi-folding scheme $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu, \nu)$ with ℓ rounds. Let ρ denote a random oracle. We construct a non-interactive multi-folding scheme $\Pi' = (\mathcal{G}', \mathcal{K}', \mathcal{P}', \mathcal{V}')$ for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu, \nu)$ in the random oracle model as follows.

- $\mathcal{G}'(1^\lambda, N) \to \mathsf{pp}$: Compute and output $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N)$.
- $\mathcal{K}'(\mathsf{pp}, \mathsf{s}) \to \mathsf{pp}$:
  1. Compute $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$.
  2. Compute $\mathsf{hs} \leftarrow \rho(\mathsf{pp}, \mathsf{s})$.
  3. Output $(\mathsf{pk}', \mathsf{vk}') \leftarrow ((\mathsf{pk}, \mathsf{hs}), (\mathsf{vk}, \mathsf{hs}))$.
- $\mathcal{P}'(\mathsf{pk}', (\vec{u}_1, \vec{u}_2), (\vec{w}_1, \vec{w}_2))$:
  1. Parse pk' as $(\mathsf{pk}, \mathsf{hs})$
  2. Run $\mathcal{P}(\mathsf{pk}, (\vec{u}_1, \vec{u}_2), (\vec{w}_1, \vec{w}_2))$. On the ith message m_i, respond with verifier randomness $r_{i+1} \leftarrow \rho(m_i, r_i)$ where $r_1 = \mathsf{hs}$. Let (u, w) be the output of $\mathcal{P}$ and let $\pi = (m_1, \ldots, m_\ell)$ consist of messages from $\mathcal{P}$.

3. Send π to the verifier.
4. Output (u, w).

- $\mathcal{V}'(\mathsf{vk}', (\vec{u}_1, \vec{u}_2))$:
  1. Parse vk' as $(\mathsf{vk}, \mathsf{hs})$
  2. Receive $\pi = (m_1, \ldots, m_\ell)$ from the prover. Compute $r_{i+1} \leftarrow \rho(m_i, r_i)$ for $r_1 = \mathsf{hs}$.
  3. Run $\mathcal{V}(\mathsf{vk}, (\vec{u}_1, \vec{u}_2))$ with randomness $(r_1, \ldots, r_{\ell+1})$. In round i, send the prover message m_i. Let u be the output of $\mathcal{V}$.
  4. Output u.

The Fiat-Shamir transformation affords Lemma 1.

C Additional related work

Halo [13] and its generalization in [19] propose a way to realize IVC (and proof-carrying data [22,7]) using SNARKs whose verifiers support the so-called accumulation schemes. Halo2 [14] switches the polynomial IOP in Halo [13] from Sonic [48] to Plonk [30]. Unfortunately, it incurs substantial prover costs as the prover must produce a SNARK using Plonk at each step of the program execution. Furthermore, the prover incurs $O(n \cdot d)$ cryptographic operations, where n is the size of the circuit at each step and d is the maximum degree of constraints proven. Switching from Plonk to HyperPlonk [20] would reduce the cryptographic operations to $O(n)$, but it does not avoid the need to produce a SNARK. Split accumulation [18] avoids succinct arguments (e.g., SNARKs) to construct IVC or PCD. Unfortunately, their construction targets R1CS. It is not clear how to extend it to handle Plonkish without making the prover incur $O(n \cdot d)$ cryptographic operations, which, as noted above, is undesirable.

Buffet [68], building on Pantry [16] and Ben-Sasson et al. [5], avoids the high cost of universal circuits yet supports a general class of programs. For example, Buffet supports any program in the C programming language as long as it neither invokes goto statements nor uses function pointers. Furthermore, Buffet provides an "a la carte" cost profile where the prover's proof generation costs are proportional only to the sum of sizes of circuits of the operations invoked by the program execution. However, Buffet adopts a "line-by-line compilation" approach [62,15,54,16], where it unrolls programs into non-uniform circuits by translating each program statement into a concise set of constraints. Unfortunately, this approach requires static bounds on program execution lengths. More importantly, it is unclear how to prove the satisfiability of unrolled non-uniform circuits in an incremental fashion. Furthermore, although general, it is unclear how to use Buffet's approach to prove program executions on a stateful machine *without* producing a non-uniform circuit for each program. Having a separate circuit for each program is undesirable in practice as it is not clear how in that model one program can invoke another program (a la "composability").

A work that follows Buffet, called vRAM [73], achieves Buffet-like costs for program executions on vnTinyRAM [4], a RAM machine with a minimal instruction set. In particular, during program execution, at the granularity of a processor cycle, vRAM uses a "trimmed" version of the vnTinyRAM universal circuit where the trimmed version eliminates circuit elements corresponding to instructions that were not invoked. Unfortunately, like Buffet, this approach is not incremental. Specifically, it requires proving that certain global invariants hold over the entire trace of program execution (e.g., to prove that the trimmed version of the circuit is correct), using randomized fingerprinting techniques. As with Buffet, it is unclear how to prove these global invariants hold in an incremental fashion. Furthermore, this approach reveals, for each program execution, the number of invocations of each instruction supported by the machine to the verifier, so vRAM's approach does not ensure zero-knowledge.

MIRAGE [38] adapts vRAM's techniques in the context of Groth's SNARK [34] (vRAM uses a CMT-based argument [24]). Like vRAM, MIRAGE still relies on proving invariants over the entire execution trace via fingerprinting techniques, making its techniques incompatible with incremental proof systems.

D Details of the zero-knowledge and succinctness layer

In this section, we formally construct and prove HyperNova's zero-knowledge and succinctness layer. Recall that our goal is to design a zero-knowledge argument for the following relation.

$$\mathcal{R}_{\mathsf{VNIVC}} = \left\{ (\mathsf{pp}, (F_1, \ldots, F_\ell, \varphi), (i, z_0, z_i), \Pi) \;\middle|\; \begin{array}{l} \mathsf{vk} \leftarrow \mathsf{NIVC}.\mathcal{K}(\mathsf{pp}, (F_1, \ldots, F_\ell, \varphi)), \\ \mathsf{NIVC}.\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi) = 1 \end{array} \right\}$$

D.1 Building blocks: randomizing and hiding folding schemes

We begin by defining randomizing folding schemes, a central building block for our zero-knowledge layer. We then demonstrate that the Nova folding scheme for relaxed R1CS features a stronger hiding property, which we will additionally leverage in our construction.

Definition 28 (Randomizing). *A multi-folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu, \nu)$ is randomizing if there exists a sampling algorithm $\mathsf{sample}_{\mathcal{R}_1}$ for instance-witness pairs in $\mathcal{R}_1$ such that for any expected polynomial-time adversary $\mathcal{A}$ given $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, N)$ and $((\mathsf{s}_1, \mathsf{s}_2), (\vec{u}_1, \vec{w}_1), (\vec{u}_2, \vec{w}_2)) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = 1$, $(\mathsf{pp}, \mathsf{s}_1, \vec{u}_1, \vec{w}_1) \in \mathcal{R}_1^{\mu-1}$, and $(\mathsf{pp}, \mathsf{s}_2, \vec{u}_2, \vec{w}_2) \in \mathcal{R}_2^{\nu}$ we have that*

$$\left\{ (\mathsf{pp}, \mathsf{s}_1, u, w) \;\middle|\; (u, w) \xleftarrow{\$} \mathsf{sample}_{\mathcal{R}_1}(\mathsf{pp}, \mathsf{s}_1) \right\} \cong$$
$$\left\{ (\mathsf{pp}, \mathsf{s}_1, u, w) \;\middle|\; \begin{array}{l} (u_{\mathsf{r}}, w_{\mathsf{r}}) \xleftarrow{\$} \mathsf{sample}_{\mathcal{R}_1}(\mathsf{pp}, \mathsf{s}_1), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)), \\ (u, w) \leftarrow \langle \mathcal{P}, \mathcal{V} \rangle ((\mathsf{pk}, \mathsf{vk}), ((u_{\mathsf{r}}, \vec{u}_1), \vec{u}_2), ((w_{\mathsf{r}}, \vec{w}_1), \vec{w}_2)) \end{array} \right\}.$$

Lemma 8 (Folding CCS is randomizing). *Construction 1 is randomizing.*

Proof. We begin by describing a sampling algorithm for linearized committed CCS.

$\underline{\mathsf{sample}_{\mathsf{LCCCS}}(\mathsf{pp}, \mathsf{s}) \to (\mathsf{u}, \mathsf{w})}$:

(1) Parse size bounds $t, \ell, s' \in \mathbb{N}$ from s.

(2) Parse matrices $\widetilde{M}_1, \ldots, \widetilde{M}_t$ from the LCCCS structure s.

(3) Randomly sample partial instance $(u, \mathsf{x}, r) \in (\mathbb{F}, \mathbb{F}^\ell, \mathbb{F})$ and witness $\widetilde{w} \in \mathbb{F}^{2^{s'}-1}$.

(4) For $i \in [t]$, compute

$$v_i \leftarrow \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r, y) \cdot \widetilde{z}(y)$$

where $\widetilde{z}$ is an s'-variate multilinear polynomial such that $z(x) = \widetilde{(w, u, \mathsf{x})}(x)$ for all $x \in \{0,1\}^{s'}$.

(5) Sample $r_{\widetilde{w}} \leftarrow \mathbb{F}$ and compute $C \leftarrow \mathsf{Commit}(\mathsf{pp}, \widetilde{w}, r_{\widetilde{w}})$

(6) Compute and output $\mathsf{u} \leftarrow (C, (u, \mathsf{x}, r, v_1, \ldots, v_t))$ and $\mathsf{w} \leftarrow (\widetilde{w}, r_{\widetilde{w}})$.

Now, consider a linearized committed CCS instance-witness pair $(C, (u, \mathsf{x}, r, v_1, \ldots, v_t))$ and $(\widetilde{w}, r_{\widetilde{w}})$ that is folded into an arbitrarily chosen instance-witness pair.

By Step 7 of the multi-folding scheme for CCS, we have that the folded linearized committed CCS instance is computed by taking a random linear combination of all incoming linearized CCS instances (and CCS instances reduced to linearized CCS instances by the sum-check protocol). Thus, we have that the folded terms C, u, and x are indistinguishable from random. Similarly, we have that the folded witness $\widetilde{w}$ is indistinguishable from random. Moreover, by construction of the sum-check protocol, we have the updated random value r'_{x} is also indistinguishable from random. Finally, we have that the terms v_j for $j \in [t]$ are completely determined by the prior values. Therefore, we have that the folded instance-witness pair is indistinguishable from one sampled randomly. □

Construction 4 (A folding scheme for committed relaxed R1CS [43]). Consider a finite field $\mathbb{F}$ and a succinct, hiding, and homomorphic commitment scheme Commit over $\mathbb{F}$. We define the generator and the encoder as follows.

- $\mathcal{G}(1^\lambda, (m, n, \ell \in \mathbb{N})) \to \mathsf{pp}$: output commitment parameters pp_W and pp_E for vectors of size m and $m - \ell - 1$ respectively.
- $\mathcal{K}(\mathsf{pp}, (A, B, C)) \to (\mathsf{pk}, \mathsf{vk})$: output $\mathsf{pk} \leftarrow (\mathsf{pp}, (A, B, C))$ and $\mathsf{vk} \leftarrow \perp$.

The verifier $\mathcal{V}$ takes two committed relaxed R1CS instances $(\overline{E}_1, u_1, \overline{W}_1, \mathsf{x}_1)$ and $(\overline{E}_2, u_2, \overline{W}_2, \mathsf{x}_2)$. The prover $\mathcal{P}$, in addition to the two instances, takes witnesses to both instances, $(E_1, r_{E_1}, W_1, r_{W_1})$ and $(E_2, r_{E_2}, W_2, r_{W_2})$. Let $Z_1 = (W_1, \mathsf{x}_1, u_1)$ and $Z_2 = (W_2, \mathsf{x}_2, u_2)$. The prover and the verifier proceed as follows.

1. $\mathcal{P}$: Send $\overline{T} \coloneqq \mathsf{Commit}(\mathsf{pp}_E, T, r_T)$, where $r_T \xleftarrow{\$} \mathbb{F}$ and
$$T = AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1 - u_1 \cdot CZ_2 - u_2 \cdot CZ_1.$$
2. $\mathcal{V}$: Sample and send a challenge $r \xleftarrow{\$} \mathbb{F}$.
3. $\mathcal{V}, \mathcal{P}$: Output the folded instance $(\overline{E}, u, \overline{W}, \mathsf{x})$, where
$$\begin{aligned} \overline{E} &\leftarrow \overline{E}_1 + r \cdot \overline{T} + r^2 \cdot \overline{E}_2 \\ u &\leftarrow u_1 + r \cdot u_2 \\ \overline{W} &\leftarrow \overline{W}_1 + r \cdot \overline{W}_2 \\ \mathsf{x} &\leftarrow \mathsf{x}_1 + r \cdot \mathsf{x}_2 \end{aligned}$$
4. $\mathcal{P}$: Output the folded witness (E, r_E, W, r_W), where
$$\begin{aligned} E &\leftarrow E_1 + r \cdot T + r^2 \cdot E_2 \\ r_E &\leftarrow r_{E_1} + r \cdot r_T + r^2 \cdot r_{E_2} \\ W &\leftarrow W_1 + r \cdot W_2 \\ r_W &\leftarrow r_{W_1} + r \cdot r_{W_2} \end{aligned}$$

Construction 5 (Sampling randomized relaxed R1CS). We now provide an algorithm to sample a randomized relaxed R1CS instance-witness pairs.

$\underline{\mathsf{sample}_{\mathsf{RR1CS}}(\mathsf{pp}, \mathsf{s}) \to (\mathsf{u}, \mathsf{w})}$:

1. Parse s as matrices (A, B, C).
2. Sample a random $(W, \mathsf{x}, u, r_E, r_W)$;
3. Compute $E \leftarrow AZ \circ BZ - u \cdot CZ$, where $Z = (W, \mathsf{x}, u)$;
4. Compute $(\overline{E}, \overline{W}) \leftarrow (\mathsf{Commit}(\mathsf{pp}, E, r_E), \mathsf{Commit}(\mathsf{pp}, W, r_W))$, where r_E and r_W are sampled randomly; and
5. Output the instance $(\overline{E}, u, \overline{W}, \mathsf{x})$ and witness (E, r_E, W, r_W).

Lemma 9 (Folding R1CS is hiding). *Consider the folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for relaxed R1CS from Construction 4. Then, for any honest-but-curious deterministic $\mathcal{V}^*$, there exists an EPT simulator $\mathcal{S}$ such that for all PPT adversary*

$\mathcal{A}$ *for* $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N)$, $(\mathsf{s}, \mathsf{u}_1, \mathsf{w}_1) \leftarrow \mathcal{A}(\mathsf{pp})$ *such that* $(\mathsf{pp}, \mathsf{s}, \mathsf{u}_1, \mathsf{w}_1) \in \mathcal{R}_{\mathsf{RR1CS}}$*, and* $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$

$$\left\{ (\mathsf{st}_2, \mathsf{w}_2) \,\middle|\, \begin{array}{l} (\mathsf{u}_{\mathsf{rb}}, \mathsf{w}_{\mathsf{rb}}) \xleftarrow{\$} \mathsf{sample}_{\mathsf{RR1CS}}(\mathsf{pp}, \mathsf{s}) \\ (\mathsf{st}_2, \mathsf{w}_2) \leftarrow \langle \mathcal{P}, \mathcal{V}^* \rangle ((\mathsf{pk}, \mathsf{vk}), (\mathsf{u}_1, \mathsf{u}_{\mathsf{rb}}), (\mathsf{w}_1, \mathsf{w}_{\mathsf{rb}})) \end{array} \right\}$$
$$\cong$$
$$\left\{ (\mathsf{st}_2, \mathsf{w}_2) \,\middle|\, (\mathsf{st}_2, \mathsf{w}_2) \leftarrow \mathcal{S}(\mathsf{pp}, \mathsf{s}, \mathsf{u}_1) \right\}.$$

where st_2 *represents the (arbitrary) output of* $\mathcal{V}^*$*.*

Proof. Consider an honest-but-curious PPT adversary $\mathcal{V}^*$. To prove hiding, we construct an EPT simulator $\mathcal{S}$ that simulates the joint distribution of the verifier's output and the prover's output witness as follows.

$\underline{\mathcal{S}(\mathsf{pp}, (A, B, C), (\overline{E}, u, \overline{W}, \mathsf{x})) \to ((\overline{E}', u', \overline{W}', \mathsf{x}'), (E', r_{E'}, W', r_{W'}), \pi)}$:

1. Sample the folded instance-witness pair $((\overline{E}', u', \overline{W}', \mathsf{x}'), (E', r_{E'}, W', r_{W'}))$.
2. Sample the verifier's challenge $r \xleftarrow{\$} \mathbb{F}$
3. Sample a uniformly random $\overline{T}$.
4. Solve for the prover's first message $u_{\mathsf{blind}} = (\overline{E}_{\mathsf{blind}}, u_{\mathsf{blind}}, \overline{W}_{\mathsf{blind}}, \mathsf{x}_{\mathsf{blind}})$:
   - $\overline{E}_{\mathsf{blind}} \leftarrow r^{-2} \cdot (\overline{E}' - r \cdot \overline{T} - \overline{W})$
   - $u_{\mathsf{blind}} \leftarrow r^{-1} \cdot (u' - u)$
   - $\overline{W}_{\mathsf{blind}} \leftarrow r^{-1} \cdot (\overline{W}' - \overline{W})$
   - $\mathsf{x}_{\mathsf{blind}} \leftarrow r^{-1} \cdot (\mathsf{x}' - \mathsf{x})$
5. Run the verifier $\mathcal{V}^*$ on input vk, and instances $(\overline{E}, u, \overline{W}, \mathsf{x})$ and $(\overline{E}_{\mathsf{blind}}, u_{\mathsf{blind}}, \overline{W}_{\mathsf{blind}}, \mathsf{x}_{\mathsf{blind}})$. Let st' be the output of $\mathcal{V}^*$. Instantiate the verifier randomness to r and send the first message $\overline{T}$.
6. Output st' and $(E', r_{E'}, W', r_{W'})$

We now argue that the simulator (i.e. the ideal setting) produces an output that is indistinguishable from the prover and the verifier output (i.e. the real setting). Indeed, consider an arbitrary adversary $\mathcal{A}$. Suppose that

$$\begin{array}{r} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N) \\ ((A, B, C), (\overline{E}, u, \overline{W}, \mathsf{x}), (E, r_E, W, r_W)) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (A, B, C)) \end{array}$$

In both the real and ideal settings, because the verifier $\mathcal{V}^*$ interacts honestly with the prover, the challenge r sampled by $\mathcal{V}^*$ in the real setting is indistinguishable from the challenge r sampled by $\mathcal{S}$ in the ideal setting.

Moreover, in both the real and ideal settings, $\overline{T}$ is a hiding commitment, so $\overline{T}$ is indistinguishable in the real and ideal settings.

In the real setting, $(\overline{E}_{\mathsf{blind}}, u_{\mathsf{blind}}, \overline{W}_{\mathsf{blind}}, \mathsf{x}_{\mathsf{blind}})$ is randomly sampled from the blinding distribution. And, in the ideal setting, $(\overline{E}_{\mathsf{blind}}, u_{\mathsf{blind}}, \overline{W}_{\mathsf{blind}}, \mathsf{x}_{\mathsf{blind}})$ is indistinguishable from random as it is computed from the randomly sampled $(\overline{E}', u', \overline{W}', \mathsf{x}')$, $\overline{T}$, and r. So, they are indistinguishable.

In the honest setting, by construction, we have that

$$\begin{aligned} W' &\leftarrow W + r \cdot W_{\mathsf{blind}} \\ \mathsf{x}' &\leftarrow \mathsf{x} + r \cdot \mathsf{x}_{\mathsf{blind}} \\ u' &\leftarrow u + r \cdot u_{\mathsf{blind}} \end{aligned}$$

Because W_{blind}, $\mathsf{x}_{\mathsf{blind}}$, and u_{blind} are uniformly random, we have that W', x', and u' are uniformly random. Moreover, r'_E and r'_W are computed as follows.

$$\begin{aligned} r'_E &\leftarrow r_E + r \cdot r_T + r^2 \cdot r_{E_{\mathsf{blind}}} \\ r'_W &\leftarrow r_W + r \cdot r_{W_{\mathsf{blind}}} \end{aligned}$$

By the same argument as above, we have that r'_E and r'_W are uniformly random.

Then, we have that E', $\overline{E}'$, and $\overline{W}'$ are completely determined by the prior values. Therefore, because W', x', u', r'_E and r'_W are also randomly sampled in the ideal setting, we have that the folded instance $(\overline{E}', u', \overline{W}', \mathsf{x}')$ and the corresponding witness $(E', r_{E'}, W', r_{W'})$ are indistinguishable in the real and ideal settings.

