MicroNova: Folding-based arguments with efficient (on-chain) verification

Jiaxing Zhao[†⋆], Srinath Setty[⋆], Weidong Cui[⋆], and Greg Zaverucha[⋆]
[†]University of Science and Technology of China [⋆]Microsoft Research

***Abstract*—We describe the design and implementation of MicroNova, a folding-based recursive argument for producing proofs of incremental computations of the form $y = F^{(\ell)}(x)$, where F is a possibly non-deterministic computation (encoded using a constraint system such as R1CS), x is the initial input, y is the output, and $\ell > 0$. The proof of an ℓ-step computation is produced step-by-step such that the proof size nor the time to verify it depends on ℓ. The proof at the final iteration is then compressed, to achieve further succinctness in terms of proof size and verification time. Compared to prior folding-based arguments, a distinguishing aspect of MicroNova is the concrete efficiency of the verifier—even in a resource-constrained environment such as Ethereum's blockchain. In particular, the compressed proof consists of $O(\log N)$ group elements and it can be verified with $O(\log N)$ group scalar multiplications and two pairing operations, where N is the number of constraints for a single invocation of F. MicroNova requires a universal trusted setup and can employ *any* existing setup material created for the popular KZG univariate polynomial commitment scheme. Finally, we implement and experimentally evaluate MicroNova. We find that MicroNova's proofs can be efficiently verified on the Ethereum blockchain with ≈2.2M gas. Furthermore, MicroNova's prover incurs minimal overheads atop its baseline Nova's prover.**

1. Introduction

A folding scheme [52] is a cryptographic protocol involving a *prover* and a *verifier*, where they interactively reduce the task of checking two NP instances into the task of checking a single NP instance. A folding scheme can be used on its own to reduce the prover's work when proving multiple instances. Beyond this, Kothapalli et al. [50], [52] show that folding schemes that satisfy certain mild requirements immediately provide an efficient realization of incrementally verifiable computation (IVC) [66], a powerful cryptographic primitive that enables one to prove the correct execution of a "long running" computation in an incremental manner.

Suppose we have an incremental computation of the form $z_i = F(\omega_i, z_{i-1})$, where $i \geq 1$, F is a function (typically represented using a circuit), ω_i is the non-deterministic input of the prover for step i, z_0 is the initial input, and z_i is the output after i steps. With an IVC scheme, the prover takes as input a proof π_i that proving that z_i is the correct output after executing i steps of the incremental computation, and then outputs a proof π_{i+1} that proves the correct execution of the first $i+1$ steps to produce an output z_{i+1}. Crucially, the prover's work (or the memory requirements to update a proof) nor the proof size grows with the number of steps executed thus far, and the verifier can verify any proof produced by the prover to verify the execution of the computation thus far. Thus, IVC unlocks proving the correct execution of large computations without the prover running out of memory.

IVC has a wide variety of applications in decentralized settings including rollups [53], [70], verifiable delay functions [17], [69], proofs of (virtual) machine executions (e.g., RISC-V). In all of these applications, the statement that the prover wishes to prove can be expressed in an incremental manner. For example, in a rollup, F checks the correct execution of a batch of blockchain transactions. In a VDF, F encodes checks that a delay function such as MinRoot [45] was executed correctly. In proofs of machine executions, F checks a certain number of iterations of the fetch-decode-execute loop of a particular machine.

Early works [15], [66] constructed IVC via succinct non-interactive arguments of knowledge (SNARKs) [13], [39], [46], [55]: the prover at step i proves the correct execution of the step computation F and that it has verified a SNARK π_{i-1} proving the correct execution of the first $i-1$ steps (the latter requires representing the SNARK verifier as a circuit alongside F). Folding schemes bypass the need for SNARKs (and even NARKs) by employing a particular type of reduction of knowledge [47]. Indeed, a folding scheme is simpler and more prover-efficient than a SNARK. Nova [52] provides a folding scheme for R1CS, a popular NP-complete problem that generalizes arithmetic circuit satisfiability. In this folding scheme, the prover's work at each step is dominated by two multi-scalar multiplications (MSMs) of sizes equal to the number of constraints and number of witness variables in R1CS expressing the step computation (this is an order of magnitude lower work compared to a SNARK prover [30], [38], [59]). Furthermore, the size of the folding scheme verifier proven at each step by the prover is constant sized (≈10,000 gates). Nova's IVC scheme is fully implemented and is public [6].