This implies that the view of the verifier $\mathcal{V}^*$ is indistinguishable in both the real and ideal setting. Therefore, the output st' is indistinguishable in both the real and ideal setting.

Putting everything together, we have that the simulator's output is indistinguishable from that of the interaction between an honest prover and $\mathcal{V}^*$. □

D.2 Core construction

We now formally define a zero-knowledge argument for $\mathcal{R}_{\mathsf{VNIVC}}$.

Construction 6 (A zero-knowledge argument for $\mathcal{R}_{\mathsf{VNIVC}}$). We construct a zero-knowledge argument of a valid HyperNova proof. Let NIFS be the non-interactive multi-folding scheme underlying HyperNova for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \nu, \mu)$ that satisfies randomization and where $\mathcal{R}_2$ is a committed relation with respect to a hiding commitment scheme. Let $\mathsf{sample}_{\mathcal{R}_1}$ be the corresponding sampling algorithm for NIFS guaranteed by the randomization property. Let $\mathsf{sample}_{\mathsf{RR1CS}}$ be the sampling algorithm corresponding to relaxed R1CS (Construction 5). Let Nova be the zero-knowledge non-interactive folding scheme for relaxed R1CS.

We first construct a blinding circuit blind that takes as input the non-interactive folding scheme's verifier key $\mathsf{vk}_{\mathsf{NIFS}}$, the NIVC statement (i, z_0, z_i), and as non-deterministic input a list of running instances U, an instance u, an index pc, a

folding proof π for u, random instances U_r, and the corresponding folding proofs $(\pi_1, \ldots, \pi_\ell)$. It outputs an updated list of running instances U'.

$\underline{\mathsf{blind}((\mathsf{vk}_{\mathsf{NIFS}}, (i, z_0, z_i)); (\mathsf{U}, \mathsf{u}, \mathsf{pc}, \pi), \mathsf{U}_\mathsf{r}, (\pi_1, \ldots, \pi_\ell)) \to \mathsf{U}'}$:

1. Parse u as (C, u'), i.e., the part that commits to the witness and the remainder.
2. Check that $\mathsf{u}' = \mathsf{enc}_{\mathsf{inst}}(\mathsf{hash}(\mathsf{vk}_{\mathsf{NIFS}}, i, z_0, z_i, \mathsf{U}, \mathsf{pc}))$.
3. Check that $1 \leq \mathsf{pc} \leq \ell$.
4. Fold in the fresh instance: $\mathsf{U}[\mathsf{pc}] \leftarrow \mathsf{NIFS}.\mathcal{V}(\mathsf{vk}_{\mathsf{NIFS}}[\mathsf{pc}], \mathsf{U}[\mathsf{pc}], \mathsf{u}, \pi)$.
5. For $j \in [\ell]$, compute $\mathsf{U}'[j] \leftarrow \mathsf{NIFS}.\mathcal{V}(\mathsf{vk}_{\mathsf{NIFS}}[j], \mathsf{U}[j], \mathsf{U}_\mathsf{r}[j], \pi_j)$
6. Output U'.

We define a zero-knowledge argument of knowledge $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ as follows.

$\underline{\mathcal{G}(1^\lambda, N) \to \mathsf{pp}}$:

1. Output $(\mathsf{pp}_{\mathsf{NIVC}}, \mathsf{pp}_{\mathsf{Nova}}) \leftarrow (\mathsf{NIFS}.\mathcal{G}(1^\lambda, N), \mathsf{Nova}.\mathcal{G}(1^\lambda, N))$.

$\underline{\mathcal{K}(\mathsf{pp}, (\varphi, (F_1, \ldots, F_\ell))) \to (\mathsf{pk}, \mathsf{vk})}$:

1. Compute $(\mathsf{pk}_{\mathsf{NIFS}}, \mathsf{vk}_{\mathsf{NIFS}}) \leftarrow \mathsf{NIFS}.\mathcal{K}(\mathsf{pp}_{\mathsf{NIFS}}, (\varphi, (F_1, \ldots, F_\ell)))$.
2. Compute $\mathsf{s}_{\mathsf{blind}} \leftarrow \mathsf{enc}_{\mathsf{str}}(\mathsf{blind})$.
3. Compute $(\mathsf{pk}_{\mathsf{Nova}}, \mathsf{vk}_{\mathsf{Nova}}) \leftarrow \mathsf{Nova}.\mathcal{K}(\mathsf{pp}_{\mathsf{Nova}}, \mathsf{s}_{\mathsf{blind}})$
4. Compute and output

$$\mathsf{vk} \leftarrow (\mathsf{pp}, (\mathsf{pk}_{\mathsf{NIFS}}, \mathsf{vk}_{\mathsf{NIFS}}), (\mathsf{pk}_{\mathsf{Nova}}, \mathsf{vk}_{\mathsf{Nova}}))$$
$$\mathsf{pk} \leftarrow \mathsf{vk}$$

$\underline{\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \mathit{\Pi})}$:

1. If $i = 0$: Output $\perp$.
2. Parse $\mathit{\Pi}$ as $((\mathsf{U}, \mathsf{W}), (\mathsf{u}, \mathsf{w}), \mathsf{pc})$.
3. Update $(\mathsf{U}[\mathsf{pc}], \mathsf{W}[\mathsf{pc}], \pi) \leftarrow \mathsf{NIFS}.\mathcal{P}(\mathsf{pk}_{\mathsf{NIFS}}[\mathsf{pc}], (\mathsf{U}[\mathsf{pc}], \mathsf{W}[\mathsf{pc}]), (\mathsf{u}, \mathsf{w}))$
4. Sample ℓ randomized running instance-witness pairs $(\mathsf{U}_\mathsf{r}, \mathsf{W}_\mathsf{r})$ in $\mathcal{R}_1$ with respect to the structures corresponding to $F'_1, \ldots, F'_\ell$.
5. For $j \in [\ell]$, compute

$$(\mathsf{U}'[j], \mathsf{W}'[j]), \pi_j \leftarrow \mathsf{NIFS}.\mathcal{P}(\mathsf{pk}_{\mathsf{NIFS}}[j], (\mathsf{U}[j], \mathsf{U}_\mathsf{r}[j]), (\mathsf{W}[j], \mathsf{W}_\mathsf{r}[j]))$$

6. Compute a relaxed R1CS instance-witness pair corresponding to the execution of blind

$$(\mathsf{u}^{\mathsf{partial}}_{\mathsf{blind}}, \mathsf{w}^{\mathsf{partial}}_{\mathsf{blind}}) \leftarrow \mathsf{enc}(\mathsf{blind}, ((\mathsf{vk}_{\mathsf{NIFS}}, (i, z_0, z_i)), \mathsf{U}'), (\mathsf{vk}_{\mathsf{NIFS}}, (i, z_0, z_i), (\mathsf{U}, \mathsf{u}, \mathsf{pc}, \pi), \mathsf{U}_\mathsf{r}, (\pi_1, \ldots, \pi_\ell)))$$

7. Sample randomness r and let $\mathsf{w}_{\mathsf{blind}} \leftarrow (\mathsf{w}_{\mathsf{blind}}^{\mathsf{partial}}, r)$ compute the committed instance
$$\mathsf{u}_{\mathsf{blind}} \leftarrow (\mathsf{Commit}(\mathsf{pp}, \mathsf{w}_{\mathsf{blind}}), \mathsf{u}_{\mathsf{blind}}^{\mathsf{partial}}).$$
8. Sample a randomized committed relaxed R1CS instance-witness pair $(\mathsf{u}_{\mathsf{rb}}, \mathsf{w}_{\mathsf{rb}})$ with respect to the structure corresponding to blind.
9. Send $\mathsf{u}_{\mathsf{blind}}$ and u_{rb} to the verifier.
10. Interactively randomize the instance-witness pair corresponding to the execution of blind
$$(\mathsf{u}'_{\mathsf{blind}}, \mathsf{w}'_{\mathsf{blind}}) \leftarrow \mathsf{Nova}.\mathcal{P}(\mathsf{pk}_{\mathsf{Nova}}, (\mathsf{u}_{\mathsf{blind}}, \mathsf{w}_{\mathsf{blind}}), (\mathsf{u}_{\mathsf{rb}}, \mathsf{w}_{\mathsf{rb}})).$$
11. Send $(\mathsf{w}'_{\mathsf{blind}}, \mathsf{W}')$ to the verifier.

$\underline{\mathcal{V}(\mathsf{vk}, (i, z_0, z_i)) \to \{0,1\}}$:

1. If $i = 0$: Output 1 if $z_0 = z_i$ and 0 otherwise.
2. Receive $\mathsf{u}_{\mathsf{blind}}$ and u_{rb} from the prover.
3. Check that $\mathsf{enc}_{\mathsf{inst}}^{-1}(\mathsf{u}_{\mathsf{blind}})$ references $\mathsf{vk}_{\mathsf{NIFS}}$ and (i, z_0, z_i).
4. Parse U' from $\mathsf{enc}_{\mathsf{inst}}^{-1}(\mathsf{u}_{\mathsf{blind}})$.
5. Interactively randomize $\mathsf{u}_{\mathsf{blind}}$:
$$\mathsf{u}'_{\mathsf{blind}} \leftarrow \mathsf{Nova}.\mathcal{V}(\mathsf{vk}_{\mathsf{Nova}}, \mathsf{u}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{rb}})$$
6. Receive $(\mathsf{w}'_{\mathsf{blind}}, \mathsf{W}')$ from the prover.
7. Check the randomized instance attests to the correct execution of blind
$$(\mathsf{pp}, \mathsf{s}_{\mathsf{blind}}, \mathsf{u}'_{\mathsf{blind}}, \mathsf{w}'_{\mathsf{blind}}) \in \mathcal{R}_{\mathsf{RR1CS}}$$
where $\mathsf{s}_{\mathsf{blind}}$ is the structure corresponding to blind.
8. For $j \in \ell$, check that
$$(\mathsf{pp}, \mathsf{s}_{F'_j}, \mathsf{U}'[j], \mathsf{W}'[j]) \in \mathcal{R}_1$$
where $\mathsf{s}_{F'_j}$ is the structure corresponding to F'_j.

D.3 Proof of properties

We now prove that Construction 6 is a zero-knowledge argument of knowledge. In particular, we prove the following theorem.

Theorem 5 (A zero-knowledge argument for $\mathcal{R}_{\mathsf{VNIVC}}$). *Construction 6 is an honest-verifier zero-knowledge argument of knowledge for $\mathcal{R}_{\mathsf{VNIVC}}$.*

The proof of Theorem 5 can be decomposed into the following lemmas. We sketch the proof for completeness and knowledge soundness, and give a full proof for honest-verifier zero-knowledge.

Lemma 10. *Construction 6 satisfies completeness and knowledge soundness.*

Proof (Sketch). The completeness and knowledge soundness of Construction 6 follows from the completeness and knowledge soundness of the underlying folding schemes. In particular, the proof proceeds similarly to that of Theorem 4 which also has the prover demonstrate the correct execution of the folding verifier, by demonstrating the knowledge of a valid instance-witness pair for a circuit containing the folding verifier (albeit recursively). □

Lemma 11 (Zero-Knowledge). *Construction 6 satisfies honest-verifier zero-knowledge.*

Proof. To prove honest-verifier zero-knowledge, we must construct a simulator (without access to the witness) that can produce a computationally equivalent transcript to that of an honest prover and a verifier. At a high-level, we observe that all running instances are randomized by blind and thus can be simulated. The instance $\mathsf{u}_{\mathsf{blind}}$ attesting to the execution of blind is itself randomized by Nova, and thus the corresponding proof of valid randomization and randomized witness can be simulated due to the hiding property of the Nova folding scheme.

Indeed, consider an honest-but-curious PPT adversary $\mathcal{V}^*$. We begin by constructing an honest-but-curious PPT adversary $\mathcal{V}^*_{\mathsf{Nova}}$ for the underlying Nova folding scheme.

$\underline{\mathcal{V}^*_{\mathsf{Nova}}(\mathsf{vk}_{\mathsf{Nova}}, \mathsf{u}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{rb}}) \to \mathsf{st}_2}$

1. Compute
$$\mathsf{u}'_{\mathsf{blind}} \leftarrow \mathsf{Nova}.\mathcal{V}(\mathsf{vk}_{\mathsf{Nova}}, \mathsf{u}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{rb}})$$
and record the transcript as π_{blind}.
2. Output $(\mathsf{vk}, \mathsf{u}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{rb}}, \mathsf{u}'_{\mathsf{blind}}, \pi_{\mathsf{blind}})$.

Given $\mathcal{V}^*_{\mathsf{novafold}}$, let $\mathcal{S}_{\mathsf{Nova}}$ be the corresponding hiding simulator for Nova guaranteed by Lemma 9. We construct the desired simulator as follows.

$\underline{\mathcal{S}(\mathsf{pp}, (F_1, \ldots, F_\ell, \varphi), (i, z_0, z_i)) \to \pi}$:

1. Compute the prover and verifier keys
$$(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (F_1, \ldots, F_\ell, \varphi))$$
2. Compute the structure corresponding to the blind circuit:
$$\mathsf{s}_{\mathsf{blind}} \leftarrow \mathsf{enc}_{\mathsf{str}}(\mathsf{blind}).$$

3. Simulate the running instance-witness pairs $(\mathsf{U}', \mathsf{W}')$ with respect to the structures corresponding to $F'_1, \ldots, F'_\ell$:

$$(\mathsf{U}'[j], \mathsf{W}'[j]) \leftarrow \mathsf{sample}_{\mathcal{R}_1}(\mathsf{pp}, \mathsf{enc}_{\mathsf{str}}(F'_j)).$$

4. Simulate a relaxed R1CS instance corresponding to the blinding step that outputs the randomized running instances U':

$$\mathsf{u}_{\mathsf{blind}}^{\mathsf{partial}} \leftarrow \mathsf{enc}_{\mathsf{inst}}(((\mathsf{vk}_{\mathsf{NIFS}}, (i, z_0, z_i)), \mathsf{U}')).$$

5. Sample commitment randomness r and simulate the full instance:

$$\mathsf{u}_{\mathsf{blind}} \leftarrow (\mathsf{Commit}(\mathsf{pp}, \perp, r), \mathsf{u}_{\mathsf{blind}}^{\mathsf{partial}}).$$

6. Simulate the randomized instance-witness pair attesting to the correct execution of blind and the corresponding proof of correct randomization:

$$((\mathsf{u}_{\mathsf{rb}}, \mathsf{u}'_{\mathsf{blind}}, \pi_{\mathsf{blind}}), \mathsf{w}'_{\mathsf{blind}}) \leftarrow \mathcal{S}_{\mathsf{Nova}}(\mathsf{pp}, \mathsf{s}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{blind}}).$$

7. Run the $\mathcal{V}^*(\mathsf{vk}, (i, z_0, z_i))$ with randomness instantiated to the randomness found in the verifier's message in π_{blind}:
   (a) Send $(\mathsf{u}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{rb}})$ as the first message to $\mathcal{V}^*$.
   (b) Send prover's message in π_{blind} as the second message to $\mathcal{V}^*$.
   (c) Send $\mathsf{w}'_{\mathsf{blind}}$ and W' as the third message to $\mathcal{V}^*$.
   (d) Let st_2 be the output of $\mathcal{V}^*$.
8. Output st_2.

We now demonstrate that the simulated proof is computationally equivalent to that of an honest interaction. Consider an arbitrary PPT adversary $\mathcal{A}$. Consider public parameters $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N)$ and consider the following adversarially chosen structure, instance, and witness pair:

$$((F_1, \ldots, F_\ell, \varphi), (i, z_0, z_i), \mathit{\Pi}) \leftarrow \mathcal{A}(\mathsf{pp})$$

Consider the prover and verifier keys $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{NIVC}.\mathcal{K}(\mathsf{pp}, (F_1, \ldots, F_\ell, \varphi))$. Suppose that

$$\mathsf{NIVC}.\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \mathit{\Pi}) = 1.$$

Now, consider the following distribution, which represents the result of an honest interaction.

$$\left\{ \mathsf{st}_2 \;\middle|\; \begin{array}{l} (\mathsf{u}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{rb}}) \leftarrow \mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \mathit{\Pi}), \\ r \leftarrow \mathcal{V}^*(\mathsf{vk}, (i, z_0, z_i), (\mathsf{u}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{rb}})) \\ \overline{T}, (\mathsf{w}'_{\mathsf{blind}}, \mathsf{W}') \leftarrow \mathcal{P}(r) \\ \mathsf{st}_2 \leftarrow \mathcal{V}^*(\overline{T}, (\mathsf{w}'_{\mathsf{blind}}, \mathsf{W}')) \end{array} \right\}$$

By construction, we have that $\mathcal{P}$ first randomly samples $(\mathsf{u}_{\mathsf{rb}}, \mathsf{w}_{\mathsf{rb}})$ and then folds it into $(\mathsf{u}_{\mathsf{blind}}, \mathsf{w}_{\mathsf{blind}})$ to produce the randomized instance-witness pair $(\mathsf{u}'_{\mathsf{blind}}, \mathsf{w}'_{\mathsf{blind}})$ and the corresponding interactive proof $\pi_{\mathsf{blind}} = (r, \overline{T})$. Moreover, by construction of Nova, The terms u_{rb}, $\mathsf{u}'_{\mathsf{blind}}$, π_{blind}, and $\mathsf{w}'_{\mathsf{blind}}$ are uncorrelated with U' and W'. Then, by the hiding property of the Nova folding scheme (Lemma 9) and by the construction of $\mathcal{V}^*_{\mathsf{Nova}}$, we have that $\mathcal{S}_{\mathsf{Nova}}$ can simulate $((\mathsf{u}_{\mathsf{rb}}, \mathsf{u}'_{\mathsf{blind}}, \pi_{\mathsf{blind}}), \mathsf{w}'_{\mathsf{blind}})$ with respect to the same verifier key $\mathsf{vk}_{\mathsf{Nova}}$ and instance $\mathsf{u}_{\mathsf{blind}}$. Therefore, the prior distribution is computationally equivalent to the following distribution.

$$\left\{ \mathsf{st}_2 \;\middle|\; \begin{array}{l} \mathsf{u}_{\mathsf{blind}} \leftarrow \mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \mathit{\Pi}), \\ \mathsf{s}_{\mathsf{blind}} \leftarrow \mathsf{enc}_{\mathsf{str}}(\mathsf{blind}), \\ (\mathsf{u}_{\mathsf{rb}}, (r, \overline{T}), \mathsf{w}'_{\mathsf{blind}}) \leftarrow \mathcal{S}_{\mathsf{Nova}}(\mathsf{pp}, \mathsf{s}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{blind}}), \\ r \leftarrow \mathcal{V}^*(\mathsf{vk}, (i, z_0, z_i), (\mathsf{u}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{rb}})) \\ \mathsf{W}' \leftarrow \mathcal{P}(r) \\ \mathsf{st}_2 \leftarrow \mathcal{V}^*(\overline{T}, (\mathsf{w}'_{\mathsf{blind}}, \mathsf{W}')) \end{array} \right\}.$$

Next, we have that $\mathsf{u}^{\mathsf{partial}}_{\mathsf{blind}}$ is completely determined by $\mathsf{vk}_{\mathsf{NIFS}}$, (i, z_0, z_i), and U'. Then, because $\mathsf{u}_{\mathsf{blind}}$ only additionally contains a hiding commitment, we have have that it can be simulated by randomly sampling a commitment. Crucially, even though $\mathsf{u}_{\mathsf{blind}}$ is no longer guaranteed to be satisfying, we have that $\mathcal{S}_{\mathsf{Nova}}$ cannot distinguish this fact, and thus performs as if $\mathsf{u}_{\mathsf{blind}}$ is satisfying. In particular, letting U' be the result of parsing the internal state of $\mathcal{P}$, we have that the following distribution is computationally equivalent to the former distribution.

$$\left\{ \mathsf{st}_2 \;\middle|\; \begin{array}{l} \mathsf{U}' \leftarrow \mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \mathit{\Pi}), \\ \mathsf{u}^{\mathsf{partial}}_{\mathsf{blind}} \leftarrow \mathsf{enc}_{\mathsf{inst}}(((\mathsf{vk}_{\mathsf{NIFS}}, (i, z_0, z_i)), \mathsf{U}')), \\ \mathsf{u}_{\mathsf{blind}} \leftarrow (\mathsf{Commit}(\mathsf{pp}, \bot, r), \mathsf{u}^{\mathsf{partial}}_{\mathsf{blind}}), \\ \mathsf{s}_{\mathsf{blind}} \leftarrow \mathsf{enc}_{\mathsf{str}}(\mathsf{blind}), \\ (\mathsf{u}_{\mathsf{rb}}, (r, \overline{T}), \mathsf{w}'_{\mathsf{blind}}) \leftarrow \mathcal{S}_{\mathsf{Nova}}(\mathsf{pp}, \mathsf{s}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{blind}}), \\ r \leftarrow \mathcal{V}^*(\mathsf{vk}, (i, z_0, z_i), (\mathsf{u}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{rb}})) \\ \mathsf{W}' \leftarrow \mathcal{P}(r) \\ \mathsf{st}_2 \leftarrow \mathcal{V}^*(\overline{T}, (\mathsf{w}'_{\mathsf{blind}}, \mathsf{W}')) \end{array} \right\}.$$

Moreover, by the randomization property of the underlying multi-folding scheme, we have that $(\mathsf{U}', \mathsf{W}')$ produced by the prover is indistinguishable from one an instance-witness pair sampled randomly. Therefore, we have that the prior distribution is equivalent to the following distribution.

$$\left\{ \mathsf{st}_2 \;\middle|\; \begin{array}{l} (\mathsf{U}', \mathsf{W}') \text{ sampled randomly}, \\ \mathsf{u}^{\mathsf{partial}}_{\mathsf{blind}} \leftarrow \mathsf{enc}_{\mathsf{inst}}(((\mathsf{vk}_{\mathsf{NIFS}}, (i, z_0, z_i)), \mathsf{U}')), \\ \mathsf{u}_{\mathsf{blind}} \leftarrow (\mathsf{Commit}(\mathsf{pp}, \bot, r), \mathsf{u}^{\mathsf{partial}}_{\mathsf{blind}}), \\ \mathsf{s}_{\mathsf{blind}} \leftarrow \mathsf{enc}_{\mathsf{str}}(\mathsf{blind}), \\ (\mathsf{u}_{\mathsf{rb}}, (r, \overline{T}), \mathsf{w}'_{\mathsf{blind}}) \leftarrow \mathcal{S}_{\mathsf{Nova}}(\mathsf{pp}, \mathsf{s}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{blind}}), \\ r \leftarrow \mathcal{V}^*(\mathsf{vk}, (i, z_0, z_i), (\mathsf{u}_{\mathsf{blind}}, \mathsf{u}_{\mathsf{rb}})) \\ \mathsf{st}_2 \leftarrow \mathcal{V}^*(\overline{T}, (\mathsf{w}'_{\mathsf{blind}}, \mathsf{W}')) \end{array} \right\}.$$

However, the above distribution precisely follows the construction of the simulator. Therefore, we have that that the above distribution is computationally equivalent

to the following distribution.

$$\left\{ \mathsf{st}_2 \,\middle|\, \mathsf{st}_2 \leftarrow \mathcal{S}(\mathsf{pp}, (F_1, \ldots, F_\ell, \varphi), (i, z_0, z_i)) \right\}.$$

Thus, we have demonstrated that the output of the simulator is computationally equivalent to that of an honest (but curious) interaction. □

D.4 Specializing the zero-knowledge layer for Nova's IVC proofs

Suppose that we are interested in randomizing Nova's IVC proof as opposed to an NIVC proof, we can substantially simplify the zero-knowledge layer. In particular, we can avoid the blind circuit as there is no need to hide pc in the case of Nova's IVC proofs. We now provide a sketch of this simplification.

The folding scheme used below is Nova (Construction 4), which provides the required hiding property (Lemma 9). Given a Nova IVC proof $\mathit{\Pi} = (\mathsf{U}_i, \mathsf{u}_i, \mathsf{W}_i, \mathsf{w}_i)$ proving a statement (i, z_0, z_i) with a verifier key of vk, the prover produces a randomized IVC proof $\mathit{\Pi}'$ with the following steps.

1. Fold the instance-witness pairs $(\mathsf{U}_i, \mathsf{W}_i)$ with $(\mathsf{u}_i, \mathsf{w}_i)$ to get a folded instance-witness pair $(\mathsf{U}_f, \mathsf{W}_f)$ and a folding proof π.
2. Randomly sample a satisfying relaxed R1CS instance-witness pair $(\mathsf{U}_r, \mathsf{W}_r)$.
3. Fold the instance-witness pairs $(\mathsf{U}_f, \mathsf{W}_f)$ with $(\mathsf{U}_r, \mathsf{W}_r)$ to get a folded instance-witness pair $(\mathsf{U}'_i, \mathsf{W}'_i)$ and a folding proof π'.
4. Output the randomized IVC proof as $\mathit{\Pi}' = (\mathsf{U}_i, \mathsf{u}_i, \mathsf{U}_r, \pi, \pi', \mathsf{W}'_i)$.

Given a randomized Nova IVC proof $\mathit{\Pi}' = (\mathsf{U}_i, \mathsf{u}_i, \mathsf{U}_r, \pi, \pi', \mathsf{W}'_i)$, for an IVC statement (i, z_0, z_i) with a verifier key of vk, the verifier proceeds as follows.

1. If $i = 0$, check that $z_0 = z_i$.
2. Otherwise:

- Check that $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i)$.
- Check that $(\mathsf{u}_i.\overline{E}, \mathsf{u}_i.u) = (\mathsf{u}_\perp.\overline{E}, 1)$.
- Fold the instance U_i with u_i using π to get a folded instance U_f.
- Fold the instance U_f with U_r using π' to get a folded instance U'_i.
- Check that W'_i is a satisfying witness to U'_i.

E Details of HyperNova over a cycle of elliptic curves

Let $(\mathbb{G}_1, \mathbb{G}_2)$ denote a 2-cycle of elliptic curves, where each curve in the cycle can be used as cryptographic group (i.e. the discrete logarithm problem is hard). Let $\mathbb{F}_p$ and $\mathbb{F}_q$ respectively denote the scalar field and the base field of $\mathbb{G}_1$. Naturally, $\mathbb{F}_q$ and $\mathbb{F}_p$ respectively denote the scalar field and the base field of $\mathbb{G}_2$.

Suppose $\mathcal{R}_{\mathsf{LCCCS}}$ and $\mathcal{R}_{\mathsf{CCCS}}$ are both defined over $\mathbb{F}_p$ (i.e., the scalar field of $\mathbb{G}_1$) and $\mathcal{R}_{\mathsf{CRR1CS}}$ is defined over $\mathbb{F}_q$ (i.e., the scalar field of $\mathbb{G}_2$). We provide a

multi-folding scheme for

$$(\mathcal{R}_1 = \mathcal{R}_{\mathsf{LCCCS}} \times \mathcal{R}_{\mathsf{CRR1CS}}, \mathcal{R}_2 = \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu = 1, \nu = 1)$$

where the compat predicate is defined below.

Overview. Our goal is to modify the folding scheme for CCS (Construction 1) such that it can be efficiently instantiated on a cycle of elliptic curves. For simplicity, we describe and prove the case of $\mu = \nu = 1$. However, both the construction and proofs naturally generalize to the case of arbitrary values of μ and ν. In particular, the generalized version simply uses additional powers of a random challenge when combining claims (as in Construction 1).

Suppose that the prover and the verifier are given as input a tuple consisting of a linearized committed CCS instance and a committed relaxed R1CS instance $(\mathsf{U}_{\mathsf{LCCCS}}, \mathsf{U}_{\mathsf{CRR1CS}})$, and a committed CCS instance $\mathsf{u}_{\mathsf{CCCS}}$. The prover additionally takes as input witnesses $(\mathsf{W}_{\mathsf{LCCCS}}, \mathsf{W}_{\mathsf{CRR1CS}})$ and $\mathsf{w}_{\mathsf{CCCS}}$.

The original folding scheme verifier folds the committed CCS instance $\mathsf{u}_{\mathsf{CCCS}}$ into the the linearized committed CCS instance $\mathsf{U}_{\mathsf{LCCCS}}$ to produce a new linearized committed CCS instance $\mathsf{U}'_{\mathsf{LCCCS}}$. Internally, this involves finite field and hash operations. In addition, it involves one scalar multiplication and point addition. In particular, provided commitment C_1 in the linearized committed CCS instance $\mathsf{U}_{\mathsf{LCCCS}}$ and commitment C_2 in the committed CCS instance $\mathsf{u}_{\mathsf{CCCS}}$, the HyperNova verifier, picks a random challenge ρ, and computes in Step 7

$$C' \leftarrow C_1 + \rho \cdot C_2.$$

Unfortunately, this computation makes it inefficient to represent the original verifier over the same curve that represents the computations that it verifies. To address this, we modify the original verifier to take the resulting value C' as non-deterministic advice. Of course, this advice must be verified.

To do so, the prover generates a relaxed R1CS instance that represents the random linear combination during the original folding protocol. In more detail, let $\mathsf{s}_{\mathsf{EC}} = (A, B, C)$ denote a committed relaxed R1CS structure defined over F_q. Its public IO consists of (ρ, C_1, C_2, C'), where $\rho \in \mathbb{F}_p, C_1 \in \mathbb{G}_1, C_2 \in \mathbb{G}_1, C' \in \mathbb{G}_1$. This constraint system enforces that $C' = C_1 + \rho \cdot C_2$, where $+$ is the elliptic curve point addition and $\cdot$ is the elliptic curve scalar multiplication operation in $\mathbb{G}_1$. Since F_q is the base field of $\mathbb{G}_1$, s_{EC} computes the required point addition and scalar multiplication operations "natively" with a concise set of constraints (i.e., without the "wrong field" arithmetic).

We modify the original verifier to read the inputs and outputs of this relaxed R1CS instance (rather than computing the random linear combination itself). Instead of directly checking this instance, it is folded into a running relaxed R1CS instance using the folding scheme underlying Nova [43]. Note that this auxiliary computation is represented on the second curve in the cycle. Thus, the Nova verifier can be natively represented over the first curve alongside the rest of the original verifier.

Putting everything together, we achieve a folding scheme that takes a committed CCS instance and folds it into a linearized CCS instance and a relaxed R1CS instance to produce a new linearized CCS instance and a relaxed R1CS instance.

Construction 7 (A multi-folding scheme for CCS over cycles). We construct a multi-folding scheme for $(\mathcal{R}_1 = \mathcal{R}_{\mathsf{LCCCS}} \times \mathcal{R}_{\mathsf{CRR1CS}}, \mathcal{R}_2 = \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu = 1, \nu = 1)$, where compat is defined as follows.

$\underline{\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) \to \{\mathsf{true}, \mathsf{false}\}}$

1. Parse s_1 as $(\mathsf{s}_{\mathsf{LCCCS}}, \mathsf{s}_{\mathsf{RR1CS}})$
2. Check that $\mathsf{s}_{\mathsf{LCCCS}} = \mathsf{s}_2$ and $\mathsf{s}_{\mathsf{RR1CS}} = \mathsf{s}_{\mathsf{EC}}$

Let $\mathsf{PC} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ denote an additively-homomorphic polynomial commitment scheme for multilinear polynomials over $\mathbb{F}_p$. Let $\mathsf{VC} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open})$ denote an additively-homomorphic commitment scheme with succinct commitments for vectors over $\mathbb{F}_q$.

We define the generator and the encoder as follows.

$\underline{\mathcal{G}(1^\lambda, (m, N, \ell, t, q, d \in \mathbb{N})) \to \mathsf{pp}}$:

1. Let $n = 2 \cdot (\ell + 1)$
2. $\mathsf{pp}_{\mathsf{PC}} \leftarrow \mathsf{PC.Gen}(1^\lambda, \log n - 1)$
3. $\mathsf{pp}_{\mathsf{VC}} \leftarrow \mathsf{VC.Gen}(1^\lambda, |\mathsf{s}_{\mathsf{EC}}|)$, where $|\mathsf{s}_{\mathsf{EC}}|$ is the maximum among the number of constraints or the number of witness variables in s_{EC}.
4. Output $(m, n, N, \ell, t, q, d, |\mathsf{s}_{\mathsf{EC}}|, \mathsf{pp}_{\mathsf{PC}}, \mathsf{pp}_{\mathsf{VC}})$

$\underline{\mathcal{K}(\mathsf{pp}, (([\widetilde{M}_1, \ldots, \widetilde{M}_t], [S_1, \ldots, S_q], [c_1, \ldots, c_q])), (A, B, C)) \to (\mathsf{pk}, \mathsf{vk})}$:

1. $\mathsf{pk} \leftarrow (\mathsf{pp}, (([\widetilde{M}_1, \ldots, \widetilde{M}_t], [S_1, \ldots, S_q], [c_1, \ldots, c_q])), (A, B, C))$
2. $\mathsf{vk} \leftarrow \perp$
3. Output $(\mathsf{pk}, \mathsf{vk})$

The verifier $\mathcal{V}$ takes a tuple consisting of a linearized committed CCS instance and a committed relaxed R1CS instance $(\mathsf{U}_{\mathsf{LCCCS}}, \mathsf{U}_{\mathsf{CRR1CS}})$, where $\mathsf{U}_{\mathsf{LCCCS}} = (C_1, u, \mathsf{x}_1, r_x, v_1, \ldots, v_t)$ and $\mathsf{U}_{\mathsf{CRR1CS}} = (\overline{E}_1, u_1, \overline{W}_1, x_1)$, and a committed CCS instance $\mathsf{u}_{\mathsf{CCCS}} = (C_2, \mathsf{x}_2)$. The prover $\mathcal{P}$, in addition to these instances, takes witnesses to all instances, $\mathsf{W}_{\mathsf{LCCCS}} = \widetilde{w_1}$, $\mathsf{W}_{\mathsf{CRR1CS}} = (E_1, W_1)$, and $\mathsf{w}_{\mathsf{CCCS}} = \widetilde{w_2}$.

Let $s = \log m$ and $s' = \log n$. Let $\widetilde{z_1} = \widetilde{(w_1, u, \mathsf{x}_1)}$ and $\widetilde{z_2} = \widetilde{(w_2, 1, \mathsf{x}_2)}$.

The prover and the verifier proceed as follows.

1. $\mathcal{V} \to \mathcal{P}$: $\mathcal{V}$ samples $\gamma \xleftarrow{\$} \mathbb{F}_p, \beta \xleftarrow{\$} \mathbb{F}_p^s$, and sends them to $\mathcal{P}$.
2. $\mathcal{V}$: Sample $r'_x \xleftarrow{\$} \mathbb{F}_p^s$.

3. $\mathcal{V} \leftrightarrow \mathcal{P}$: Run the sum-check protocol $c \leftarrow \langle \mathcal{P}, \mathcal{V}(r'_x) \rangle (g, s, d+1, \sum_{j \in [t]} \gamma^j \cdot v_j)$, where:

$$g(x) \coloneqq \left(\sum_{j \in [t]} \gamma^j \cdot L_j(x) \right) + \gamma^{t+1} \cdot Q(x)$$

$$L_j(x) \coloneqq \widetilde{eq}(r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(x, y) \cdot \widetilde{z}_1(y) \right)$$

$$Q(x) \coloneqq \widetilde{eq}(\beta, x) \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(x, y) \cdot \widetilde{z}_2(y) \right) \right)$$

4. $\mathcal{P} \rightarrow \mathcal{V}$: $((\sigma_1, \ldots, \sigma_t), (\theta_1, \ldots, \theta_t))$, where for all $i \in [t]$:

$$\sigma_i = \sum_{y \in \{0,1\}^{s'}} \widetilde{M_i}(r'_x, y) \cdot \widetilde{z}_1(y)$$

$$\theta_i = \sum_{y \in \{0,1\}^{s'}} \widetilde{M_i}(r'_x, y) \cdot \widetilde{z}_2(y)$$

5. $\mathcal{V}$: Compute $e_1 \leftarrow \widetilde{eq}(r_x, r'_x)$ and $e_2 \leftarrow \widetilde{eq}(\beta, r'_x)$, and check that

$$c = \left(\sum_{j \in [t]} \gamma^j \cdot e_1 \cdot \sigma_j + \gamma^{t+1} \cdot e_2 \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \theta_j \right) \right)$$

6. $\mathcal{V} \rightarrow \mathcal{P}$: $\mathcal{V}$ samples $\rho \xleftarrow{\$} \mathbb{F}_p$ and sends it to $\mathcal{P}$.

7. $\mathcal{P} \rightarrow \mathcal{V}$: $\mathcal{P}$ computes a committed relaxed R1CS instance $\mathsf{u}_{\mathsf{CRR1CS}} = (\overline{E}_2, u_2, \overline{W}_2, x_2)$ with structure s_{EC} and witness $\mathsf{w}_{\mathsf{CRR1CS}} = (E_2, W_2)$ to compute the quantity $C_1 + \rho \cdot C_2$, such that the following hold: (1) $u_2 = 1$, (2) $\overline{E}_1 = \overline{0}$, and (3) $x_2 = (\rho, C_1, C_2, C')$ for some $C' \in \mathbb{G}_1$. $\mathcal{P}$ then sends $\mathsf{u}_{\mathsf{CRR1CS}}$ to $\mathcal{V}$.

8. $\mathcal{V}$: Check that $\overline{E}_2 = \overline{0}$, $u_2 = 1$, and $x_2 = (\rho, C_1, C_2, C')$ for some $C' \in \mathbb{G}_1$.

9. $\mathcal{P} \rightarrow \mathcal{V}$: Send $\overline{T} = \mathsf{VC.Commit}(\mathsf{pp}_{\mathsf{VC}}, T)$, where $T = AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1 - u_1 \cdot CZ_2 - u_2 \cdot CZ_1$, $Z_1 = (W_1, x_1, u_1)$, and $Z_2 = (W_2, x_2, u_2)$.

10. $V \rightarrow \mathcal{P}$: $\mathcal{V}$ samples $\rho^\star \xleftarrow{\$} \mathbb{F}_p$ and sends it to $\mathcal{P}$.

11. $\mathcal{V}, \mathcal{P}$: Output the folded linearized committed CCS instance $(C', u', \mathsf{x}', r'_x, v'_1, \ldots, v'_t)$ and the folded committed relaxed R1CS instance $(\overline{E}^\star, u^\star, \overline{W}^\star, x^\star)$, where for

all $i \in [t]$:

$$\begin{aligned}
u' &\leftarrow u + \rho \cdot 1 \\
\mathsf{x}' &\leftarrow \mathsf{x}_1 + \rho \cdot \mathsf{x}_2 \\
v'_i &\leftarrow \sigma_i + \rho \cdot \theta_i \\
\overline{E}^\star &\leftarrow \overline{E}_1 + \rho^\star \cdot \overline{T} \\
u^\star &\leftarrow u_1 + \rho^\star \cdot 1 \\
\overline{W}^\star &\leftarrow \overline{W}_1 + \rho^\star \cdot \overline{W}_2 \\
x^\star &\leftarrow x_1 + \rho^\star \cdot x_2
\end{aligned}$$

12. $\mathcal{P}$: Output the folded witnesses $\mathsf{W}_{\mathsf{LCCCS}} = \widetilde{w}' \leftarrow \widetilde{w}_1 + \rho \cdot \widetilde{w}_2$ and $\mathsf{W}_{\mathsf{CRR1CS}} = (E^\star, W^\star)$, where $E^\star \leftarrow E_1 + \rho^\star \cdot T$ and $W^\star \leftarrow W_1 + \rho^\star \cdot W_2$.

We recall Theorem 4 below. We prove correctness in Appendix H.4.

Theorem 6 (A multi-folding scheme for CCS over cycles). *Construction 7 is a public-coin multi-folding scheme for* $(\mathcal{R}_1 = \mathcal{R}_{\mathsf{LCCCS}} \times \mathcal{R}_{\mathsf{CRR1CS}}, \mathcal{R}_2 = \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu = 1, \nu = 1)$*. with perfect completeness and knowledge soundness.*

Assumption 2 (Non-interactivity). There exists a non-interactive multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}} \times \mathcal{R}_{\mathsf{CRR1CS}}, \mathcal{R}_{\mathsf{CCCS}}, 1, 1)$ in the plain model.

Justification. By applying the Fiat-Shamir transformation (Construction 3) to the multi-folding scheme in Construction 7, we obtain a non-interactive multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}} \times \mathcal{R}_{\mathsf{CRR1CS}}, \mathcal{R}_{\mathsf{CCCS}}, 1, 1)$ in the random oracle model. By instantiating the random oracle with an appropriate cryptographic hash function, we heuristically obtain a non-interactive multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}} \times \mathcal{R}_{\mathsf{CRR1CS}}, \mathcal{R}_{\mathsf{CCCS}}, 1, 1)$ in the plain model. □

Given the above multi-folding scheme, we prove that it is IVC-compatible.

Lemma 12 (IVC-compatibility). *The non-interactive multi-folding scheme for* $(\mathcal{R}_{\mathsf{LCCCS}} \times \mathcal{R}_{\mathsf{CRR1CS}}, \mathcal{R}_{\mathsf{CCCS}}, 1, 1)$ *(Construction 7, Assumption 2) is IVC-compatible.*

Proof (Intuition). NP-completeness, partial functions, and monotonicity for $\mathcal{R}_{\mathsf{CCS}}$, the underlying relation of $\mathcal{R}_{\mathsf{CCCS}}$, holds by Lemma 3. To show the default instances property, we must show that $\mathcal{R}_{\mathsf{LCCCS}}$ and $\mathcal{R}_{\mathsf{CRR1CS}}$ both have default instances. The former requirement holds by Lemma 3, and the latter requirement holds due to Kothapalli et al. [43]. □

We recall Theorem 2 below.