Unlike SNARK-based IVC, folding-scheme-based IVC produces IVC proofs whose size and time to verify scale linearly with $|F|$. For example, in Nova, an IVC proof is $\approx 3 \cdot |F|$ field elements (where each field element is ≈32 bytes). Even when F is of modest size such as a when F is represented with a million R1CS constraints, the proof size is ≈96 MB, and the verifier spends multiple seconds to verify a proof. To address this, Nova describes an approach to use SNARKs

to compress IVC proofs: the prover uses a SNARK (e.g., Spartan [59]) to prove the knowledge of a valid IVC proof. The resulting proofs are no longer incrementally updateable with a folding scheme, but this is acceptable in many target applications. In the rollup example, a verifier running on a blockchain merely requires a succinct proof of correct execution of a batch of transactions.[1]

1.1. Problem: Verifier times

A topic that has received significantly less attention—even with a flurry of recent works on folding schemes [7], [12], [18], [24], [27], [28], [32], [34], [35], [48]–[51]—is the verifier's costs in folding-based arguments. A folding scheme requires an additively homomorphic commitment scheme with succinct commitments for vectors. For this, Nova uses Pedersen's commitment instantiated over a cycle of elliptic curves. Concretely, Nova's implementation [6] uses the Pasta curves [8], which are simple and fast. Then, to compress IVC proofs [52, §5–6], Nova treats these (vector) commitments as commitments to multilinear polynomials in evaluation form over the Boolean hypercube. Nova then uses Spartan [59] with an IPA-based polynomial commitment scheme [20], [23] to prove the knowledge of a valid IVC proof.

Under the above scheme, for an N-sized step circuit, the compressed proofs are $O(\log N)$ group elements (e.g., a few KBs in practice), which is acceptable. However the time to verify a compressed proof is $O(N)$-sized MSM, where N is at least $10,000$ (given that Nova's folding scheme's verifier must be represented as a circuit at each step of the computation). For values of N that one may want to use, this requires the verifier runtime to be up to several seconds of CPU time. Also, it is well known that this verifier cannot feasibly be run on Ethereum's blockchain: the verifier incurs several billions gas, which is orders of magnitude higher gas than what is allowed by the block limit.

A solution to address this problem is to compose Nova with an existing SNARK (e.g., Plonk [38]) i.e., use Plonk to prove the knowledge of a valid Nova proof. Unfortunately, the resulting prover is impractical (§9). Our work can be viewed as an optimization of Nova's proof compression layer.

1.2. Our solution: MicroNova

MicroNova is a new IVC scheme with an efficient verifier for compressed IVC proofs. In particular, MicroNova contributes new techniques to improve verifier times in folding-based proof systems and to make the verifier suitable for on-chain verification (e.g., on Ethereum). MicroNova builds on Nova [52] given Nova is fully implemented and heavily optimized [6]. However, our techniques apply easily to other folding-scheme-based arguments (e.g., HyperNova [50], Protostar [24], Protogalaxy [34], NeutronNova [51]) to reduce verifier times and to achieve efficient on-chain verification. Many of MicroNova's techniques are of independent interest. For instance, we provide a variant of KZG that directly works with multilinear polynomials in the evaluation basis. We also provide a verifier-efficient version of Spartan [59], which we call MicroSpartan.

1. Constructing SNARKs via folding schemes is still preferred as the prover's costs are still orders of magnitude lower than with SNARKs [30], [38] constructed from other means (prior work [48] provides details of benefits of incremental proof systems).

1.2.1. Implementation and experimental evaluation.

We implement MicroNova as a modular library in about 11,000 lines of Rust. We also implement MicroNova's verifier in about 3,300 lines of Solidity. The prover is generic over a cycle of elliptic curves and a hash function that instantiates the random oracle for Fiat-Shamir transform [37]. For efficient on-chain verification, we employ the BN254/Grumpkin elliptic curve cycle (where BN254 is pairing-friendly and Grumpkin is not pairing-friendly), Poseidon (for in-circuit hash function), and Keccak (for on-chain hash function). The library accepts F described as a circuit with `bellpepper` [1].

We experimentally evaluate MicroNova on an Azure VM of size Standard F64s_v2 (64 vCPUs, 2.70 GHz Intel processor, and 128 GiB memory), and compare it with Nova [6], [52] on the same testbed. For MicroNova's on-chain verifier, we evaluate it with Foundry [4]. In our experiments, we vary the number of constraints N in F. We find the following.

IVC. The per-step prover cost in MicroNova's IVC scheme is about the same as the per-step prover cost in Nova—except when N is small. This is because MicroNova, which aims to keep the verifier circuit on non-pairing-friendly curve small, incurs a higher recursion overhead (≈74,000 constraints on BN254 and ≈1,300 on Grumpkin) when compared to Nova (≈10,000 constraints on BN254 and ≈10,000 on Grumpkin). However, this overhead is dwarfed when N exceeds the size of the recursive verifier circuit.

IVC proof compression and on-chain verification. MicroNova's compressed proofs are small, and quick to produce and verify. These costs are independent of the number of steps of IVC. In other words, the prover's and the verifier's costs can be amortized indefinitely with any number of steps prior to posting proofs on a blockchain.