Theorem 7 (HyperNova over cycles). *Given the multi-folding scheme in Construction 7 instantiated with the Pedersen commitment scheme, Construction 2 produces an IVC scheme such that for step functions F_j for $j \in [\ell]$ that can be expressed in CCS with m_j constraints of degree d and q_j monomials, n_j witness variables, t_j CCS matrices, and N_j non-zero entries in CCS matrices, and control function φ that can be expressed in CCS with m constraints of degree d and q_φ monomials, n_φ witness variables, t_φ CCS matrices, and N_φ non-zero entries in the CCS matrices, the efficiency characteristics are as follows.*

- *The NIVC prover time for each step is a single MSM of size $O(n_\varphi + n_j)$ and $O((N_\varphi + N_j) + (t_\varphi + t_j) \cdot (m_\varphi + m_j) + (q_\varphi + q_j) \cdot (m_\varphi + m_j) \cdot d \cdot \log^2 d)$ finite field operations*
- *The size of the verifier circuit is $o(|\varphi| + 2 \cdot \mathsf{G} + (d \cdot \log m_j) \cdot \mathsf{F} + \log m_j \cdot \mathsf{R}_d + 2 \cdot \mathsf{H}_{\ell,t_j} + 2 \cdot \mathsf{M})$ on the first curve and G on the second curve in a cycle of elliptic curves*

where G is the number of constraints required to encode a group scalar multiplication natively (i.e., without field emulation), H is the number of constraints required to encode a hash function, F is the number of constraints to encode field operations, R is the number of constraints to encode a cryptographic hash function used for randomness, and M is the number of constraints to encode to memory read/write over a memory of size $O(\ell)$.

Proof (Intuition). This follows from [43, Lemma 4] and Theorem 3. □

F nlookup: A lookup argument for HyperNova

This section describes a lookup argument, which we refer to as nlookup, that is suitable for use in recursive arguments such as Nova, HyperNova, and others.

Suppose that there is a table T of size n. Now consider m variables $v_1, \ldots, v_m$ in a CCS instance and we wish to enforce that those values are contained in T.

A classic approach is to store T as a Merkle tree for which the circuit gets as public input a commitment. Then to prove that a certain value is in T, the prover could supply as non-deterministic advice to the circuit a Merkle proof of inclusion, and the circuit verifies the Merkle proof of inclusion. This unfortunately requires $O(m \cdot \log n)$ hash evaluations inside the circuit, which is prohibitive. Plookup [29] provides an approach where the number of constraints is $O(\max(m, n))$, which is acceptable when $m \approx n$. It is unsuitable in the context of recursive SNARKs such as Nova where a particular recursive step may perform $m << n$ lookup operations. A recent flurry of works (e.g., see cq [26] for the latest in this line of work) consider the case where $m << n$, but it is unclear how to adapt them to the setting of recursive SNARKs without incurring high recursion overheads.

We provide a conceptually simple and yet efficient lookup argument, that we refer to as nlookup. For m lookups on a table of size n entries, nlookup requires $O(m \log n)$ multiplications and $O(\log n)$ hash operations inside a circuit (with

small constants) and the prover performs $O(n)$ finite field operations. In particular, the prover does not commit to any additional polynomials. This lookup argument is *not* suitable for accelerating bitwise operations in the circuit model of computation, but it is a perfect tool for expressing finite state machines efficiently with Nova and HyperNova (e.g., see [63, §2.4]).

nlookup in a nutshell. Without loss of generality, assume that $n = 2^\ell$. We can view T as a function from $\{0,1\}^\ell \to \mathbb{F}$. Furthermore, let $\widetilde{T}$ denote the unique multilinear extension of the function T. In other words, $\widetilde{T}$ is a multilinear polynomial in ℓ variables where the entries in the table are evaluations of $\widetilde{T}$ over the Boolean hypercube $\{0,1\}^\ell$. To prove m lookup operations, the prover specifies m evaluation points $q_1, \ldots, q_m$ over the Boolean hypercube such that $\widetilde{T}(q_i) = v_i$ for all $i \in [m]$. This requires $O(m \log n)$ Booleanity checks in the circuit to ensure that $q_i \in \{0,1\}^\ell$ for all $i \in [m]$.

We now devise a multi-folding scheme where the prover and the verifier fold the task of checking the correctness of m lookup operations into task of checking an evaluation of $\widetilde{T}$ at a single point in its domain. Furthermore, in our context, the circuit maintains a running claim about an evaluation of $\widetilde{T}$ and the folding scheme folds incoming lookup claims into this running claim.

Suppose that the running claim is $\widetilde{T}(q_r) \stackrel{?}{=} v_r$ for some $q_r \in \mathbb{F}^{\log n}$ and $v_r \in \mathbb{F}$. At initialization, q_r can be arbitrary and $v_r \leftarrow \widetilde{T}(q_r)$. Now, the folding scheme reduces the following claim to an evaluation of $\widetilde{T}$, where $\rho \in \mathbb{F}$ is picked by the verifier at random.

$$v_r + \sum_{i=\{1,\ldots,m\}} \rho^i \cdot v_i \stackrel{?}{=} \sum_{j \in \{0,1\}^{\log n}} \widetilde{eq}(q_r, j) \cdot \widetilde{T}(j) + \sum_{i=\{1,\ldots,m\}} \rho^i \cdot \sum_{j \in \{0,1\}^{\log n}} \widetilde{eq}(q_i, j) \cdot \widetilde{T}(j)$$

The folding scheme applies the sum-check protocol and outputs a new claim about $\widetilde{T}(q'_r) \stackrel{?}{=} v'_r$. The prover's work in the folding scheme is $O(n)$ finite field operations. The verifier's work in the non-interactive folding scheme is $O(\log n)$ hash and field operations. Furthermore, at the end of the sum-check protocol (i.e., inside the folding scheme), the verifier computes evaluations of m $\widetilde{eq}$ polynomials at a random point, this takes $O(m \cdot \log n)$ multiplications.

F.1 Details and security proofs

Definition 29 (Polynomial Evaluation Relation). *We define the polynomial evaluation relation $\mathcal{R}_{\mathsf{poly}}$ as follows. Let the public parameters consist of size parameter $\ell \in \mathbb{N}$. An $\mathcal{R}_{\mathsf{poly}}$ structure consists of $\widetilde{T}$, a multilinear polynomial in ℓ variables. An $\mathcal{R}_{\mathsf{poly}}$ instance is $(r, v) \in (\mathbb{F}^\ell, \mathbb{F})$ where r is an evaluation point and*

v is a claimed evaluation. An $\mathcal{R}_{\mathsf{poly}}$ witness is $\perp$. We define $\mathcal{R}_{\mathsf{poly}}$ as follows.

$$\mathcal{R}_{\mathsf{poly}} = \left\{ ((\ell, \widetilde{T}), (r, v), \perp) \;\middle|\; \begin{array}{l} \ell \in \mathbb{N}, \widetilde{T} \in \mathbb{F}^1[X_1, \ldots, X_\ell], (r, v) \in (\mathbb{F}^\ell, \mathbb{F}) \\ \widetilde{T}(r) = v \end{array} \right\}.$$

Definition 30 (Lookup Relation). *We define the lookup relation $\mathcal{R}_{\mathsf{lookup}}$ as follows. Let the public parameters consist of size parameter $\ell \in \mathbb{N}$. For vector $T \in \mathbb{F}^n$ (where $n = 2^\ell$), an $\mathcal{R}_{\mathsf{lookup}}$ structure consists of the corresponding multilinear extension in ℓ variables, $\widetilde{T}$. An $\mathcal{R}_{\mathsf{lookup}}$ instance consists of value $v \in \mathbb{F}$. An $\mathcal{R}_{\mathsf{lookup}}$ witness consists of index $q \in \{0,1\}^\ell$. We define $\mathcal{R}_{\mathsf{lookup}}$ as follows.*

$$\mathcal{R}_{\mathsf{lookup}} = \left\{ ((\ell, \widetilde{T}), v, q) \;\middle|\; \begin{array}{l} \ell \in \mathbb{N}, \widetilde{T} \in \mathbb{F}^1[X_1, \ldots, X_\ell], v \in \mathbb{F}, q \in \{0,1\}^\ell \\ \widetilde{T}(q) = v \end{array} \right\}.$$

We now provide a multi-folding between two relations, a polynomial evaluation instance and a collection of lookup instances.

Construction 8 (A multi-folding scheme for lookup instances). We construct a multi-folding scheme for $(\mathcal{R}_{\mathsf{poly}}, \mathcal{R}_{\mathsf{lookup}}, \mathsf{compat}, \mu = 1, \nu)$ for arbitrary $\nu \in \mathbb{N}$.

$\underline{\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) \to \{\mathsf{true}, \mathsf{false}\}}$

1. If $\mathsf{s}_1 = \mathsf{s}_2$, then return true, otherwise, return false.

We define the generator and the encoder as follows.

- $\mathcal{G}(1^\lambda, N) \to \mathsf{pp}$:
  1. Sample size bound $\ell \in \mathbb{N}$.
  2. Output $\mathsf{pp} = \ell$.
- $\mathcal{K}(\mathsf{pp} = \ell, \widetilde{T} \in \mathbb{F}^1[X_1, \ldots, X_\ell]) \to (\mathsf{pk}, \mathsf{vk})$: Output $(\mathsf{pk}, \mathsf{vk}) = ((\mathsf{pp}, \widetilde{T}), \mathsf{pp})$.

The prover takes as input $\mathsf{pk} = (\mathsf{pp} = \ell, \widetilde{T})$ and the verifier take as input $\mathsf{vk} = \mathsf{pp} = \ell$. The verifier $\mathcal{V}$ takes a polynomial evaluation instance (q_r, v_r) and a vector of lookup instances $(v_1, \ldots, v_m)$ The prover $\mathcal{P}$, in addition to the instances, takes witnesses to the lookup instances $(q_1, \ldots, q_m)$.

The prover and the verifier proceed as follows.

1. $\mathcal{P} \to \mathcal{V}$: $(q_1, \ldots, q_m)$.
2. $\mathcal{V}$: Check that for all $i \in [m]$, $q_i \in \{0,1\}^\ell$.
3. $\mathcal{V} \to \mathcal{P}$: $\mathcal{V}$ samples $\rho \xleftarrow{\$} \mathbb{F}$ and send it to $\mathcal{P}$.
4. $\mathcal{V}$: Sample $q'_r \xleftarrow{\$} \mathbb{F}^s$.

5. $\mathcal{V} \leftrightarrow \mathcal{P}$: Run the sum-check protocol $c \leftarrow \langle \mathcal{P}, \mathcal{V}(q'_r) \rangle (g, \ell, 2, v_r + \sum_{i \in [m]} \rho^i \cdot v_i)$, where:

$$g(x) \coloneqq \widetilde{eq}(q_r, x) \cdot \widetilde{T}(x) + \sum_{i \in [m]} \rho^i \cdot \widetilde{eq}(q_i, x) \cdot \widetilde{T}(x)$$

6. $\mathcal{P} \to \mathcal{V}$: v'_r, where $v'_r = \widetilde{T}(q'_r)$.
7. $\mathcal{V}$: Compute $e \leftarrow \widetilde{eq}(q_r, q'_r)$ and $e_i \leftarrow \widetilde{eq}(q_i, q'_r)$ for all $i \in [m]$. Abort if

$$c \neq e \cdot v'_r + \sum_{i \in [m]} \rho^i \cdot e_i \cdot v'_r.$$

8. $\mathcal{V}, \mathcal{P}$: Output the folded polynomial evaluation instance (q'_r, v'_r).

Theorem 8 (nlookup). *Construction 8 is a public-coin multi-folding scheme for* $(\mathcal{R}_{\mathsf{poly}}, \mathcal{R}_{\mathsf{lookup}}, \mathsf{compat}, \mu = 1, \nu)$ *for arbitrary* $\nu \in \mathbb{N}$.

Proof (Intuition). Completeness and knowledge soundness holds by the completeness and soundness of the sumcheck protocol. We provide a formal proof in Appendix H.5. □

G Building HyperNova with a black-box use of Nova

We design a step circuit for Nova that runs the verifier's logic in any IVC-compatible non-interactive multi-folding scheme. The step circuit is encoded with R1CS (a popular NP-complete constraint system [31]) and proven incrementally with Nova, but the step circuit is only in charge of running the verifier of the non-interactive multi-folding scheme, in addition to simple bookkeeping. As a result, this provides an IVC scheme, where each step of the incremental computation is expressed with any NP-complete language that has an IVC-compatible multi-folding scheme. Furthermore, we achieve this with a black box use of an IVC scheme for R1CS.

In Nova, each step circuit takes as input the output of the previous step and produces the output for the current step. In HyperNova, besides the application's IO, we augment them with the latest running instance. At each recursive step, the step circuit gets as non-deterministic input a purported instance u in $\mathcal{R}_{\mathsf{CCCS}}$ and π, where π is the prover's output in the non-interactive multi-folding scheme. The step circuit checks that the public input of u matches the application's input provided to the step circuit. If so, it runs the verifier of the non-interactive folding scheme on $(\mathsf{vk}, \mathsf{U}, \mathsf{u}, \pi)$, where vk is the verifier's key and U is the latest running instance passed from the prior step. It then provides uses the public output of u and the output of the folding scheme verifier to construct the step's output.

Construction 9 (A step circuit for Nova). Let NIFS be an IVC-compatible non-interactive multi-folding scheme for $(\mathcal{R}_1, \mathcal{R}_2, 1, 1)$. Let IVC denote the Nova's IVC scheme for functions expressed as R1CS constraints.

We first define a non-deterministic polynomial-time function step, represented as an R1CS structure, that iteratively folds instances expressed in $\mathcal{R}_{\mathsf{CCCS}}$.

$\underline{\mathsf{step}(\mathsf{vk}, \mathsf{U}_i, z_i; (\mathsf{u}, \pi)) \to (\mathsf{vk}, \mathsf{U}_{i+1}, z_{i+1})}$

1. Parse $(\mathsf{in}, \mathsf{out}) \leftarrow \mathsf{enc}^{-1}_{\mathsf{inst}}(\mathsf{u}')$, where u' represents the portion of u that does not contain commitments to the witness.
2. Check that $\mathsf{in} = z_i$
3. Compute $\mathsf{U}_{i+1} \leftarrow \mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}, \pi)$
4. Output $(\mathsf{vk}, \mathsf{U}_{i+1}, \mathsf{out})$

Given F, we define the corresponding IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$, which uses Nova in a black-box manner.

$\underline{\mathcal{G}(1^\lambda, N) \to \mathsf{pp}}$: Output $(\mathsf{NIFS}.\mathcal{G}(1^\lambda, N), \mathsf{IVC}.\mathcal{G}(1^\lambda, N))$

$\underline{\mathcal{K}((\mathsf{pp}_{\mathsf{NIFS}}, \mathsf{pp}_{\mathsf{IVC}}), F) \to (\mathsf{pk}, \mathsf{vk})}$:

1. Compute $(\mathsf{s}_1, \mathsf{s}_2) \leftarrow \mathsf{enc}_{\mathsf{str}}(F)$
2. Compute $(\mathsf{pk}_{\mathsf{NIFS}}, \mathsf{vk}_{\mathsf{NIFS}}) \leftarrow \mathsf{NIFS.K}(\mathsf{pp}_{\mathsf{NIFS}}, \mathsf{s}_1, \mathsf{s}_2)$
3. Compute $(\mathsf{pk}_{\mathsf{IVC}}, \mathsf{vk}_{\mathsf{IVC}}) \leftarrow \mathsf{IVC.K}(\mathsf{pp}_{\mathsf{IVC}}, \mathsf{step})$
4. Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((F, \mathsf{pk}_{\mathsf{NIFS}}, \mathsf{vk}_{\mathsf{NIFS}}, \mathsf{pk}_{\mathsf{IVC}}), (\mathsf{step}, \mathsf{pp}_{\mathsf{NIFS}}, \mathsf{vk}_{\mathsf{NIFS}}, \mathsf{vk}_{\mathsf{IVC}}))$.

$\underline{\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \mathit{\Pi}_i) \to \mathit{\Pi}_{i+1}}$:

1. Parse $\mathit{\Pi}_i$ as $(\mathit{\Pi}'_i, \mathsf{U}_i, \mathsf{W}_i)$
2. Let $z_{i+1} \leftarrow F(z_i, \omega_i)$ and compute $(\mathsf{u}_i, \mathsf{w}_i) \leftarrow \mathsf{enc}(F, (z_i, z_{i+1}), \omega_i)$.
3. Compute $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi_{i+1}) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}_{\mathsf{NIFS}}, (\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$
4. Compute $\mathit{\Pi}'_{i+1} \leftarrow \mathsf{IVC.P}(\mathsf{pk}_{\mathsf{IVC}}, i, (\mathsf{vk}_{\mathsf{NIFS}}, \mathsf{u}_\perp, z_0), (\mathsf{vk}_{\mathsf{NIFS}}, \mathsf{U}_i, z_i), (\mathsf{u}_i, \pi_{i+1}), \mathit{\Pi}'_i)$
5. Output $\mathit{\Pi}_{i+1} = (\mathit{\Pi}'_{i+1}, \mathsf{U}_{i+1}, \mathsf{W}_{i+1})$

$\underline{\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \mathit{\Pi}_i) \to \{0, 1\}}$:

1. Parse $\mathit{\Pi}_i$ as $(\mathit{\Pi}'_i, \mathsf{U}_i, \mathsf{W}_i)$.
2. Check that $\mathsf{IVC.V}(\mathsf{vk}_{\mathsf{IVC}}, i, (\mathsf{vk}_{\mathsf{NIFS}}, \mathsf{u}_\perp, z_0), (\mathsf{vk}_{\mathsf{NIFS}}, \mathsf{U}_i, z_i), \mathit{\Pi}'_i) = 1$
3. Check that $(\mathsf{pp}_{\mathsf{NIFS}}, \mathsf{step}, \mathsf{U}_i, \mathsf{W}_i) \in \mathcal{R}_{\mathsf{LCCCS}}$

Theorem 9 (A simple construction for IVC). *Construction 9 is an IVC scheme.*

H Deferred Proofs

H.1 Proof of Theorem 1 (A multi-folding scheme for CCS)

Lemma 13 (Perfect Completeness). *Construction 1 satisfies perfect completeness.*

Proof. Consider the public parameters $\mathsf{pp} = (m, n, N, \ell, t, q, d, \mathsf{pp}_{\mathsf{PC}}) \leftarrow \mathcal{G}(1^\lambda, N)$ and let $s = \log m$ and $s' = \log n$.

Consider arbitrary structures $(\mathsf{s}_1, \mathsf{s}_2)$, where $\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = \mathsf{true}$

$$\mathsf{s}_1 = \mathsf{s}_2 = (\widetilde{M}_1, \ldots, \widetilde{M}_t), (S_1, \ldots, S_q), (c_1, \ldots, c_q) \leftarrow \mathcal{A}(\mathsf{pp}).$$

Consider the prover and verifier keys $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2))$. Suppose that the prover and the verifier are provided μ linearized committed CCS instances $\vec{\mathsf{u}}_1$ and ν committed CCS instances $\vec{\mathsf{u}}_2$. Suppose that the prover additionally is provided with the corresponding satisfying witnesses $\vec{\mathsf{w}}_1$ and $\vec{\mathsf{w}}_2$.

As in the construction, let $s = \log m$ and $s' = \log n$. Let $\widetilde{z}_{1,k} = \widetilde{(w, u, \mathsf{x})}$, where $w = \mathcal{L}_k.w$, $u = \mathcal{L}_k.\phi.u$, and $\mathsf{x} = \mathcal{L}_k.\phi.\mathsf{x}$. Similarly, let $\widetilde{z}_{2,k} = \widetilde{(w, 1, \mathsf{x})}$, where $w = \mathcal{C}_k.w$ and $\mathsf{x} = \mathcal{C}_k.\phi.\mathsf{x}$.

Because the input linearized committed CCS instance-witness pairs are satisfying, we have for all $j \in [t]$ and $k \in [\mu]$

$$\begin{aligned}
v_{j,k} &= \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r_x, y) \cdot \widetilde{z}_{1,k}(y) && \text{By precondition.} \\
&= \sum_{x \in \{0,1\}^s} \widetilde{eq}(r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_{1,k}(y) \right) && \text{By Lemma 6.} \\
&= \sum_{x \in \{0,1\}^s} L_{j,k}(x) && \text{By construction.}
\end{aligned}$$

Moreover, because the input committed CCS instance-witness pairs are satisfying, for all $k \in [\nu]$, we have, for all $x \in \{0,1\}^s$ that

$$0 = \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_{2,k}(y) \right)$$

Therefore, for all $k \in [\nu]$, we have that the polynomial in variables t

$$\sum_{x \in \{0,1\}^s} \widetilde{eq}(t, x) \cdot \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_{2,k}(y) \right)$$

must be the zero polynomial. Therefore, for β sampled by the verifier, we have that for all $k \in [\nu]$

$$\begin{aligned} 0 &= \sum_{x \in \{0,1\}^s} \widetilde{eq}(\beta, x) \cdot \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_{2,k}(y) \right) \\ &= \sum_{x \in \{0,1\}^s} Q_k(x) \qquad \text{By construction.} \end{aligned}$$

Therefore, for γ sampled by the verifier, by linearity, we have that

$$\begin{aligned} \sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot \mathcal{L}_k.\phi.v_j &= \sum_{x \in \{0,1\}^s} \left(\left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot L_{j,k}(x) \right) + \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot Q_k(x) \right) \right) \\ &= \sum_{x \in \{0,1\}^s} g(x) \end{aligned}$$

Therefore, by the perfect completeness of the sum-check protocol, we have for $e_1 = \widetilde{eq}(r_x, r'_x)$, $e_2 = \widetilde{eq}(\beta, r'_x)$,

$$\sigma_{j,k} = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z}_{1,k}(y),$$

for $j \in [t]$ and $k \in [\mu]$, and

$$\theta_{j,k} = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z}_{2,k}(y)$$

$j \in [t]$ and $k \in [\nu]$ that

$$\begin{aligned} c &= g(r'_x) \\ &= \left(\left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot L_{j,k}(r'_x) \right) + \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot Q_k(r'_x) \right) \right) \\ &= \left(\left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot e_1 \cdot \sigma_{j,k} \right) + \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot e_2 \sum_{i \in [q]} c_i \cdot \prod_{j \in S_i} \theta_{j,k} \right) \right). \end{aligned}$$

This implies that the verifier will not abort.

Now, consider the following linearized committed CCS instances obtained by reducing input committed CCS instances (for all $k \in [\nu]$):

$$(\mathcal{C}_k.\phi.C, 1, \mathcal{C}_k.\phi.\mathsf{x}, r'_x, \theta_{1,k}, \ldots, \theta_{t,k}).$$

By the precondition that committed CCS instance-witness pairs are satisfying and by the definition of $(\theta_{1,k}, \ldots, \theta_{t,k})$ for all $k \in [\nu]$, we have that kth linearized

committed CCS instance is satisfied by the witness of the kth committed CCS instances i.e., $\mathcal{C}_k.w$.

Therefore, for a random ρ sampled by the verifier, and for

$$\begin{aligned}
C &\leftarrow \textstyle\sum_{k\in[\mu]} \rho^k \cdot \mathcal{L}_k.\phi.C + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot \mathcal{C}_k.\phi.C \\
u &\leftarrow \textstyle\sum_{k\in[\mu]} \rho^k \cdot \mathcal{L}_k.\phi.u + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot 1 \\
\mathsf{x} &\leftarrow \textstyle\sum_{k\in[\mu]} \rho^k \cdot \mathcal{L}_k.\phi.\mathsf{x} + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot \mathcal{C}_k.\phi.\mathsf{x} \\
v_j &\leftarrow \textstyle\sum_{k\in[\mu]} \rho^k \cdot \sigma_{j,k} + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot \theta_{j,k}
\end{aligned}$$

we have that the output linearized CCS instance

$$(C, u, \mathsf{x}, r'_x, v_1, \ldots, v_t)$$

is satisfied by the witness $\widetilde{w} \leftarrow \sum_{k\in[\mu]} \rho^k \cdot \mathcal{L}_k.w + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot \mathcal{C}_k.w$ by linearity and the additive homomorphism property of the polynomial commitment scheme. □

Some of our probabilistic analysis below is adapted from the proof of forking lemma for folding schemes [43], which itself builds on the proof of the forking lemma for interactive arguments [11].

Lemma 14 (Knowledge Soundness). *Construction 1 satisfies knowledge soundness.*

Proof. Consider an adversary $\mathcal{A}$ that adaptively picks the structure and instances, and a malicious prover $\mathcal{P}^*$ that succeeds with probability ϵ. Let $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N)$. Suppose on input pp and random tape r, the adversary $\mathcal{A}$ picks structures satisfying compat

$$\mathsf{s} = \mathsf{s}_1 = \mathsf{s}_2 = ([M_1, \ldots, M_t], [S_1, \ldots, S_q], [c_1, \cdots, c_q]),$$

μ linearized committed CCS instances, ν committed CCS instances, and some auxiliary state st. We now construct an expected-polynomial time extractor $\mathcal{E}$ that succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$ in obtaining satisfying witnesses for the original instances.

$\underline{\mathcal{E}(\mathsf{pp}, \mathsf{r})}$:

1. Obtain the output tuple from $\mathcal{A}$:

$$((\mathsf{s}_1, \mathsf{s}_2), (\vec{\mathsf{u}}_1, \vec{\mathsf{u}}_2), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{r}).$$

2. Compute $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2))$.

3. Run the interaction

$$(\mathsf{u}^{(1)}, \mathsf{w}^{(1)}) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), \vec{\mathsf{u}}_1, \vec{\mathsf{u}}_2, \mathsf{st})$$

once with the final verifier challenge $\rho^{(1)} \xleftarrow{\$} \mathbb{F}$.

4. Abort if $(\mathsf{pp}, \mathsf{s}, \mathsf{u}^{(1)}, \mathsf{w}^{(1)}) \notin \mathcal{R}_{\mathsf{LCCCS}}$.

5. For $i \in \{2, \ldots, \mu + \nu\}$, rewind the interaction

$$(\mathsf{u}^{(i)}, \mathsf{w}^{(i)}) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), \vec{\mathsf{u}}_1, \vec{\mathsf{u}}_2, \mathsf{st})$$

with a *different* verifier's final challenge $\rho^{(i)} \xleftarrow{\$} \mathbb{F}$ while maintaining the same prior randomness. Keep doing so until $(\mathsf{pp}, \mathsf{s}, \mathsf{u}^{(i)}, \mathsf{w}^{(i)}) \in \mathcal{R}_{\mathsf{LCCCS}}$.

6. Interpolating points $(\rho^{(i)}, \mathsf{w}^{(i)})$ for all $i \in [\mu + \nu]$ retrieve μ witnesses $\vec{\mathsf{w}}_1 = (\mathsf{w}_{1,1}, \ldots, \mathsf{w}_{1,\mu})$ and ν witnesses $\vec{\mathsf{w}}_2 = (\mathsf{w}_{2,1}, \ldots, \mathsf{w}_{2,\nu})$ such that for $i \in [\mu + \nu]$

$$\mathsf{w}^{(i)} = \sum_{k \in [\mu]} \rho^k \cdot \mathsf{w}_{1,k} + \sum_{k \in [\nu]} \rho^{\mu + k} \cdot \mathsf{w}_{2,k}. \tag{12}$$

7. Output $(\vec{\mathsf{w}}_1, \vec{\mathsf{w}}_2)$.

We first demonstrate that the extractor $\mathcal{E}$ runs in expected polynomial time. Observe that $\mathcal{E}$ runs the interaction once, and if it does not abort, keeps rerunning the interaction until $\mathcal{P}^*$ succeeds. Thus, the expected number of times $\mathcal{E}$ runs the interaction is

$$1 + \Pr[\text{First call to } \langle \mathcal{P}^*, \mathcal{V} \rangle \text{ succeeds}] \cdot \frac{\mu + \nu - 1}{\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle \text{ succeeds}]} = 1 + \epsilon \cdot \frac{\mu + \nu - 1}{\epsilon} = \mu + \nu.$$

Therefore, we have that the extractor runs in expected polynomial-time.

We now analyze $\mathcal{E}$'s success probability. We must demonstrate that $\mathcal{E}$ succeeds in producing $\vec{\mathsf{w}}_1$ and $\vec{\mathsf{w}}_2$ such that

$$(\mathsf{pp}, \mathsf{s}, \vec{\mathsf{u}}_1, \vec{\mathsf{w}}_1) \in \mathcal{R}_{\mathsf{LCCCS}} \quad \text{and} \quad (\mathsf{pp}, \mathsf{s}, \vec{\mathsf{u}}_2, \vec{\mathsf{w}}_2) \in \mathcal{R}_{\mathsf{CCCS}}$$

with probability $\epsilon - \mathsf{negl}(\lambda)$.

To do so, we first show that the extractor successfully produces *some* output (i.e., does not abort) in under $|\mathbb{F}|$ rewinding steps with probability $\epsilon - \mathsf{negl}(\lambda)$. Note that $|\mathbb{F}|$ is a worst case bound and we have already established that the extractor runs in expected polynomial time. By the malicious prover's success probability, we have that the extractor does not abort in step (4) with probability ϵ. Given that the extractor does not abort in step (4), by Markov's inequality, we have that the extractor rewinds more than $|\mathbb{F}|$ times with probability $(\mu + \nu)/|\mathbb{F}|$. Thus, the probability that the extractor does not abort in step (4) and requires less than $|\mathbb{F}|$ rewinds is $(1 - (\mu + \nu)/|\mathbb{F}|) \cdot \epsilon = \epsilon - \mathsf{negl}(\lambda)$.

Next, if the extractor does not abort, we show that the extractor succeeds in producing satisfying witnesses with probability $1-\mathsf{negl}(\lambda)$. This brings the overall extractor success probability to $\epsilon - \mathsf{negl}(\lambda)$.

Indeed, for $i \in \{1, \ldots, \mu+\nu\}$, let $\mathsf{u}^{(i)} = (C^{(i)}, u^{(i)}, \mathsf{x}^{(i)}, r_x^{(i)}, v_1^{(i)}, \ldots, v_t^{(i)})$. We first show that the retrieved polynomials are valid openings to the corresponding commitments in the instance. For $i \in \{1, \ldots, \mu+\nu\}$, because $\mathsf{w}^{(i)}$ is a satisfying witness, by construction,

$$
\begin{aligned}
&\sum_{k\in[\mu]} {\rho^{(i)}}^k \cdot \mathsf{Commit}(\mathsf{pp}, \mathsf{w}_{1,k}) + \sum_{k\in[\nu]} {\rho^{(i)}}^{\mu+k} \cdot \mathsf{Commit}(\mathsf{pp}, \mathsf{w}_{2,k}) \\
&= \mathsf{Commit}(\mathsf{pp}, (\sum_{k\in[\mu]} {\rho^{(i)}}^k \cdot \mathsf{w}_{1,k}) + (\sum_{k\in[\nu]} {\rho^{(i)}}^{\mu+k} \cdot \mathsf{w}_{2,k})) && \text{By additive homomorphism.} \\
&= \mathsf{Commit}(\mathsf{pp}, \mathsf{w}^{(i)}) && \text{By Equation (12).} \\
&= C^{(i)} && \text{Witness } \mathsf{w}^{(i)} \text{ is a satisfying opening.} \\
&= \sum_{k\in[\mu]} {\rho^{(i)}}^k \cdot \mathsf{u}_{1,k}.C + \sum_{k\in[\nu]} {\rho^{(i)}}^{\mu+k} \cdot \mathsf{u}_{2,k}.C
\end{aligned}
$$

Interpolating, we have that for all $i \in [\mu]$ and $j \in [\nu]$

$$\mathsf{Commit}(\mathsf{pp}, \mathsf{w}_{1,i}) = \mathsf{u}_{1,i}.C \tag{13}$$
$$\mathsf{Commit}(\mathsf{pp}, \mathsf{w}_{2,j}) = \mathsf{u}_{2,j}.C. \tag{14}$$

Next, we must argue that $\vec{\mathsf{w}}_1$ and $\vec{\mathsf{w}}_2$ satisfy the remainder of the instances $\vec{\mathsf{u}}_1$ and $\vec{\mathsf{u}}_2$ respectively under the structure s.

Indeed, consider $\{\sigma_{j,k}\}$ (for all $j \in [t]$ and $k \in [\mu]$), and $\{\theta_{j,k}\}$ (for all $j \in [t]$ and $k \in [\nu]$) sent by the prover which by the extractor's construction are identical across all executions of the interaction. By the verifier's computation we have that for $i \in \{1, \ldots, \mu+\nu\}$ and all $j \in [t]$

$$\sum_{k\in[\mu]} (\rho^{(i)})^k \cdot \sigma_{j,k} + \sum_{k\in[\nu]} (\rho^{(i)})^{\mu+k} \cdot \theta_{j,k} = v_j^{(i)} \tag{15}$$

Now, because $\mathsf{w}^{(i)}$ is a satisfying witness, for $i \in \{1, \ldots, \mu+\nu\}$ we have for all $j \in [t]$ that

$$v_j^{(i)} = \sum_{y\in\{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z}^{(i)}(y),$$

where $\widetilde{z}^{(i)} = (\widetilde{w^{(i)}, u^{(i)}}, \mathsf{x}^{(i)})$ where $w^{(i)}$ is the result of interpreting $\mathsf{w}^{(i)}$ as a multilinear polynomial.

However, by Equations (12) and (15), for $i \in \{1, \ldots, \mu + \nu\}$ and $j \in [t]$, this implies that

$$\begin{aligned}\sum_{k\in[\mu]} (\rho^{(i)})^k \cdot \sigma_{j,k} + \sum_{k\in[\nu]} (\rho^{(i)})^{\mu+k} \cdot \theta_{j,k} \\ = \sum_{k\in[\mu]} (\rho^{(i)})^k \cdot \sum_{y\in\{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z_{1,k}}(y) \\ + \sum_{k\in[\nu]} (\rho^{(i)})^{\mu+k} \cdot \sum_{y\in\{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z_{2,k}}(y),\end{aligned}$$

where $\widetilde{z_{1,k}} = (w_{1,k}, \widetilde{u, \mathsf{u}_{1,k}.\mathsf{x}})$ for $k \in [\mu]$ where $w_{1,k}$ denotes the multilinear polynomial interpretation of $\mathsf{w}_{1,k}$ and $\widetilde{z_{2,k}} = (w_{2,k}, \widetilde{1, \mathsf{u}_{2,k}.\mathsf{x}})$ for $k \in [\nu]$ where $w_{2,k}$ represents the multilinear polynomial interpretation of $\mathsf{w}_{2,k}$. Interpolating, we have that, for all $j \in [t]$

$$\begin{aligned}\forall k \in [\mu],\ \sigma_{j,k} = \sum_{y\in\{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z_{1,k}}(y) \\ \forall k \in [\nu],\ \theta_{j,k} = \sum_{y\in\{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z_{2,k}}(y)\end{aligned}$$

Thus, because that the verifier does not abort, we have that

$$\begin{aligned}c &= \left(\sum_{j\in[t],k\in[\mu]} \gamma^{(k-1)\cdot t+j} \cdot e_1 \cdot \sigma_{j,k}\right) + \left(\sum_{k\in[\nu]} \gamma^{\mu\cdot t+k} \cdot e_2 \cdot \sum_{i\in[q]} c_i \cdot \prod_{j\in S_i} \theta_j\right) \\ &= \left(\sum_{j\in[t],k\in[\mu]} \gamma^{(k-1)\cdot t+j} \cdot \widetilde{eq}(r_x, r'_x) \cdot \sigma_{j,k}\right) + \left(\sum_{k\in[\nu]} \gamma^{\mu\cdot t+k} \cdot \widetilde{eq}(\beta, r'_x) \cdot \sum_{i\in[q]} c_i \cdot \prod_{j\in S_i} \theta_{j,k}\right) \\ &= \left(\sum_{j\in[t],k\in[\mu]} \gamma^{(k-1)\cdot t+j} \cdot \widetilde{eq}(r_x, r'_x) \cdot \sum_{y\in\{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z_{1,k}}(y)\right) + \\ &\quad \left(\sum_{k\in[\nu]} \gamma^{\mu\cdot t+k} \cdot \widetilde{eq}(\beta, r'_x) \cdot \sum_{i\in[q]} c_i \cdot \prod_{j\in S_i} \sum_{y\in\{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z_{2,k}}(y)\right) \\ &= \left(\left(\sum_{j\in[t],k\in[\mu]} \gamma^{(k-1)\cdot t+j} \cdot L_{j,k}(r'_x)\right) + \left(\sum_{k\in[\nu]} \gamma^{\mu\cdot t+k} \cdot Q_k(r'_x)\right)\right) \\ &= g(r'_x)\end{aligned}$$

By the soundness of the sum-check protocol, this implies that with probability $1 - O(d \cdot s)/|\mathbb{F}| = 1 - \mathsf{negl}(\lambda)$ over the choice of r'_x,

$$\begin{aligned}
T &= \sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot v_{j,k} + \sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot 0 \\
&= \sum_{x \in \{0,1\}^s} g(x) \\
&= \sum_{x \in \{0,1\}^s} \left(\left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot L_{j,k}(x) \right) + \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot Q_k(x) \right) \right) \\
&= \left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot \left(\sum_{x \in \{0,1\}^s} L_{j,k}(x) \right) \right) + \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot \left(\sum_{x \in \{0,1\}^s} Q_k(x) \right) \right)
\end{aligned}$$

By the Schwartz-Zippel lemma [56], this implies that with probability $1 - O(t \cdot \mu + \nu)/|\mathbb{F}| = 1 - \mathsf{negl}(\lambda)$ over the choice of γ, we have for all $j \in [t]$ and $k \in [\mu]$

$$v_{j,k} = \sum_{x \in \{0,1\}^s} L_{j,k}(x),$$

and for all $k \in [\nu]$

$$0 = \sum_{x \in \{0,1\}^s} Q_k(x).$$

Now, for all $j \in [t]$ and $k \in [\mu]$, we have

$$\begin{aligned}
v_{j,k} &= \sum_{x \in \{0,1\}^s} L_{j,k}(x) \\
&= \sum_{x \in \{0,1\}^s} \widetilde{eq}(r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} M_j(x, y) \cdot \widetilde{z}_{1,k}(y) \right) \\
&= \sum_{y \in \{0,1\}^{s'}} M_j(r_x, y) \cdot \widetilde{z}_{1,k}(y)
\end{aligned}$$

This implies that $\vec{\mathsf{w}}_1$ is a satisfying witness to $\vec{\mathsf{u}}_1$.

Finally, we have that for all $k \in [\nu]$

$$\begin{aligned}
0 &= \sum_{x \in \{0,1\}^s} Q_k(x) \\
&= \sum_{x \in \{0,1\}^s} \widetilde{eq}(\beta, x) \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_{2,k}(y) \right) \right) \\
&= \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(\beta, y) \cdot \widetilde{z}_{2,k}(y) \right)
\end{aligned}$$

By the Schwartz-Zippel lemma, this implies that with probability $1 - s/|\mathbb{F}| = 1 - \mathsf{negl}(\lambda)$ over the choice of β, we have that for all $x \in \{0,1\}^s$

$$0 = \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_{2,k}(y) \right)$$

This implies that $\vec{\mathsf{w}}_2$ is a satisfying witness to $\vec{\mathsf{u}}_2$.

Thus, if the extractor does not abort, it succeeds in producing satisfying witness $(\vec{\mathsf{w}}_1, \vec{\mathsf{w}}_2)$ with probability $1 - \mathsf{negl}(\lambda)$. □

H.2 Proof of Lemma 3 (Folding CCS NIVC-compatibility)

Lemma 15 (NIVC-compatibility). *Construction 1 is NIVC-compatible.*

Proof. NP-completeness of $\mathcal{R}_{\mathsf{CCS}}$ follows from [60, Lemma 1], which reduces R1CS to $\mathcal{R}_{\mathsf{CCS}}$. Furthermore, Gennaro et al. [31, Section 7.4] and Parno et al. [54, Section 2.2.1] implicitly reduce circuit satisfiability to R1CS (Thaler [65, Section 8.4] provides explicit details). These two reductions induce the functions enc, $\mathsf{enc}_{\mathsf{str}}$ and $\mathsf{enc}_{\mathsf{inst}}$.

In particular, let the circuit satisfiability relation be characterized by circuits over n total gates, m multiplication gates, and without loss of generality input size $\ell/2$ and output size $\ell/2$. We demonstrate how to reduce circuit satisfiability to $\mathcal{R}_{\mathsf{CCS}}$ characterized by size bounds $m, n, N = \Omega(m), \ell, t = 3, q = 2, d = 2$.