- The prover's cost to compress an IVC proof scales close to linearly with N and is concretely small (tens of seconds). The prover's cost is dominated by a constant cost until $N \approx 2^{21}$: for on-chain verification, MicroNova, in addition to proving a folded F, proves a constant-sized R1CS (≈1.7M constraints) to prove the evaluation of a polynomial committed over the non-pairing-friendly curve (i.e., Grumpkin).
- The compressed proofs are small ≈ 11 KB until $N \approx 2^{21}$, and then has a logarithmic scaling with N (the proof length increases by 0.2 KB each time N doubles).
- The verifier takes ≈14 ms to verify a compressed proof.
- MicroNova's Solidity verifier takes ≈2.2M gas on Ethereum blockchain until $N \approx 2^{21}$, and then its gas cost has a

logarithmic scaling with N (the cost increases by 32K gas each time N doubles).

2. A technical overview of MicroNova

This section provides an overview of MicroNova, building on prior work [52], [56].[2] Section 4 provides a self-contained description of MicroNova.

(1) A new IVC scheme over half-pairing cycles. We begin with a simple modification to Nova: we instantiate Nova over a "half pairing" cycle of elliptic curves.[3] Specifically, instead of the Pasta curves, MicroNova uses the BN254/Grumpkin cycle (we enhanced Nova's implementation so that it is generic over the curve cycle and can use other "half pairing" curve cycles such as Pluto/Eris). One reason for this change is so that the verifier can make use of BN254 precompiles available in Ethereum. The other reason to use a paring-friendly curve is that we replace Pedersen's commitment scheme with the KZG polynomial commitment scheme [44]. In particular, Nova's IVC scheme only requires a succinct, additively homomorphic commitment scheme, so MicroNova uses KZG to commit to vectors defined over the scalar field of the pairing-friendly elliptic curve in the cycle, i.e., MicroNova treats entries in a vector of size n as coefficients of a univariate polynomial of degree n. On the non-pairing-friendly curve, MicroNova uses Pedersen commitments.

Since MicroNova treats the entries in a vector as coefficients of a univariate polynomial, it does not require any FFTs to change basis (e.g., from an evaluation from to coefficient form). In other words, the prover's commitment costs in MicroNova is same as that of Nova's—albeit on a different curve cycle. Because of this design, MicroNova exhibits another desirable feature of not requiring a new setup: MicroNova can use any existing trusted setup material that was created for other proof systems such as Plonk. Since KZG's trusted setup is updatable, one can start with existing material and update it to add additional security.

(i) Minimize circuit on non-pairing-friendly curve. A more substantial change is that we replace the folding strategy in Nova (which is formally described in [56]) with a strategy based on CycleFold [49]. This ensures that the size of the circuit defined on the secondary (i.e., non-pairing-friendly) curve goes down from 10,000 gates to about 1,300 gates. The verifier circuit on the primary (i.e., pairing-friendly) curve increases from 10,000 gates to about 37,000 gates. However, this trade-off is crucial for on-chain verification (which we unpack below). The higher overhead on the primary curve can be easily offset by using a step function F that is large enough to make the recursion overhead small.

(ii) Avoid circuit-friendly hashes in the on-chain verifier. Another substantial change is related to a hash check in the Nova verifier. In Nova, an IVC proof contains R1CS instances and their witnesses, and the verifier must ensure that the public IO of one of the instances contains a hash of the other instances. This hash check is also performed by the recursive verifier circuit, so the hash function is chosen to be circuit-friendly (e.g., Poseidon). In our context, the verifier runs on a blockchain, and on Ethereum, there are no precompiles for Poseidon hash function, so it must be implemented by using basic opcodes (which is expensive). Furthermore, Poseidon requires storing a large-sized parameters (which is also expensive). Replacing Poseidon with an on-chain-efficient hash function (e.g., Keccak) is not an option as this results in non-trivial recursion overheads. We introduce new techniques to get the best of both worlds: our mechanism allows the recursive verifier circuit to use a circuit-friendly hash function (e.g., Poseidon) while allowing the on-chain verifier to use an on-chain-efficient hash function (e.g., Keccak). In a nutshell, we leverage interaction (which is turned non-interactive with Fiat-Shamir transformation) where the verifier circuit "fingerprints" the message (it hashes with Poseidon) using a random challenge. The on-chain verifier then only has to derive a challenge by hashing the purported pre-image (e.g., with Keccak) and then reexecute the fingerprinting operation. Both are inexpensive for the on-chain verifier.

(2) A new proof compression layer. MicroNova follows the blueprint in Nova to compress IVC proofs: prove the knowledge of valid IVC proofs with Spartan [59]. We make substantial modifications to Spartan to enable efficient on-chain verification. In MicroNova, an IVC proof contains two committed relaxed R1CS instances, one on a pairing-friendly curve and another on the non-pairing-friendly curve. We devise specialized variants of Spartan for each of these.

For the instance on the pairing-friendly curve, we design MicroSpartan that only requires a *single* invocation of the sum-check protocol [54] and a single polynomial evaluation argument producing a proof of length $O(\log N)$ group elements. For the polynomial evaluation argument, building on a prior idea [21], we devise a reduction of knowledge [47] from a multilinear polynomial evaluation instance (where the multilinear polynomial is given in evaluation form) to a set of univariate polynomial evaluation instances, which we prove with a batched version of KZG [44]. We refer to this multilinear polynomial evaluation argument as HyperKZG.

(i) MicroSpartan: A SNARK with minimal proof sizes. Spartan [59] is a SNARK for R1CS. To prove the knowledge of a satisfying witness $\widetilde{w}$ to an R1CS instance with R1CS matrices (A, B, C), Spartan invokes the sum-check protocol multiple times in sequence. In particular, it invokes the sum-check protocol twice in sequence to reduce R1CS satisfiability to checking evaluations of multilinear polynomials $\widetilde{w}$, $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$ at a random point. To prove the evaluation of $\widetilde{w}$, one can invoke the evaluation argument of a polynomial commitment scheme. To prove evaluations of sparse multilinear polynomials encoding R1CS matrices, Spartan invokes Spark, a special-purpose SNARK that internally involves multiple sequential invocations of the sum-check protocol [60] and a polynomial evaluation argument. BabySpartan [61] improves

2. See [11], [16], [33] for resources on Nova and [65] for foundations.

3. A "half pairing" cycle is a two-cycle of elliptic curves where only one curve in the cycle supports efficient pairing operations.

upon Spartan in terms of commitment costs but retains Spartan's proof length and verification costs. A key takeaway is that the verifier has to evaluate multiple sum-check proofs and multiple polynomial evaluation arguments.

We redesign Spartan with a focus on minimizing verifier's costs—at the cost of the prover providing additional commitments. We refer to this variant as MicroSpartan. In particular, we provide a new reduction from circuit satisfiability to a zero-check and two lookup checks. MicroSpartan reduces all these checks simultaneously to a collection of multilinear polynomial evaluation instances, with a *single* invocation of the sum-check protocol. We introduce additional claims in this single sum-check protocol so that all committed multilinear polynomials are evaluated at the *same* random point. To prove these multilinear polynomial evaluation instances, the prover and the verifier fold them into a single instance with a simple random linear combination (without requiring any additional invocations of the sum-check protocol or other auxiliary protocols). To prove that single multilinear polynomial evaluation instance, MicroSpartan invokes HyperKZG. Since MicroSpartan is invoked only once after many steps of IVC, the additional commitment work by the prover is tolerable.

(ii) HyperKZG: Prove multilinear polynomial evaluation instances with a reduction to univariate KZG. MicroSpartan treats a vector w of size n containing the coefficients of a degree-n univariate polynomial as evaluations of a multilinear polynomial in $\log n$ variables over the Boolean hypercube. Accordingly, the univariate KZG commitment to w serves as a commitment to a multilinear polynomial $\widetilde{w}$, where $\widetilde{w}$ denotes the multilinear extension of the vector w. This matches exactly the requirement of MicroSpartan. Now, all that is necessary is a way to prove evaluations of multilinear polynomials committed in this manner.

Gemini [21, §2.4.2] describes a protocol to reduce a multilinear polynomial evaluation instance in *coefficient* form to a logarithmic number of univariate polynomial evaluations. However, in our context, the multilinear polynomials are in evaluation form rather than coefficient form. Directly applying Gemini's technique in our context requires polynomial interpolations during folding, which increases the prover's work. We make a small—but crucial—modification to Gemini's reduction protocol. This modification allows us to directly prove evaluations of multilinear polynomials in evaluation form, allowing us to use Spartan and its variant MicroSpartan to compress IVC proofs. We additionally specialize this protocol to use the univariate KZG scheme, allowing us to employ standard techniques to batch multiple univariate evaluations into one [14], [21], [36].

(iii) DelegatedSpartan: Proving the instance on the non-pairing-friendly curve. In MicroNova's IVC proof, for the committed relaxed R1CS instance defined over the non-pairing-friendly curve (e.g., Grumpkin), the commitment scheme used is Pedersen (one cannot use KZG on non-pairing-friendly curves). In this setting, the only option is to use an IPA-based evaluation argument [20], [23]. Unfortunately, the on-chain verifier will have to compute a multi-scalar multiplication (MSM) of size proportional to the number of constraints in the circuit defined over the secondary non-pairing-friendly curve. MicroNova's circuit on the secondary curve is minimal, but this still requires at least 1,000 group scalar multiplications from the on-chain verifier. There are no precompiles for operations over a curve such as Grumpkin, so this is infeasible (even with a precompile, the cost will be prohibitive). The on-chain verifier must also evaluate the multilinear extension (MLE) of R1CS matrices on its own or ask the prover to prove it via Spark. The latter increases the size of the MSM that the verifier must compute to be proportional to the number of non-zero entries in the R1CS matrices (in our setting this is about 5,600). It is infeasible to have the verifier perform these tasks.