First, we explicitly describe the function $\mathsf{enc}_{\mathsf{str}}$, which takes as input an arithmetic circuit F and produces $\mathcal{R}_{\mathsf{CCS}}$ and $\mathcal{R}_{\mathsf{LCCS}}$ structures s_1 and s_2

$\underline{\mathsf{enc}_{\mathsf{str}}(F) \to (\mathsf{s}_1, \mathsf{s}_2)}$:

1. Initalize matrices $A, B, C \in \mathbb{F}^{m \times (n+1)}$ with zeros.
2. For each multiplication gate index $i \in [m]$ in F, do the following:

(a) let L be the indices of all the upstream left input addition gates such that gate i is the first downstream multiplication gate. Similarly let R be all the upstream right input gates such that gate i is the first downstream multiplication gate.

(b) Let $A_{i,j} = 1$ for $j \in L$, $B_{i,j} = 1$ for $j \in R$, and $C_{i,i} = 1$. This encodes the constraint that the result of multiplying the sum of all the upstream left addition gates and the sum of all the upstream right addition gates results in the wire value assigned at gate i.

3. Let $S_1 \leftarrow \{1, 2\}$ and $S_2 \leftarrow \{3\}$. Output CCS and LCCS structure

$$\mathsf{s}_1 = \mathsf{s}_2 = \left((\widetilde{A}, \widetilde{B}, \widetilde{C}), (S_1, S_2), (1, -1)\right)$$

where $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$ denote the multilinear extensions of A, B, and C.

By observation, we have that $\mathsf{enc}_{\mathsf{str}}$ is invertible. In particular, given matrices A, B, and C, we can parse out the inputs to each gate in the circuit F.

Next, we explicitly describe the function $\mathsf{enc}_{\mathsf{inst}}$, which takes as input an arithmetic circuit public input x and output y and produces an $\mathcal{R}_{\mathsf{CCS}}$ instance x

$\underline{\mathsf{enc}_{\mathsf{inst}}((x, y)) \rightarrow \mathsf{x}}$:

1. Output (x, y).

By observation, we have that $\mathsf{enc}_{\mathsf{inst}}$ is invertible. In particular, provided an $\mathcal{R}_{\mathsf{CCS}}$ instance $\mathsf{x} \in \mathbb{F}^{\ell}$ we can recover an arithmetic circuit public input and output tuple $(x, y) \in (\mathbb{F}^{\ell/2}, \mathbb{F}^{\ell/2})$ by splitting x into its first and second half.

Given $\mathsf{enc}_{\mathsf{str}}$ and $\mathsf{enc}_{\mathsf{inst}}$, we can explicitly describe enc, which takes as input an arithmetic circuit F, a circuit input x, a non-deterministic input w, and an output y, and outputs a corresponding $\mathcal{R}_{\mathsf{CCS}}$ structure-instance-witness tuple.

$\underline{\mathsf{enc}(F, (x, y), w) \rightarrow \mathsf{x}}$:

1. Let $(\mathsf{s}_1, \mathsf{s}_2) \leftarrow \mathsf{enc}_{\mathsf{str}}(F)$.
2. Let $\mathsf{x} \leftarrow \mathsf{enc}_{\mathsf{inst}}((x, y))$.
3. Let w be the gate wire values that results from executing F on inputs x and w excluding the public input x and output y wire values.
4. Output CCS structure-instance-witness tuple $(\mathsf{s}_2, \mathsf{x}, \mathsf{w})$.

By Setty et al. [60, Lemma 1] and Parno et al. [54], for any arithmetic circuit F, input x, non-deterministic input w, and output y, for $(\mathsf{s}_2, \mathsf{u}, \mathsf{w}) \leftarrow \mathsf{enc}(F, (x, y), w)$ we have that $(\mathsf{s}_2, \mathsf{u}, \mathsf{w}) \in \mathcal{R}'_2$ if and only if $F(x, w) = y$.

Moreover, we have that enc is invertible. In particular, Given a CCS structure-instance-witness pair $(\mathsf{s}_2, \mathsf{u}, \mathsf{w})$, we can compute F and (x, y) by the invertibility of $\mathsf{enc}_{\mathsf{str}}$ and $\mathsf{enc}_{\mathsf{inst}}$, and compute w by parsing the appropriate gates in w.

By observation, we have that $(\mathsf{s}_2, \mathsf{u}, \mathsf{w}) = \mathsf{enc}(F, (x, y), w)$. Combining all prior assertions we have that the NP-completeness property holds.

Next, by construction, for any arithmetic circuit F, an input x, a non-deterministic input w, and an output y, for $\mathcal{R}'_1$ and $\mathcal{R}'_2$ structures $(\mathsf{s}_1, \mathsf{s}_2) \leftarrow \mathsf{enc}_{\mathsf{str}}(F)$ and $\mathcal{R}'_2$ instance $\mathsf{u} \leftarrow \mathsf{enc}_{\mathsf{inst}}((x, y))$ we have that $(\mathsf{s}_2, \mathsf{u}, \mathsf{w}) = \mathsf{enc}(F, (x, y), w)$ for some $\mathcal{R}'_2$ witness w and that $\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = 1$. Combining all the prior assertions we have that the partial functions property holds.

Moreover, we have monotonicity holds by the construction of $\mathsf{enc}_{\mathsf{str}}$. In particular, each gate in the input arithmetic circuit F corresponds to exactly one constraint in the output CCS structure.

Finally, we have that $\mathcal{R}_{\mathsf{LCCCS}}$ has default instances because for any public parameters and structure, we have that $\mathsf{u}_\perp = (u = 0, \mathsf{x} = \vec{0}, r = 0, v_1 = 0, \ldots, v_t = 0)$ and $\mathsf{w}_\perp = 0$ is a satisfying instance-witness pair in $\mathcal{R}_1$. □

H.3 Proof of Theorem 4 (HyperNova)

Lemma 16 (Completeness). *Construction 2 is an NIVC scheme that satisfies completeness.*

Proof. Consider arbitrary PPT adversary $\mathcal{A}$. Suppose $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N)$. Suppose, on input pp, $\mathcal{A}$ produces polynomial-time functions $(\varphi, (F_1, \ldots, \mathbb{F}_\ell))$, instance (i, z_0, z_i), private input ω_i, and NIVC proof Π_i. Suppose that for

$$(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\varphi, (F_1, \ldots, F_\ell)))$$

we have that

$$\mathcal{V}(\mathsf{vk}, i, z_0, z_i, \Pi_i) = 1.$$

Then, for $\mathsf{pc}_{i+1} \in [\ell] \leftarrow \varphi(z_i, \omega_i)$, given

$$z_{i+1} \leftarrow F_{\mathsf{pc}_{i+1}}(z_i, \omega_i)$$

and

$$\Pi_{i+1} \leftarrow \mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i)$$

we must show that

$$\mathcal{V}(\mathsf{vk}, i+1, z_0, z_{i+1}, \Pi_{i+1}) = 1$$

with probability 1. We show this by considering the case when $i = 0$ and when $i \geq 1$.

Indeed, suppose $i = 0$. By the base case of $\mathcal{P}$ and F'_{pc_1}, we have

$$\Pi_1 = (((\mathsf{u}_\perp, \ldots, \mathsf{u}_\perp), (\mathsf{w}_\perp, \ldots, \mathsf{w}_\perp)), (\mathsf{u}_1, \mathsf{w}_1), \mathsf{pc}_1)$$

for some $(\mathsf{u}_1, \mathsf{w}_1)$. By definition, the instance-witness pair $(\mathsf{u}_\perp, \mathsf{w}_\perp)$ is satisfying. Moreover, by construction, $(\mathsf{u}_1, \mathsf{w}_1)$ must also be satisfying. Additionally, by the construction of F'_{pc_1}, we have

$$\mathsf{u}'_1 = \mathsf{enc}_{\mathsf{inst}}(\mathsf{hash}(\mathsf{vk}, 1, z_0, F_{\mathsf{pc}_1}(z_0, w_0), \mathsf{u}_\perp, \mathsf{pc}_1)).$$

where u'_1 is the portion of u_1 that excludes the commitment to the w_1. Therefore, we have

$$\mathcal{V}(\mathsf{pp}, 1, z_0, z_1, \mathit{\Pi}_1) = 1.$$

Suppose instead that $i \geq 1$. Let $\mathit{\Pi}_i$ be parsed as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \pi_i)$ and let $\mathit{\Pi}_{i+1}$ be parsed as $((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}), \pi_{i+1})$. By the construction of $\mathcal{P}$, we have that

$$(\mathsf{U}_{i+1}[\mathsf{pc}_i], \mathsf{W}_{i+1}[\mathsf{pc}_i], \pi) = \mathsf{NIFS.P}(\mathsf{pk}[\mathsf{pc}_i], (\mathsf{U}_i[\mathsf{pc}_i], \mathsf{W}_i[\mathsf{pc}_i]), (\mathsf{u}_i, \mathsf{w}_i)).$$

Thus, because $\mathit{\Pi}_i$ is satisfying, we have that $(\mathsf{u}_i, \mathsf{w}_i)$ and $(\mathsf{U}_i[\mathsf{pc}_i], \mathsf{W}_i[\mathsf{pc}_i])$ are satisfying instance-witness pairs (with respect to compatible structures). Then, by the completeness of the underlying folding scheme, we have that $(\mathsf{U}_{i+1}[\mathsf{pc}_i], \mathsf{W}_{i+1}[\mathsf{pc}_i])$ is a satisfying instance-witness pair. Therefore, because $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1})$ copies the remaining elements from $(\mathsf{U}_i, \mathsf{W}_i)$, we have that $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1})$ contains satisfying instance-witness pairs. Additionally, by the premise, we have that $\mathsf{u}'_i = \mathsf{enc}_{\mathsf{inst}}(\mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i, \mathsf{pc}_i))$ where u'_i represents the portion of u_i that excludes the commitment to the witness. Therefore, $\mathcal{P}$ can construct a satisfying instance-witness pair $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$ that represents the correct execution of $F'_{\mathsf{pc}_{i+1}}$ on input $(\mathsf{vk}_{\mathsf{fs}}, \mathsf{U}, \mathsf{u}, \mathsf{pc}_i, (i, z_0, z_i), \omega_i, \pi)$. By construction, this particular input implies that

$$\mathsf{u}'_{i+1} = \mathsf{enc}_{\mathsf{inst}}(\mathsf{hash}(\mathsf{vk}, i+1, z_0, z_{i+1}, \mathsf{U}_{i+1}, \mathsf{pc}_{i+1})) \tag{16}$$

by the correctness of the underlying folding scheme (again u'_{i+1} represents the portion of u_{i+1} that excludes the commitment to the witness). Moreover, because $\mathsf{pc}_{i+1} = \varphi(z_i, \omega_i)$, by construction, we have that $1 \leq \mathsf{pc}_{i+1} \leq \ell$. Thus, by Equation (16) we have

$$\mathcal{V}(\mathsf{vk}, i+1, z_0, z_{i+1}, \mathit{\Pi}_{i+1}) = 1.$$

□

Lemma 17 (Knowledge Soundness). *Construction 2 is an IVC scheme that satisfies knowledge soundness.*

Proof. Our approach is inspired by a recursive extraction technique described by Bünz et al [18]. Let n be a global constant. Consider deterministic expected polynomial-time adversary $\mathcal{P}^*$. Let $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N)$. Suppose on input pp and

randomness r, $\mathcal{P}^*$ outputs deterministic polynomial-time function φ, ℓ polynomial-time functions $(F_1, \ldots, F_\ell)$, instance (z_0, z), and NIVC proof Π. Let $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\varphi, (F_1, \ldots, F_\ell)))$. Suppose that

$$\mathcal{V}(\mathsf{vk}, (n, z_0, z), \Pi) = 1$$

with probability ϵ. We must construct an expected polynomial-time extractor $\mathcal{E}$ that, with input $(\mathsf{pp}, \mathsf{r})$, outputs $(\omega_0, \ldots, \omega_{n-1})$ such that by computing

$$z_{i+1} \leftarrow F_{\varphi(z_i, \omega_i)}(z_i, \omega_i)$$

we have that $z_n = z$ with probability $\epsilon - \mathsf{negl}(\lambda)$.

We show inductively that $\mathcal{E}$ can construct an expected polynomial-time extractor $\mathcal{E}_i(\mathsf{pp})$ that outputs $((z_i, \ldots, z_{n-1}), (\omega_i, \ldots, \omega_{n-1}), \Pi_i)$ such that for all $j \in \{i+1, \ldots, n\}$,

$$z_j = F_{\varphi(z_{j-1}, \omega_{j-1})}(z_{j-1}, \omega_{j-1})$$

and

$$\mathcal{V}(\mathsf{vk}, i, z_0, z_i, \Pi_i) = 1 \tag{17}$$

for $z_n = z$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Then, because in the base case when $i = 0$, $\mathcal{V}$ checks that $z_0 = z_i$, the values $(\omega_0, \ldots, \omega_{n-1})$ retrieved by $\mathcal{E}_0(\mathsf{pp})$ are such that computing $z_{i+1} = F(z_i, \omega_i)$ for all $i \geq 1$ gives $z_n = z$.

At a high level, to construct an extractor $\mathcal{E}_{i-1}$, we first assume the existence of $\mathcal{E}_i$ that satisfies the inductive hypothesis. We then use $\mathcal{E}_i(\mathsf{pp})$ to construct an adversary for the non-interactive folding scheme (which we denote as $\widetilde{\mathcal{P}}_{i-1}$). This in turn guarantees an extractor for the non-interactive folding scheme, which we denote as $\widetilde{\mathcal{E}}_{i-1}$. We then use $\widetilde{\mathcal{E}}_{i-1}$ to construct $\mathcal{E}_{i-1}$ that satisfies the inductive hypothesis.

In the base case, for $i = n$, let $\mathcal{E}_n(\mathsf{pp}, \mathsf{r})$ output $(\bot, \bot, \Pi_n)$ where Π_n is the output of $\mathcal{P}^*(\mathsf{pp}, \mathsf{r})$. By the premise, $\mathcal{E}_n$ succeeds with probability ϵ in expected polynomial-time.

For $i \geq 1$, suppose $\mathcal{E}$ can construct an expected polynomial-time extractor $\mathcal{E}_i$ that outputs $((z_i, \ldots, z_{n-1}), (\omega_i, \ldots, \omega_{n-1}))$, and Π_i that satisfies the inductive hypothesis. To construct an extractor $\mathcal{E}_{i-1}$, $\mathcal{E}$ first constructs an adversary $\widetilde{\mathcal{P}}_{i-1}$ for the non-interactive folding scheme as follows:

$\underline{\widetilde{\mathcal{P}}_{i-1}(\mathsf{pp}, \mathsf{r})}$:

1. Let $((z_i, \ldots, z_{n-1}), (\omega_i, \ldots, \omega_{n-1}), \Pi_i) \leftarrow \mathcal{E}_i(\mathsf{pp}, \mathsf{r})$.
2. Parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \mathsf{pc}_i)$.
3. Compute compatible structures $(\mathsf{s}_{1,\mathsf{pc}_i}, \mathsf{s}_{2,\mathsf{pc}_i}) \leftarrow \mathsf{enc}_{\mathsf{str}}(F'_{\mathsf{pc}_i})$.

4. Parse non-deterministic inputs $(\mathsf{U}_{i-1}, \mathsf{u}_{i-1}, \pi_{i-1}, \mathsf{pc}_{i-1})$ to F'_{pc_i} from $\mathsf{enc}^{-1}(\mathsf{s}_{2,\mathsf{pc}_i}, \mathsf{u}_i, \mathsf{w}_i)$.
5. Output structures $(\mathsf{s}_{1,\mathsf{pc}_{i-1}}, \mathsf{s}_{2,\mathsf{pc}_{i-1}})$, unfolded instances $(\mathsf{U}_{i-1}[\mathsf{pc}_{i-1}], \mathsf{u}_{i-1})$, folded instance-witness pair $(\mathsf{U}_i[\mathsf{pc}_{i-1}], \mathsf{W}_i[\mathsf{pc}_{i-1}])$, and folding proof π_{i-1}.

We now analyze the success probability of $\widetilde{\mathcal{P}}_{i-1}$. By the inductive hypothesis, we have that $\mathcal{V}(\mathsf{vk}, i, z_0, z_i, \mathit{\Pi}_i) = 1$, where $\mathit{\Pi}_i \leftarrow \mathcal{E}_i(\mathsf{pp}, \mathsf{r})$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Therefore, by the the verifier's checks we have that $(\mathsf{u}_i, \mathsf{w}_i)$ is satisfying, $(\mathsf{U}_i, \mathsf{W}_i)$ consists of satisfying instance-witness pairs, and that

$$\mathsf{u}'_i = \mathsf{enc}_{\mathsf{inst}}(\mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i, \mathsf{pc}_i))$$

where u'_i represents the portion of u_i that excludes the commitment to the witness. Then, by the construction of F'_{pc_i} and the binding property of the hash function, we have that $1 \leq \mathsf{pc}_{i-1} \leq \ell$ and

$$\mathsf{U}_i[\mathsf{pc}_{i-1}] = \mathsf{NIFS}.\mathcal{V}(\mathsf{vk}, \mathsf{U}_{i-1}[\mathsf{pc}_{i-1}], \mathsf{u}_{i-1}, \pi_{i-1})$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. Thus, $\widetilde{\mathcal{P}}_{i-1}$ succeeds in producing an accepting folded instance-witness pair $(\mathsf{U}_i[\mathsf{pc}_{i-1}], \mathsf{W}_i[\mathsf{pc}_{i-1}])$, for instances $\mathsf{U}_{i-1}[\mathsf{pc}_{i-1}]$ and u_{i-1}, with probability $\epsilon - \mathsf{negl}(\lambda)$ in expected polynomial-time.

Then, by the knowledge soundness of the underlying non-interactive multi-folding scheme (Assumption 1) there exists an extractor $\widetilde{\mathcal{E}}_{i-1}$ that outputs $(\mathsf{W}_{i-1}[\mathsf{pc}_{i-1}], \mathsf{w}_{i-1})$ such that $(\mathsf{U}_{i-1}[\mathsf{pc}_{i-1}], \mathsf{W}_{i-1}[\mathsf{pc}_{i-1}])$ and $(\mathsf{u}_{i-1}, \mathsf{w}_{i-1})$ are satisfying with respect to structures $\mathsf{s}_{1,\mathsf{pc}_{i-1}}$ and $\mathsf{s}_{2,\mathsf{pc}_{i-1}}$ respectively with probability $\epsilon - \mathsf{negl}(\lambda)$ in expected polynomial-time.

Given an expected polynomial-time $\widetilde{\mathcal{P}}_{i-1}$ and an expected polynomial-time $\widetilde{\mathcal{E}}_{i-1}$, $\mathcal{E}$ constructs an expected polynomial time $\mathcal{E}_{i-1}$ as follows

$\underline{\mathcal{E}_{i-1}(\mathsf{pp}, \mathsf{r})}$:

1. Run $\widetilde{\mathcal{P}}_{i-1}(\mathsf{pp}, \mathsf{r})$ to retrieve unfolded instances $(\mathsf{u}'_{i-1}, \mathsf{u}_{i-1})$ and parse

   $$((z_i, \ldots, z_{n-1}), (\omega_i, \ldots, \omega_{n-1}), \mathit{\Pi}_i)$$

   from its internal state.
2. Parse $\mathit{\Pi}_i$ as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \mathsf{pc}_i)$.
3. Compute $(\mathsf{s}_{1,\mathsf{pc}_i}, \mathsf{s}_{2,\mathsf{pc}_i}) \leftarrow \mathsf{enc}_{\mathsf{str}}(F'_{\mathsf{pc}_i})$
4. Parse private inputs z_{i-1}, ω_{i-1}, and pc_{i-1} to F'_{pc_i} from $\mathsf{enc}^{-1}(\mathsf{s}_2, \mathsf{u}_i, \mathsf{w}_i)$.
5. Let $(\mathsf{w}'_{i-1}, \mathsf{w}_{i-1}) \leftarrow \widetilde{\mathcal{E}}_{i-1}(\mathsf{pp})$.
6. Set $(\mathsf{U}_{i-1}, \mathsf{W}_{i-1}) \leftarrow (\mathsf{U}_i, \mathsf{W}_i)$ and update

   $$(\mathsf{U}_{i-1}[\mathsf{pc}_{i-1}], \mathsf{W}_{i-1}[\mathsf{pc}_{i-1}]) \leftarrow (\mathsf{u}'_{i-1}, \mathsf{w}'_{i-1})$$

7. Let $\mathit{\Pi}_{i-1} \leftarrow ((\mathsf{U}_{i-1}, \mathsf{W}_{i-1}), (\mathsf{u}_{i-1}, \mathsf{w}_{i-1}), \mathsf{pc}_{i-1})$.
8. Output $((z_{i-1}, \ldots, z_{n-1}), (\omega_{i-1}, \ldots, \omega_{n-1}), \mathit{\Pi}_{i-1})$.