To address these, we design a variant of Spartan, which we refer to as DelegatedSpartan. DelegatedSpartan runs Spartan's core protocol [59, §4.1], with two modifications: (1) the verifier efficiently evaluates the MLE of R1CS matrices on its own by leveraging the highly repetitive nature of the circuit; and (2) the verifier delegates the polynomial evaluation to the prover by using MicroSpartan.

- We observe that the circuit, which computes a group scalar multiplication on the non-pairing-friendly curve is highly structured: it consists of 128 iterations of the textbook double-and-add loop (using the incomplete affine addition law), followed by a complete addition to add the resulting point to the accumulator. We leverage this structure to make the verifier's work to evaluate the MLE of R1CS matrices inexpensive: the verifier performs finite field operations proportional to the number of non-zero entries resulting from a *single* iteration of the double and add operation and a *single* complete addition law. In a nutshell, the idea is to express the MLE of the entire matrix in terms of the MLE of repeated sub-matrix, so the verifier only has to evaluate the MLE of the repeated sub-matrix and then performs work logarithmic in the number of copies to compute the desired MLE of the entire matrix. This computation is inexpensive for the verifier—even when the verifier is implemented as a smart contract on Ethereum.
- We use MicroSpartan to prove the evaluation of a polynomial committed with Pedersen's commitment over the non-pairing-friendly curve. This is done by writing the polynomial evaluation algorithm as a circuit over the pairing-friendly curve in the cycle. To minimize the size of this circuit, MicroNova uses matrix commitments [68] to commit to the witness of the circuit on the second curve: view the witness as a matrix of size $n_1 \times n_2$ and commit to rows of the matrix with Pedersen's commitment scheme. During MicroNova's IVC, folding a matrix commitment requires n_1 group scalar multiplications instead of one in Pedersen's commitment, but this limits the size of MSM—to be proven via MicroSpartan—when compressing IVC proofs to be of size $O(n_1 + n_2)$ rather than $O(n_1 \cdot n_2)$. Concretely, we set $n_1 = 8, n_2 = 256$, which ensures that the polynomial evaluation circuit proven via MicroSpartan is only 1.7M constraints. With matrix

commitments, MicroNova's recursion overhead goes up from about 37,000 gates to about 74,000 gates.

3. Preliminaries

In this section, we recall reductions of knowledge. In Appendix A, we formally present multilinear polynomials and relevant properties, commitment schemes, arguments of knowledge, and IVC. This section borrows text from prior work [51].

3.1. Notation

We let λ to denote the security parameter. We let $\mathsf{negl}(\lambda)$ 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 let PPT denote probabilistic polynomial time and let EPT denote expected probabilistic polynomial time. We let $[n]$ denote the set $\{1, \ldots, n\}$. We let $\{u_i\}_{i\in[n]}$ denote the set $\{u_1, \ldots, u_n\}$.

We let $\mathbb{F}$ to denote a finite field (e.g., the prime field $\mathbb{F}_p$ for a large prime p) and let $\mathbb{F}^n$ denote vectors of length n over elements in $\mathbb{F}$. 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 degree bound. We denote vectors as $\boldsymbol{v} = (v_1, \ldots, v_n)$. Given a vector of polynomials $\boldsymbol{g}$, we let $\boldsymbol{g}(x) = (g_1(x), \ldots, g_n(x))$. We let $\mathsf{eq}(x, y) \in \mathbb{F}^1[X_1, \ldots, X_\ell, Y_1, \ldots, Y_\ell]$ denote the polynomial that outputs 1 if $x = y$ and 0 otherwise for $x, y \in \{0, 1\}^\ell$. For vector $v \in \mathbb{F}^n$ we let $\widetilde{v} \in \mathbb{F}^1[X_1, \ldots, X_{\log n}]$ denote the multilinear polynomial extension of v (i.e., $\widetilde{v}(i) = \sum_j \mathsf{eq}(i, j) \cdot v_j$).

3.2. Committed Relaxed R1CS

Definition 3.1 (Committed relaxed R1CS). Consider a finite field $\mathbb{F}$ and a commitment scheme Com over $\mathbb{F}$. Let the public parameters consist of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$, and commitment parameters pp_W and pp_E for vectors of size m and $m - \ell - 1$ respectively. The committed relaxed 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. A committed relaxed R1CS instance is a tuple $(\overline{E}, u, \overline{W}, \mathsf{x})$, where $\overline{E}$ and $\overline{W}$ are commitments, $u \in \mathbb{F}$, and $\mathsf{x} \in \mathbb{F}^\ell$ are public inputs and outputs. An instance $(\overline{E}, u, \overline{W}, \mathsf{x})$ is satisfied by a witness $(E, r_E, W, r_W) \in (\mathbb{F}^m, \mathbb{F}, \mathbb{F}^{m-\ell-1}, \mathbb{F})$ if $\overline{E} = \mathsf{Com}(\mathsf{pp}_E, E, r_E)$, $\overline{W} = \mathsf{Com}(\mathsf{pp}_W, W, r_W)$, and $(A \cdot Z) \circ (B \cdot Z) = u \cdot (C \cdot Z) + E$, where $Z = (W, \mathsf{x}, u)$.

3.3. Reductions of Knowledge

We now recall reductions of knowledge (RoK) [47], which are a generalization of arguments of knowledge, in which a verifier interactively *reduces* checking a prover's knowledge of a witness in a relation $\mathcal{R}_1$ to checking the prover's knowledge of a witness in another (simpler) relation $\mathcal{R}_2$. In particular, both parties take as input a claimed instance u_1 to be checked, and the prover additionally takes as input a corresponding witness w_1 such that $(u_1, w_1) \in \mathcal{R}_1$. After interaction, the prover and verifier together output a new statement u_2 to be checked in place of the original statement, and the prover additionally outputs a corresponding witness w_2 such that $(u_2, w_2) \in \mathcal{R}_2$. Appendix A.2 provides a formal definition.

Definition 3.2 (Folding scheme). A folding scheme for $\mathcal{R}_1^m$ and $\mathcal{R}_2^n$ is a RoK of type $\mathcal{R}_1^m \times \mathcal{R}_2^n \to \mathcal{R}_1$ where the encoder outputs its input structure. We call a reduction of type $\mathcal{R}^n \to \mathcal{R}$ simply as a folding scheme for $\mathcal{R}$.

3.4. The sum-check protocol

Recall that the standard sum-check relation checks that the sum of evaluations of an ℓ-variate polynomial Q (under a commitment) on the Boolean hypercube results in some value T. Formally, the sum-check relation is defined as follows.

Definition 3.3 (Unstructured sum-check relation). Let $(\mathsf{Gen}, \mathsf{Com})$ denote an additively homomorphic commitment scheme. Consider a size bound $\ell \in \mathbb{N}$. The unstructured sum-check relation USC over public parameter, instance, witness pairs is defined as follows.

$$\mathsf{USC} = \left\{ (\mathsf{pp}, (\overline{Q}, T), Q) \;\middle|\; \begin{array}{l} Q \in \mathbb{F}[X_1, \ldots, X_\ell], \\ \overline{Q} = \mathsf{Com}(\mathsf{pp}, Q), \\ T = \sum_{x\in\{0,1\}^\ell} Q(x) \end{array} \right\}$$

Central to our development is the sum-check protocol [54], which, when recast as a reduction of knowledge [22], reduces from the sum-check relation to the polynomial evaluation relation, which we define below.

Definition 3.4 (Polynomial evaluation relation). Let $(\mathsf{Gen}, \mathsf{Com})$ denote an additively homomorphic commitment scheme. Consider a size bound $\ell \in \mathbb{N}$ and let pp denote public parameters of the commitment scheme. We define the polynomial evaluation relation, PE, as follows.

$$\mathsf{PE} = \left\{ (\mathsf{pp}, (\overline{Q}, x, y), Q) \;\middle|\; \begin{array}{l} Q \in \mathbb{F}[X_1, \ldots, X_\ell], \\ \overline{Q} = \mathsf{Com}(\mathsf{pp}, Q), \\ y = Q(x) \end{array} \right\}.$$

Lemma 3.1 (The sum-check protocol)**.** *There exists a succinct, public-coin, tree-extractable reduction of knowledge (Lemma A.4) from* USC *to* PE *compatible with all encoder, generator, and commitment algorithms where the output commitment is the same as the input commitment. For polynomials in* $\mathbb{F}^d[X_1, \ldots, X_\ell]$ *the communication complexity is* $O(d \cdot \ell)$ *elements in* $\mathbb{F}$.

Definition 3.5 (Multilinear polynomial evaluation relation). Let $(\mathsf{Gen}, \mathsf{Com})$ denote an additively homomorphic commitment scheme. Consider a size bound $\ell \in \mathbb{N}$ and let

$\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, 2^\ell)$. We define the multilinear polynomial evaluation relation, MPE, as follows.

$$\mathsf{MPE} = \left\{ (\mathsf{pp}, (\overline{Q}, x, y), Q) \;\middle|\; \begin{array}{l} Q \in \mathbb{F}^1[X_1, \ldots, X_\ell], \\ \overline{Q} = \mathsf{Com}(\mathsf{pp}, Q), \\ y = Q(x) \end{array} \right\}.$$

Definition 3.6 (Univariate polynomial evaluation relation). Let $(\mathsf{Gen}, \mathsf{Com})$ denote an additively homomorphic commitment scheme. Consider a size bound $n \in \mathbb{N}$ and let $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, n)$. We define the univariate polynomial evaluation relation, UPE, as follows.

$$\mathsf{UPE} = \left\{ (\mathsf{pp}, (\overline{Q}, x, y), Q) \;\middle|\; \begin{array}{l} Q \in \mathbb{F}^n[X], \\ \overline{Q} = \mathsf{Com}(\mathsf{pp}, Q), \\ y = Q(x) \end{array} \right\}.$$

4. MicroNova's IVC with proof compression

This section describes MicroNova's IVC scheme. In a nutshell, MicroNova's IVC scheme starts with Nova's IVC scheme and makes crucial modifications to enable efficient verification of compressed IVC proofs. This section focuses on MicroNova's IVC scheme; we also describe how to compress MicroNova's IVC proofs assuming a black box SNARK. The next section describes a modified Spartan [59] that provides efficient on-chain verification. Our presentation here borrows from and extends the description in Nova [52].