We first reason that the output $(z_{i-1}, \ldots, z_{n-1})$, and $(\omega_{i-1}, \ldots, \omega_{n-1})$ are valid. By the inductive hypothesis, we already have that for all $j \in \{i+1, \ldots, n\}$,

$$z_j = F_{\mathsf{pc}_j}(z_{j-1}, \omega_{j-1}),$$

and that $\mathcal{V}(\mathsf{vk}, i, z_0, z_i, \mathit{\Pi}_i) = 1$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Because $\mathcal{V}$ additionally checks that

$$\mathsf{u}'_i = \mathsf{enc}_{\mathsf{inst}}(\mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i, \mathsf{pc}_i)), \tag{18}$$

where u'_i represents the portion of u_i that excludes the commitment to the witness, by the construction of F'_{pc_i} and the binding property of the hash function, we have

$$F_{\mathsf{pc}_i}(z_{i-1}, \omega_{i-1}) = z_i$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. Next, we argue that $\mathit{\Pi}_{i-1}$ is valid. Because $(\mathsf{u}_i, \mathsf{w}_i)$ satisfies F', and $(\mathsf{U}_{i-1}, \mathsf{u}_{i-1})$ were retrieved from w_i, by the binding property of the hash function, and by Equation (18), we have that

$$\mathsf{u}'_{i-1} = \mathsf{enc}_{\mathsf{inst}}(\mathsf{hash}(\mathsf{vk}, i-1, z_0, z_{i-1}, \mathsf{U}_{i-1}, \mathsf{pc}_{i-1}))$$

where u'_{i-1} represents the portion of u_{i-1} that excludes the commitment the witness. Additionally, in the case where $i = 1$, by the base case check of $F'_{\varphi(z_0,\omega_0)}$, we have that $z_{i-1} = z_0$. Because $\widetilde{\mathcal{E}}_{i-1}$ succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$, and the remainder of the elements of $(\mathsf{U}_{i-1}, \mathsf{W}_{i-1})$ are directly copied from $(\mathsf{U}_i, \mathsf{W}_i)$ we have that all the elements of $(\mathsf{U}_{i-1}, \mathsf{W}_{i-1})$ are satisfying. Moreover, by construction of F'_{pc_i} we have that $1 \leq \mathsf{pc}_{i-1} \leq \ell$. Thus, we have that

$$\mathcal{V}(\mathsf{vk}, i-1, z_0, z_{i-1}, \mathit{\Pi}_{i-1}) = 1$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. □

H.4 Proof of Theorem 4 (CycleFold)

Lemma 18 (Perfect Completeness). *Construction 7 satisfies perfect completeness.*

Proof. Consider public parameters $\mathsf{pp} = (m, n, N, \ell, t, q, d, |\mathsf{s}_{\mathsf{EC}}|, \mathsf{pp}_{\mathsf{PC}}, \mathsf{pp}_{\mathsf{VC}}) \leftarrow \mathcal{G}(1^\lambda, N)$ and let $s = \log m$ and $s' = \log n$. Let $\mathsf{s}_{\mathsf{EC}} = (A, B, C)$ denote a committed relaxed R1CS structure defined over F_q, with public IO (ρ, C_1, C_2, C'), where $\rho \in \mathbb{F}_p, C_1 \in \mathbb{G}_1, C_2 \in \mathbb{G}_1, C' \in \mathbb{G}_1$. This constraint system enforces that $C' = C_1 + \rho \cdot C_2$, where $+$ is the elliptic curve point addition and $\cdot$ is the elliptic curve scalar multiplication operation in $\mathbb{G}_1$.

Consider arbitrary structures $(\mathsf{s}_1, \mathsf{s}_2) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = \mathsf{true}$. Let $\mathsf{s}_1 = ((\widetilde{M}_1, \ldots, \widetilde{M}_t), (S_1, \ldots, S_q), (c_1, \ldots, c_q))$, and let $\mathsf{s}_2 = (A, B, C)$. Consider prover and verifier keys $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2))$. Suppose that the prover and the verifier are provided with a linearized committed CCS instance and a committed relaxed R1CS instance

$$((C_1, u, \mathsf{x}_1, r_x, v_1, \ldots, v_t), (\overline{E}_1, u_1, \overline{W}_1, x_1)),$$

and a committed CCS instance

$$(C_2, \mathsf{x}_2).$$

Suppose that the prover additionally is provided with the corresponding satisfying witnesses $(\widetilde{w}_1, (E_1, W_1))$ and $\widetilde{w}_2$.

Because the input linearized committed CCS instance-witness pair is satisfying, we have, for $\widetilde{z}_1 = (\widetilde{w_1, u, \mathsf{x}_1})$, that

$$\begin{aligned}
v_j &= \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r_x, y) \cdot \widetilde{z}_1(y) && \text{By precondition.} \\
&= \sum_{x \in \{0,1\}^{s}} \widetilde{eq}(r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_1(y) \right) && \text{By Lemma 6} \\
&= \sum_{x \in \{0,1\}^{s}} L_j(x) && \text{By construction.}
\end{aligned}$$

Furthermore, because the input committed relaxed R1CS instance-witness pair is also satisfying, we have for $Z_1 = (W_1, u_1, x_1)$, $AZ_1 \circ BZ_1 = u \cdot CZ_1 + E_1$.

Moreover, because the input committed CCS instance-witness pair is satisfying, we have, for all $x \in \{0,1\}^s$ and for $\widetilde{z}_2(x) = (\widetilde{w_2, 1, \mathsf{x}_2})(x)$, that

$$0 = \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_2(y) \right)$$

Because the RHS vanishes on all $x \in \{0,1\}^s$, for β sampled by the verifier, we have that

$$\begin{aligned}
0 &= \sum_{x \in \{0,1\}^{s}} \widetilde{eq}(\beta, x) \cdot \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_2(y) \right) \\
&= \sum_{x \in \{0,1\}^{s}} Q(x) && \text{By construction.}
\end{aligned}$$

Therefore, for γ sampled by the verifier, by linearity, we have that

$$\begin{aligned}\sum_{j\in[t]} \gamma^j \cdot v_j &= \sum_{x\in\{0,1\}^s} \left(\left(\sum_{j\in[t]} \gamma^j \cdot L_j(x) \right) + \gamma^{t+1} \cdot Q(x) \right) \\ &= \sum_{x\in\{0,1\}^s} g(x) && \text{By construction.}\end{aligned}$$

Therefore, by the perfect completeness of the sum-check protocol, we have for $e_1 = \widetilde{eq}(r_x, r'_x)$, $e_2 = \widetilde{eq}(\beta, r'_x)$ and

$$\sigma_i = \sum_{y\in\{0,1\}^{s'}} \widetilde{M_i}(r'_x, y) \cdot \widetilde{z}_1(y) \quad \text{and} \quad \theta_i = \sum_{y\in\{0,1\}^{s'}} \widetilde{M_i}(r'_x, y) \cdot \widetilde{z}_2(y)$$

that

$$\begin{aligned} c &= g(r'_x) \\ &= \left(\left(\sum_{j\in[t]} \gamma^j \cdot L_j(r'_x) \right) + \gamma^{t+1} \cdot Q(r'_x) \right) \\ &= \left(\left(\sum_{j\in[t]} \gamma^j \cdot e_1 \cdot \sigma_j \right) + \gamma^{t+1} \cdot e_2 \sum_{i\in[q]} c_i \cdot \prod_{j\in S_i} \theta_j \right). \end{aligned}$$

This implies that the verifier will not abort on step 5.

By construction, the prover can construct $\mathsf{u}_{\mathsf{CRR1CS}}$ such that the verifier does not abort on step 8. Furthermore, the prover can construct $\mathsf{u}_{\mathsf{CRR1CS}}$ such that $\mathsf{w}_{\mathsf{CRR1CS}}$ is a satisfying witness under structure s_{EC}. This implies that $C' = C_1 + \rho \cdot C_2$, where C' is parsed from x_2, which is the public IO of $\mathsf{u}_{\mathsf{CRR1CS}}$.

Now, consider the linearized CCS instance

$$(C_2, 1, \mathsf{x}_2, r'_x, \theta_1, \ldots, \theta_t).$$

By the precondition that the committed CCS instance (C_2, x_2) is satisfied by $\widetilde{w}_2$ and by the definition of $\theta_1, \ldots, \theta_t$ we have that this linearized CCS instance is satisfied by the witness $\widetilde{w}_2$.

Therefore, for random ρ sampled by the verifier, and for $C' = C_1 + \rho \cdot C_2$, $u' = u + \rho \cdot 1$, $\mathsf{x}' = \mathsf{x}_1 + \rho \cdot \mathsf{x}_2$, $v'_i = \sigma_i + \rho \cdot \theta_i$, we have that the output linearized CCS instance

$$(C', u', \mathsf{x}', r'_x, v'_1, \ldots, v'_t)$$

is satisfied by the witness $\widetilde{w}' = \widetilde{w}_1 + \rho \cdot \widetilde{w}_2$ by the linearity and the additive homomorphism property of the commitment scheme.

Now, we argue that the the output committed relaxed R1CS instance $(\overline{E}^\star, u^\star, \overline{W}^\star, x^\star)$ is satisfying under the witness $(E^\star, W^\star)$, for relaxed R1CS structure $\mathsf{s_{EC}} = (A, B, C)$. We need to establish the following. Let $Z^\star = (W^\star, u^\star, x^\star)$.

$$AZ^\star \circ BZ^\star = u^\star \cdot CZ^\star + E^\star \tag{19}$$

$$\overline{W}^\star = \mathsf{VC.Commit}(\mathsf{pp_{VC}}, W^\star) \tag{20}$$

$$\overline{E}^\star = \mathsf{VC.Commit}(\mathsf{pp_{VC}}, E^\star) \tag{21}$$

The latter two requirements hold from the additive homomorphism of the commitment scheme. We now focus on proving the first requirement. We are given that the input committed relaxed R1CS instance $(\overline{E}_1, u_1, \overline{W}_1, x_1)$ is satisfying under the witness (E_1, W_1) and structure $\mathsf{s_{EC}}$. This implies that

$$AZ_1 \circ BZ_2 = u_1 \cdot CZ_1 + E_1,$$

where $Z_1 = (W_1, u_1, x_1)$. As noted above, the committed relaxed R1CS instance sent by the prover $(\overline{E}_2, u_2, \overline{W}_2, x_2)$ is satisfying under the witness (E_2, W_2) and structure $\mathsf{s_{EC}}$. This implies that

$$AZ_2 \circ BZ_2 = CZ_2,$$

where $Z_2 = (W_2, 1, x_2)$. (This is because by construction $u_2 = 1$ and $E_2 = 0$.)

Now, consider the LHS of the desired equality.

$$\begin{aligned} AZ^\star \circ BZ^\star &= A(Z_1 + \rho^\star \cdot Z_2) \circ B(Z_1 + \rho^\star \cdot Z_2) \\ &= AZ_1 \circ BZ_1 + \rho^\star \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + (\rho^\star)^2 \cdot (AZ_2 \circ BZ_2) \\ &= u_1 \cdot CZ_1 + E_1 + \rho^\star \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + (\rho^\star)^2 \cdot CZ_2 \end{aligned}$$

Consider the RHS of the desired equality.

$$\begin{aligned} u^\star \cdot CZ^\star + E^\star &= (u_1 + \rho^\star) \cdot C(Z_1 + \rho^\star \cdot Z_2) + E_1 + \rho^\star \cdot T \\ &= (u_1 + \rho^\star) \cdot (CZ_1 + \rho^\star \cdot CZ_2) + E_1 + \rho^\star \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1 - u_1 \cdot CZ_2 - CZ_1) \\ &= u_1 \cdot CZ_1 + \rho^\star \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + (\rho^\star)^2 \cdot CZ_2 \end{aligned}$$

This establishes the desired requirements. □

Lemma 19 (Knowledge Soundness). *Construction 7 satisfies knowledge soundness.*

Proof. Consider an adversary $\mathcal{A}$ that adaptively picks the structure and instances, and a malicious prover $\mathcal{P}^*$ that succeeds with probability ϵ. Let $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N)$. Suppose on input pp and random tape r, the adversary $\mathcal{A}$ picks a structure $(\mathsf{s}_1, \mathsf{s}_2) = (((\widetilde{M}_1, \ldots, \widetilde{M}_t), (S_1, \ldots, S_q), (c_1, \ldots, c_q)), (A, B, C))$ such

that $\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = \mathsf{true}$, a pair of linearized committed CCS instance and a committed relaxed R1CS instance

$$\varphi_1 = ((C_1, u, \mathsf{x}_1, r_x, v_1, \ldots, v_t), (\overline{E}_1, u_1, \overline{W}_1, x_1))$$

and a committed CCS instance

$$\varphi_2 = (C_2, \mathsf{x}_2),$$

and some auxiliary state st.

We construct an expected-polynomial time extractor $\mathcal{E}$ that succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$ in obtaining satisfying witnesses for the original instances as follows. Below, let $\mathcal{R}_1 = \mathcal{R}_{\mathsf{LCCCS}} \times \mathcal{R}_{\mathsf{CRR1CS}}$ and $\mathcal{R}_2 = \mathcal{R}_{\mathsf{CCCS}}$.

$\underline{\mathcal{E}(\mathsf{pp}, \mathsf{r})}$:

1. Obtain the output tuple from $\mathcal{A}$:

$$(\mathsf{s}, \varphi_1, \varphi_2, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{r}).$$

2. Compute $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$.
3. Run the interaction

$$(\varphi^{(1,1)}, (\widetilde{w}, (E, W))^{(1,1)}) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle ((\mathsf{pk}, \mathsf{vk}), \varphi_1, \varphi_2, \mathsf{st})$$

   once with the verifier's final challenges $\rho^{(1)} \xleftarrow{\$} \mathbb{F}$ and $\rho^{\star(1,1)} \xleftarrow{\$} \mathbb{F}$.
4. Abort if $(\mathsf{pp}, \mathsf{s}, \varphi^{(1,1)}, (\widetilde{w}, (E, W))^{(1,1)}) \notin \mathcal{R}_1$.
5. Rewind the interaction

$$(\varphi^{(1,2)}, (\widetilde{w}, (E, W))^{(1,2)}) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle ((\mathsf{pk}, \mathsf{vk}), \varphi_1, \varphi_2, \mathsf{st})$$

   with a different verifier's challenge $\rho^{\star(2,1)} \xleftarrow{\$} \mathbb{F}$ while maintaining the same prior randomness. Repeat until $(\mathsf{pp}, \mathsf{s}, \varphi^{(1,2)}, (\widetilde{w}, (E, W))^{(1,2)}) \in \mathcal{R}_1$.
6. Rewind the interaction

$$(\varphi^{(2,1)}, (\widetilde{w}, (E, W))^{(2,1)}) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle ((\mathsf{pk}, \mathsf{vk}), \varphi_1, \varphi_2, \mathsf{st})$$

   with different verifier's challenges $\rho^{(2)} \xleftarrow{\$} \mathbb{F}$ and $\rho^{\star(2,1)} \xleftarrow{\$} \mathbb{F}$ while maintaining the same prior randomness. Repeat until $(\mathsf{pp}, \mathsf{s}, \varphi^{(2,1)}, (\widetilde{w}, (E, W))^{(2,1)}) \in \mathcal{R}_1$.
7. Rewind the interaction

$$(\varphi^{(2,2)}, (\widetilde{w}, (E, W))^{(2,2)}) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle ((\mathsf{pk}, \mathsf{vk}), \varphi_1, \varphi_2, \mathsf{st})$$

   with a different verifier's challenge $\rho^{\star(2,2)} \xleftarrow{\$} \mathbb{F}$ while maintaining the same prior randomness. Repeat until $(\mathsf{pp}, \mathsf{s}, \varphi^{(2,1)}, (\widetilde{w}, (E, W))^{(2,2)}) \in \mathcal{R}_1$.

8. Abort if $\rho^{\star(1,1)} = \rho^{\star(1,2)}$, $\rho^{(1)} = \rho^{(2)}$, or $\rho^{\star(2,1)} = \rho^{\star(2,2)}$.
9. Interpolating points $(\rho^{(1)}, \widetilde{w}^{(1,1)})$ and $(\rho^{(2)}, \widetilde{w}^{(2,1)})$, retrieve the witness polynomials $\widetilde{w}_1$ and $\widetilde{w}_2$ such that for $i \in \{1, 2\}$

$$\widetilde{w}_1 + \rho^{(i)} \cdot \widetilde{w}_2 = \widetilde{w}^{(i,1)}. \tag{22}$$

10. Interpolating points $(\rho^{\star(1,1)}, (E, W)^{(1,1)})$ and $(\rho^{\star(1,2)}, (E, W)^{(1,2)})$, retrieve (E_1, W_1) and (T, W_2) such that for $j \in \{1, 2\}$

$$E_1 + \rho^{\star(1,j)} \cdot T = E^{(1,j)} \tag{23}$$

$$W_1 + \rho^{\star(1,j)} \cdot W_2 = W^{(1,j)} \tag{24}$$

11. Output $((\widetilde{w_1}, (E_1, W_1)), \widetilde{w_2})$.

We first demonstrate that the extractor $\mathcal{E}$ runs in expected polynomial time. Observe that $\mathcal{E}$ runs the interaction once, and if it does not abort, keeps rerunning the interaction until $\mathcal{P}^*$ succeeds three additional times. Thus, the expected number of times $\mathcal{E}$ runs the interaction is

$$1 + \Pr[\text{First call to } \langle \mathcal{P}^*, \mathcal{V} \rangle \text{ succeeds}] \cdot \frac{3}{\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle \text{ succeeds}]} = 1 + \epsilon \cdot \frac{3}{\epsilon} = 4.$$

Therefore, we have that the extractor runs in expected polynomial-time.

We now analyze $\mathcal{E}$'s success probability. We must demonstrate that $\mathcal{E}$ succeeds in producing $(\widetilde{w}_1, (E_1, W_1))$ and $\widetilde{w}_2$ such that

$$(\mathsf{pp}, \mathsf{s}, \varphi_1, (\widetilde{w}_1, (E_1, W_1))) \in \mathcal{R}_1 \quad \text{and} \quad (\mathsf{pp}, \mathsf{s}_1, \varphi_2, \widetilde{w}_2) \in \mathcal{R}_2$$

with probability $\epsilon - \mathsf{negl}(\lambda)$.

To do so, we first show that the extractor successfully produces *some* output (i.e., does not abort) in under $|\mathbb{F}|$ rewinding steps with probability $\epsilon - \mathsf{negl}(\lambda)$. Note that $|\mathbb{F}|$ is a worst case bound and we have already established that the extractor runs in expected polynomial time. By the malicious prover's success probability, we have that the extractor does not abort in step (4) with probability ϵ. Given that the extractor does not abort in step (4), by Markov's inequality, we have that the extractor rewinds more than $|\mathbb{F}|$ times with probability $4/|\mathbb{F}|$. Thus, the probability that the extractor does not abort in step (4) and requires less than $|\mathbb{F}|$ rewinds is $\epsilon \cdot (1 - 4/|\mathbb{F}|)$.

Now, suppose that the extractor does not abort in step (4). Then, because the extractor randomly samples $\rho^{\star(1,2)}$, we have that $\rho^{\star(1,1)} \neq \rho^{\star(1,2)}$ with probability $1/|\mathbb{F}|$. Similarly, we have that, $\rho^{(1)} \neq \rho^{(2)}$ with probability $1/|\mathbb{F}|$ and $\rho^{\star(2,1)} = \rho^{\star(2,2)}$ with probability $1/|\mathbb{F}|$. Thus, we have that the probability the extractor successfully produces some output in under $|\mathbb{F}|$ rewinding steps is

$$\epsilon \cdot \left(1 - \frac{4}{|\mathbb{F}|}\right) \cdot \left(1 - \frac{3}{|\mathbb{F}|}\right) = \epsilon - \mathsf{negl}(\lambda).$$

Next, if the extractor does not abort, we show that the extractor succeeds in producing satisfying witnesses with probability $1 - \mathsf{negl}(\lambda)$. This brings the overall extractor success probability to $\epsilon - \mathsf{negl}(\lambda)$.

We first show that the in the transcripts retrieved, the output witnesses for linearized committed CCS instances do not depend on the choice of $\rho^\star$. More precisely, we show that, for $i \in \{1, 2\}$, $\widetilde{w}^{(i,1)} = \widetilde{w}^{(i,2)}$.

For $i \in \{1, 2\}$ and $j \in \{1, 2\}$, let

$$\varphi^{(i,j)} = ((C^{(i,j)}, u^{(i,j)}, \mathsf{x}^{(i,j)}, r_x^{(i,j)}, v_1^{(i,j)}, \ldots, v_t^{(i,j)}), (\overline{E}^{(i,j)}, u^{\star(i,j)}, \overline{W}^{(i,j)}, x^{\star(i,j)})).$$

By the verifier's construction and because the transcripts share the same prefix prior to the choice of $\rho^\star$, we have for $i \in \{1, 2\}$ that

$$(C^{(i,1)}, u^{(i,1)}, \mathsf{x}^{(i,1)}, r_x^{(i,1)}, v_1^{(i,1)}, \ldots, v_t^{(i,1)}) = (C^{(i,2)}, u^{(i,2)}, \mathsf{x}^{(i,2)}, r_x^{(i,2)}, v_1^{(i,2)}, \ldots, v_t^{(i,2)}). \tag{25}$$

We are given that for $i \in \{1, 2\}$ and $j \in \{1, 2\}$, $\widetilde{w}^{(i,j)}$ is a satisfying witness and hence a valid opening of the commitment $C^{(i,j)}$. By Equation 25, we have that for $i \in \{1, 2\}$, $C^{(i,1)} = C^{(i,2)}$. Therefore, by the binding property of the polynomial commitment scheme, with probability $1 - \mathsf{negl}(\lambda)$, we have for $i \in \{1, 2\}$ that

$$\widetilde{w}^{(i,1)} = \widetilde{w}^{(i,2)}. \tag{26}$$

Given this equality of commitments and the associated witnesses for the output linearized committed CCS instances, we drop the second index when appropriate.

We now show that the retrieved polynomials and vectors $((\widetilde{w}_1, (E_1, W_1)), \widetilde{w}_2)$ are valid openings to the corresponding commitments in the instance.

For $j \in \{1, 2\}$, because $(E, W)^{(1,j)}$ is a satisfying witness to the folded committed relaxed R1CS instance, by construction,

$$\begin{aligned}
&\mathsf{Commit}(\mathsf{pp}_{\mathsf{VC}}, W_1) + \rho^{\star(1,j)} \cdot \mathsf{Commit}(\mathsf{pp}_{\mathsf{VC}}, W_2) \\
&= \mathsf{Commit}(\mathsf{pp}_{\mathsf{VC}}, W_1 + \rho^{\star(1,j)} \cdot W_2) && \text{By additive homomorphism.} \\
&= \mathsf{Commit}(\mathsf{pp}_{\mathsf{VC}}, W^{(1,j)}) && \text{By Equation (24).} \\
&= \overline{W}^{(1,j)} && \text{Witness } \widetilde{W}^{(1,j)} \text{ is a satisfying opening.} \\
&= \overline{W}_1 + \rho^{\star(1,j)} \cdot \overline{W}_2 && \text{By the verifier's computation.}
\end{aligned}$$

Interpolating, we have that

$$\mathsf{Commit}(\mathsf{pp}_{\mathsf{VC}}, W_1) = \overline{W}_1 \tag{27}$$

$$\mathsf{Commit}(\mathsf{pp}_{\mathsf{VC}}, W_2) = \overline{W}_2 \tag{28}$$

Similarly,

$$
\begin{aligned}
&\mathsf{Commit}(\mathsf{pp}_{\mathsf{VC}}, E_1) + \rho^{\star(1,j)} \cdot \mathsf{Commit}(\mathsf{pp}_{\mathsf{VC}}, T) \\
&= \mathsf{Commit}(\mathsf{pp}_{\mathsf{VC}}, E_1 + \rho^{\star(1,j)} \cdot T) && \text{By additive homomorphism.} \\
&= \mathsf{Commit}(\mathsf{pp}_{\mathsf{VC}}, E^{(1,j)}) && \text{By Equation (23).} \\
&= \overline{E}^{(1,j)} && \text{Witness } \widetilde{E}^{(1,j)} \text{ is a satisfying opening.} \\
&= \overline{E}_1 + \rho^{\star(1,j)} \cdot \overline{T} && \text{By the verifier's computation.}
\end{aligned}
$$

Interpolating, we have that

$$\mathsf{Commit}(\mathsf{pp}_{\mathsf{VC}}, E_1) = \overline{E}_1 \tag{29}$$

For $j \in \{1, 2\}$, because $(E, W)^{(1,j)}$ is a satisfying witness to the committed relaxed R1CS instance $(\overline{E}^{(1,j)}, u^{\star(1,j)}, \overline{W}^{(1,j)}, x^{\star(1,j)})$, we have the following, where $Z^{(1,j)} = (W^{(1,j)}, u^{\star(1,j)}, x^{\star(1,j)})$.

$$AZ^{(1,j)} \circ BZ^{(1,j)} = u^{\star(1,j)} \cdot CZ^{(1,j)} + E^{(1,j)}$$

By Equation (24), this implies that for $j \in \{1, 2\}$

$$
\begin{aligned}
&A \cdot (Z_1 + \rho^{\star(1,j)} \cdot Z_2) \circ B \cdot (Z_1 + \rho^{\star(1,j)} \cdot Z_2) \\
&= (u_1 + \rho^{\star(1,j)}) \cdot C \cdot (Z_1 + \rho^{\star(1,j)} \cdot Z_2) + (E_1 + \rho^{\star(1,j)} \cdot T),
\end{aligned}
$$

where $Z_1 = (W_1, u_1, x_1)$, $Z_2 = (W_2, 1, x_2)$, and x_2 is parsed from the transcripts and is identical across the two executions with the same ρ.

Because the prover commits to W_1, W_2, and T before the verifier sends the challenge $\rho^{\star(1,j)}$, we have with probability $1 - \mathsf{negl}(\lambda)$ that

$$AZ_1 \circ BZ_1 = u_1 \cdot CZ_1 + E_1 \tag{30}$$

$$AZ_2 \circ BZ_2 = CZ_2. \tag{31}$$

This implies that (E_1, W_1) and $(\vec{0}, W_2)$ meet the requirements of a satisfying witness for committed relaxed R1CS instances with structure (A, B, C). In particular, we have established that (E_1, W_1) is a satisfying witness to the committed relaxed R1CS instance in φ_1.

Furthermore, since the verifier checks that $x_2 = (\rho^{(1)}, C_1, C_2, C')$ for some $C' \in \mathbb{G}_1$, given that the we have have a witness satisfying Equation 31, this implies that for $j \in \{1, 2\}$

$$C^{(1,j)} = C_1 + \rho^{(1)} \cdot C_2 \tag{32}$$

With a similar reasoning via the accepting transcripts with $\rho^{(2)}$ as the verifier's randomness, we can establish that for $j \in \{1, 2\}$:

$$C^{(2,j)} = C_1 + \rho^{(2)} \cdot C_2 \tag{33}$$

For $i \in \{1,2\}$ and $j \in \{1,2\}$, because $\widetilde{w}^{(i,j)}$ is a satisfying witness to the folded linearized CCS instance, by construction,

$$
\begin{aligned}
&\mathsf{Commit}(\mathsf{pp}_{\mathsf{PC}}, \widetilde{w}_1) + \rho^{(i)} \cdot \mathsf{Commit}(\mathsf{pp}_{\mathsf{PC}}, \widetilde{w}_2) \\
&= \mathsf{Commit}(\mathsf{pp}_{\mathsf{PC}}, \widetilde{w}_1 + \rho^{(i)} \cdot \widetilde{w}_2) && \text{By additive homomorphism.} \\
&= \mathsf{Commit}(\mathsf{pp}_{\mathsf{PC}}, \widetilde{w}^{(i,j)}) && \text{By Equations (22) and (26).} \\
&= C^{(i,j)} && \text{Witness } \widetilde{w}^{(i,j)} \text{ is a satisfying opening.} \\
&= C_1 + \rho^{(i)} \cdot C_2 && \text{By Equations 32 and 33}
\end{aligned}
$$

Interpolating, we have that

$$\mathsf{Commit}(\mathsf{pp}_{\mathsf{PC}}, \widetilde{w}_1) = C_1 \tag{34}$$
$$\mathsf{Commit}(\mathsf{pp}_{\mathsf{PC}}, \widetilde{w}_2) = C_2. \tag{35}$$

Next, we must argue that $\widetilde{w}_1$ and $\widetilde{w}_2$ satisfy the remainder of the instances φ_1 and φ_2 respectively under the structure s.

Indeed, consider $(\sigma_1, \ldots, \sigma_t)$ and $(\theta_1, \ldots, \theta_t)$ sent by the prover which by the extractor's construction are identical across all executions of the interaction. By the verifier's computation we have that for $i \in \{1,2\}$ and all $j \in [t]$

$$\sigma_j + \rho^{(i)} \cdot \theta_j = v_j^{(i)} \tag{36}$$

Now, because $\widetilde{w}^{(i)}$ is a satisfying witness, for $i \in \{1,2\}$ we have for all $j \in [t]$ that

$$v_j^{(i)} = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z}^{(i)}(y),$$

where $\widetilde{z}^{(i)} = \widetilde{(w^{(i)}, u^{(i)}, \mathsf{x}^{(i)})}$.

However, by Equations (22) and (36), for $i \in \{1,2\}$ and $j \in [t]$, this implies that

$$\sigma_j + \rho^{(i)} \cdot \theta_j = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z_1}(y) + \rho^{(i)} \cdot \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z_2}(y),$$

where $\widetilde{z_1} = \widetilde{(w_1, u, \mathsf{x}_1)}$ and $\widetilde{z_2} = \widetilde{(w_2, 1, \mathsf{x}_2)}$. Interpolating, we have that, for all $j \in [t]$

$$\sigma_j = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z_1}(y)$$
$$\theta_j = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z_2}(y)$$

Thus, because that the verifier does not abort, we have that

$$\begin{aligned}
c &= \left(\sum_{j\in t} \gamma^j \cdot e_1 \cdot \sigma_j\right) + \left(\gamma^{t+1} \cdot e_2 \cdot \sum_{i\in[q]} c_i \cdot \prod_{j\in S_i} \theta_j\right) \\
&= \left(\sum_{j\in t} \gamma^j \cdot \widetilde{eq}(r_x, r'_x) \cdot \sigma_j\right) + \left(\gamma^{t+1} \cdot \widetilde{eq}(\beta, r'_x) \cdot \sum_{i\in[q]} c_i \cdot \prod_{j\in S_i} \theta_j\right) \\
&= \left(\sum_{j\in t} \gamma^j \cdot \widetilde{eq}(r_x, r'_x) \cdot \sum_{y\in\{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z}_1(y)\right) + \\
&\quad \left(\gamma^{t+1} \cdot \widetilde{eq}(\beta, r'_x) \cdot \sum_{i\in[q]} c_i \cdot \prod_{j\in S_i} \sum_{y\in\{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z}_2(y)\right) \\
&= \sum_{j\in[t]} \gamma_j \cdot L_j(r'_x) + \gamma^{t+1} \cdot Q(r'_x) \\
&= g(r'_x)
\end{aligned}$$

by the soundness of the sum-check protocol, this implies that with probability $1 - O(d \cdot s)/|\mathbb{F}| = 1 - \mathsf{negl}(\lambda)$ over the choice of r'_x,

$$\begin{aligned}
\sum_{j\in[t]} \gamma^j \cdot v_j + \gamma^{t+1} \cdot 0 &= \sum_{x\in\{0,1\}^s} g(x) \\
&= \sum_{x\in\{0,1\}^s} \left(\sum_{j\in[t]} \gamma^j \cdot L_j(x) + \gamma^{t+1} \cdot Q(x)\right) \\
&= \sum_{j\in[t]} \gamma^j \cdot \left(\sum_{x\in\{0,1\}^s} L_j(x)\right) + \gamma^{t+1} \cdot \sum_{x\in\{0,1\}^s} Q(x)
\end{aligned}$$

By the Schwartz-Zippel lemma [56], this implies that with probability $1 - O(t)/|\mathbb{F}| = 1 - \mathsf{negl}(\lambda)$ over the choice of γ, we have

$$v_j = \sum_{x\in\{0,1\}^s} L_j(x)$$

for all $j \in [t]$ and

$$0 = \sum_{x\in\{0,1\}^s} Q(x).$$

Now, for all $j \in [t]$, we have

$$\begin{aligned} v_j &= \sum_{x \in \{0,1\}^s} L_j(x) \\ &= \sum_{x \in \{0,1\}^s} \widetilde{eq}(r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} M_j(x,y) \cdot \widetilde{z}_1(y) \right) \\ &= \sum_{y \in \{0,1\}^{s'}} M_j(r_x, y) \cdot \widetilde{z}_1(y) \end{aligned}$$

This implies that $\widetilde{w_1}$ is a satisfying witness to the linearized committed CCS instance in φ_1.

Finally, we have that

$$\begin{aligned} 0 &= \sum_{x \in \{0,1\}^s} Q(x) \\ &= \sum_{x \in \{0,1\}^s} \widetilde{eq}(\beta, x) \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(x,y) \cdot \widetilde{z}_2(y) \right) \right) \\ &= \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(\beta, y) \cdot \widetilde{z}_2(y) \right) \end{aligned}$$

By the Schwartz-Zippel lemma, this implies that with probability $1 - s/|\mathbb{F}| = 1 - \mathsf{negl}(\lambda)$ over the choice of β, we have that for all $x \in \{0,1\}^s$

$$0 = \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(x,y) \cdot \widetilde{z}_2(y) \right)$$

This implies that $\widetilde{w_2}$ is a satisfying witness to φ_2.

Thus, if the extractor does not abort, it succeeds in producing satisfying witness $(\widetilde{w_1}, \widetilde{w_2})$ with probability $1 - \mathsf{negl}(\lambda)$. □

H.5 Proof of Theorem 8 (nlookup)

Lemma 20 (Perfect Completeness). *Construction 8 satisfies perfect completeness.*

Proof. Consider public parameters $\mathsf{pp} = \ell \leftarrow \mathcal{G}(1^\lambda, N)$. Consider a common structure $\mathsf{s}_1 = \mathsf{s}_2 = \widetilde{T} \in \mathbb{F}[X_1, \ldots, X_\ell]$. Consider the prover and verifier keys

$(\mathsf{pk}, \mathsf{vk}) = (\widetilde{T}, \perp) \leftarrow \mathcal{K}(\mathsf{pp}, \widetilde{T})$. Suppose that the prover and the verifier are provided an instance in $\mathcal{R}_{\mathsf{poly}}$

$$(q_r, v_r)$$

and a vector of $\mathcal{R}_{\mathsf{lookup}}$ instances

$$(v_1, \ldots, v_m).$$

Suppose that the prover is additionally provided with the corresponding satisfying witnesses for the $\mathcal{R}_{\mathsf{lookup}}$ instances

$$(q_1, \ldots, q_m).$$

By the satisfiability of the input instances we have that $v_r = \widetilde{T}(q_r)$ and $v_i = \widetilde{T}(q_i)$ for all $i \in [m]$.

Therefore, for $\rho \in \mathbb{F}$, we have that

$$\begin{aligned}
v_r + \sum_{i \in [m]} \rho^i \cdot v_i &= \widetilde{T}(q_r) + \sum_{i \in [m]} \rho^i \cdot \widetilde{T}(q_i) && \text{By precondition.} \\
&= \sum_{x \in \{0,1\}^\ell} \left(\widetilde{eq}(q_r, x) \cdot \widetilde{T}(x) + \sum_{i \in [m]} \rho^i \cdot \widetilde{eq}(q_i, x) \cdot \widetilde{T}(x) \right) && \text{By Lemma 6.} \\
&= g(x) && \text{By definition.}
\end{aligned}$$

Therefore, by the perfect completeness of the sum-check protocol, we have that $c = g(q'_r)$ Thus, for $v'_r = \widetilde{T}(q'_r)$, $e = \widetilde{eq}(q_r, q'_r)$, and $e_i = \widetilde{eq}(q_i, q'_r)$ for all $i \in [m]$, we have that

$$\begin{aligned}
c &= g(q'_r) \\
&= \widetilde{eq}(q_r, q'_r) \cdot \widetilde{T}(q'_r) + \sum_{i \in [m]} \rho^i \cdot \widetilde{eq}(q_i, q'_r) \cdot \widetilde{T}(q'_r) \\
&= e \cdot v'_r + \sum_{i \in [m]} \rho^i \cdot e_i \cdot v'_r
\end{aligned}$$

Therefore, we have that the verifier does not abort.

By construction, we have that $v'_r = \widetilde{T}(q'_r)$. Therefore, the folded polynomial evaluation instance is satisfying.

□

Lemma 21 (Knowledge Soundness). *Construction 8 satisfies knowledge soundness assuming that* $|\mathbb{F}| = \Theta(2^\lambda)$.

Proof. Consider an adversary $\mathcal{A}$ that adaptively picks the structure and instances, and a malicious prover $\mathcal{P}^*$ that succeeds with probability ϵ. Let $\mathsf{pp} = \ell \leftarrow$

$\mathcal{G}(1^\lambda, N)$. Suppose on input pp and random tape r, the adversary $\mathcal{A}$ picks a structure $\mathsf{s}_1 = \mathsf{s}_2 = \widetilde{T} \in \mathbb{F}[X_1, \ldots, X_\ell]$, an $\mathcal{R}_{\mathsf{poly}}$ instance (q_r, v_r), a vector of $\mathcal{R}_{\mathsf{lookup}}$ instances $(v_1, \ldots, v_m)$, and some auxiliary state st.

We construct an extractor $\mathcal{E}$ that succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$ in obtaining satisfying witnesses for the original instances. It works as follows.

On input pp and r, $\mathcal{E}$ first obtains the following tuple from $\mathcal{A}$:

$$(\widetilde{T}, (q_r, v_r), (v_1, \ldots, v_m), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{r}).$$

$\mathcal{E}$ then computes $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \widetilde{T})$. Next, $\mathcal{E}$ runs

$$((q_r', v_r'), \bot) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle ((\mathsf{pk}, \mathsf{vk}), (q_r, v_r), (v_1, \ldots, v_m), \mathsf{st})$$

and obtains the first message $(q_1, \ldots, q_m)$ from $\mathcal{P}^*$ by parsing the corresponding transcript. The extractor $\mathcal{E}$ outputs $(\bot, (q_1, \ldots, q_m))$ as the witness. Because the extractor only runs $\mathcal{P}^*$ once, it runs in expected polynomial-time.

We must argue that $\bot$ is a satisfying $\mathcal{R}_{\mathsf{poly}}$ witness for (q_r, v_r), and that $(q_1, \ldots, q_m)$ are satisfying $\mathcal{R}_{\mathsf{lookup}}$ witnesses for the input instances $(v_1, \ldots, v_m)$ with probability $\epsilon - \mathsf{negl}(\lambda)$.

Suppose that we have that the witness $\bot$ output by $\mathcal{P}^*$ is satisfying for the corresponding verifier's output (q_r', v_r') with probability ϵ. By definition, this means that

$$v_r' = \widetilde{T}(q_r') \tag{37}$$

with probability ϵ. Moreover, this means that the verifier does not abort with probability at least ϵ, and thus we have the following:

$$\begin{aligned}
c &= e \cdot v_r' + \sum_{i \in [m]} \rho^i \cdot e_i \cdot v_r' \\
&= \widetilde{eq}(q_r, q_r') \cdot v_r' + \sum_{i \in [m]} \rho^i \cdot \widetilde{eq}(q_i, q_r') \cdot v_r' && \text{By the verifier's computation.} \\
&= \widetilde{eq}(q_r, q_r') \cdot \widetilde{T}(q_r') + \sum_{i \in [m]} \rho^i \cdot \widetilde{eq}(q_i, q_r') \cdot \widetilde{T}(q_r') && \text{By Equation 37.} \\
&= g(q_r') && \text{By definition.}
\end{aligned}$$

with probability ϵ.

Then, by the soundness of the sum-check protocol, we must have that

$$\begin{aligned}
v_r + \sum_{i \in [m]} \rho^i \cdot v_i &= \sum_{x \in \{0,1\}^\ell} g(x) \\
&= \sum_{x \in \{0,1\}^\ell} \left(\widetilde{eq}(q_r, x) \cdot \widetilde{T}(x) + \sum_{i \in [m]} \rho^i \cdot \widetilde{eq}(q_i, x) \cdot \widetilde{T}(x) \right) && \text{By definition.} \\
&= \widetilde{T}(q_r) + \sum_{i \in [m]} \rho^i \cdot \widetilde{T}(q_i) && \text{By Lemma 6.}
\end{aligned}$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. By the Schwartz-Zippel lemma over ρ, this implies that $v_r = \widetilde{T}(q_r)$ and $v_i = \widetilde{T}(q_i)$ for all $i \in [m]$ with probability $\epsilon - \mathsf{negl}(\lambda)$.

Moreover, by the verifier's initial check, we have that $q_i \in \{0,1\}^\ell$ for all $i \in [m]$. Therefore, we have that

$$(\mathsf{pp}, \widetilde{T}, (q_r, v_r), \bot) \in \mathcal{R}_{\mathsf{poly}}$$

and

$$(\mathsf{pp}, \widetilde{T}, (v_1, \ldots, v_m), (q_1, \ldots, q_m)) \in \mathcal{R}^{(m)}_{\mathsf{lookup}}.$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. Because the extractor $\mathcal{E}$ outputs the initial message from the prover, we have that the extractor succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$. $\square$