4.1. MicroNova's folding scheme

MicroNova's folding scheme combines Nova's folding scheme [52] with CycleFold technique [49], to get a folding scheme over a 2-cycle of elliptic curves (E_1, E_2) such that the circuit defined over E_2 is constant-sized and small.

This allows us to use "half pairing" cycles (i.e., E_1 is pairing friendly but E_2 is not), which are significantly more efficient than cycles of pairing-friendly curves and as efficient as non-pairing-friendly cycle of curves (e.g., Pasta curves). The motivation to keep the instance on E_2 small is that when compressing IVC proofs (which we discuss in the next section), the prover has to prove the knowledge of a valid witness for the instance defined over E_2. Since E_2 is not pairing-friendly, one must use a depth-1 recursion using a SNARK defined over E_1 and keeping the instance on E_2 small significantly reduces the prover's work in this. The use of CycleFold [49] in MicroNova reduces the instance defined over E_2 by more than $10\times$ compared to Nova, but this is not sufficient. We discuss additional techniques in the next section to reduce the depth-1 recursion costs further.

Details. Recall that Nova [52] provides a folding scheme for committed relaxed R1CS, where the verifier performs two group scalar multiplications and two group additions. If the committed relaxed R1CS instances are defined over the scalar field of E_1, then these elliptic curve operations in the folding scheme's verifier represented as a circuit involve operations over the base field of E_1, so they cannot be efficiently represented in the scalar field of E_1.

Notation. Let $(\mathsf{u}_\perp, \mathsf{w}_\perp)$ be the trivially satisfying instance-witness pair in committed relaxed R1CS, where E, W, and x are appropriately-sized zero vectors, $r_E = 0$, $r_W = 0$, and $\overline{E}$ and $\overline{W}$ are commitments of E and W respectively.

Definition 4.1. Let EC denote a subset of committed relaxed R1CS (Definition 3.1) in which the constraints are defined over the scalar field of E_2 and the commitment scheme commits to vectors over the scalar field of E_2, and the structure is fixed to compute the following deterministic computation: $R \leftarrow P + r \cdot Q$, where P, Q, R are arbitrary (elliptic curve) group elements on E_1, and r is a scalar in the corresponding scalar field of E_1. For instances in EC, $\mathsf{x} = (r, P, Q, R)$, $u = 1$, and $\overline{E} = \mathsf{u}_\perp.\overline{E}$.

Since the elliptic curve points on E_1 are a pair of field elements in the base field of E_1 (which equals the scalar field of E_2), the elliptic curve group scalar multiplication and point addition can be represented "natively" on E_2 (i.e., without any wrong-field arithmetic or field emulation). Concretely, when the scalar is 128-bits, the structure in EC requires only ≈1,300 constraints in R1CS. We denote this committed relaxed R1CS structure as s_{EC}.

We interpret Nova's folding scheme as a RoK from two committed relaxed R1CS instance-witness pairs (defined over the scalar field of E_1) to a single committed relaxed R1CS instance-witness pair (defined over the scalar field of E_1) and two committed relaxed R1CS instance-witness pairs in EC (defined over the scalar field of E_2) i.e., Nova's folding scheme can be interpreted as an RoK of the form

$$\Pi_1 : \mathsf{CRR1CS} \times \mathsf{CRR1CS} \to \mathsf{CRR1CS} \times \mathsf{EC}^2.$$

In other words, the verifier in the Π_1 RoK obtains untrusted advice regarding the elliptic curve operations that it would have otherwise performed. This untrusted advice is provided in the form of a pair of committed relaxed R1CS instances that are satisfying if and only if the advice is correct.

We then invoke Nova's folding scheme twice to fold the advice instances into a running instance. This can be viewed as a RoK: $\Pi_2 : \mathsf{EC} \times \mathsf{EC}^2 \to \mathsf{EC}$. By sequentially composing the two RoKs, $\Pi_2 \circ (1_{\mathsf{EC}} \times \Pi_1)$, we get a multi-folding scheme, a generalization of folding schemes [50]:

$$(\mathsf{CRR1CS}, \mathsf{EC}) \times \mathsf{CRR1CS} \to (\mathsf{CRR1CS}, \mathsf{EC}).$$

Crucially, the verifier of this folding scheme is efficiently represented in E_1's scalar field. Also, the instance defined over E_2 is constant-sized (e.g., $\approx 1,300$ R1CS constraints).

Non-interactivity. To design MicroNova's IVC scheme, we require our folding scheme to be non-interactive in the standard model. As in prior work [50], [52], we apply the Fiat-Shamir transformation [37] to achieve non-interactivity. Next, we heuristically instantiate the random oracle using a cryptographic hash function. Naturally, we can only heuristically argue the security of the resulting non-interactive folding scheme. Note that most recursive arguments require such an assumption [24]–[26], [31], [48], [50], [52].

4.2. MicroNova's IVC scheme

We construct an IVC scheme from MicroNova's folding scheme using an approach similar to Nova's. We make a crucial modification to enable efficient verification of compressed IVC proofs.

In Nova, the verifier has to perform a hash check to ensure that a value of public input in one of the R1CS instances in the proof is a hash of another instance and other material (a verification key, initial input and final output of IVC, etc.). For efficiency, MicroNova and its base Nova instantiate this hash function with Poseidon [40]. However, Poseidon is inefficient to verify on Ethereum and requires a large set of parameters to be stored on-chain. A naive solution is to use a general proof system: the prover proves that the required hash check holds by producing a SNARK. Unfortunately, this requires producing and verifying a SNARK. To address this, we design a light-weight RoK and we embed certain checks inside the recursive verifier circuit. This RoK enables the verifier circuit to use a circuit-friendly hash function (e.g., Poseidon) while allowing the on-chain verifier to use an on-chain-efficient hash function (e.g., Keccak). Furthermore, our approach does not require producing any additional SNARK, improving efficiency.

4.2.1. An auxiliary RoK.

Definition 4.2 (Hash relation). Let $n \in \mathbb{N}$ denote a size bound. Let h denote a hash function that can hash a vector of n elements over a finite field $\mathbb{F}$. We define the hash relation over public parameter, instance, witness tuples as follows

$$\mathsf{HASH} = \left\{ (\mathsf{pp}, (d, v), \bot) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda), \\ v \in \mathbb{F}^n, \\ h(\mathsf{pp}, v) = d, \end{array} \right\}.$$

Definition 4.3 (Hash circuit relation). Consider a subset of committed relaxed R1CS, where the public IO contains three values $(c, d, f) \in \mathbb{F}^3$, a part of the non-deterministic witness is a vector $v \in \mathbb{F}^n$, the structure, which we denote with $\mathsf{s}_{\mathsf{HAC}}$, computes $d \leftarrow h(\mathsf{pp}, v)$ and $f \leftarrow \sum_{i=0}^{n-1} v_i \cdot c^i$, $u = 1$, and $\overline{E} = \mathsf{u}_\bot.\overline{E}$. Denote this subset with HAC.

Construction 1 (RoK from HASH to HAC). We construct a RoK as follows.

- $\mathsf{Gen}(1^\lambda) \to \mathsf{pp}$: produce parameters for h.
- $\mathcal{K}(\mathsf{pp}, \mathsf{s} = \bot) \to (\mathsf{pk}, \mathsf{vk})$: output $(\mathsf{pp}, \mathsf{pp})$.

$\mathcal{P}$ and $\mathcal{V}$ are given as HASH instance: (d, v).

1) $\mathcal{V} \to \mathcal{P}$: $c \in_R \mathbb{F}$.
2) $\mathcal{P} \to \mathcal{V}$: The prover computes an instance-witness pair $(\mathsf{u}, \mathsf{w}) \in \mathsf{HAC}$ and sends $\overline{W} \leftarrow \mathsf{u}.\overline{W}$.
3) $\mathcal{V}$: Compute $f = \sum_{i=0}^{n-1} v_i \cdot c^i$.
4) $\mathcal{P}, \mathcal{V}$: Produce the following instance-witness pair in HAC (the verifier only outputs the instance):

$$((\mathsf{u}_\bot.\overline{E}, 1, \overline{W}, (c, d, f)), \mathsf{w}) \in \mathsf{HAC}$$

.

Lemma 4.1. *Construction 1 is an RoK of type* $\mathsf{HASH} \to \mathsf{HAC}$ *with a single round, a constant communication complexity, and the prover's work is linear in* n.

Proof. The claimed efficiency is easy to check. Perfect completeness is easy to check. Given a valid instance (d, v) in HASH, $d = h(v)$. For any $c \in \mathbb{F}$ sent by the verifier, the prover can compute a satisfying instance-witness pair in HAC. Specifically, the prover computes a committed relaxed R1CS instance-witness pair (u, w) that computes $d \leftarrow h(v)$ and $f \leftarrow \sum_{i=0}^{n-1} v_i \cdot c^i$ and places (c, d, f) in the public IO. In the honest prover case, the relaxed instance is "strict" i.e., $\mathsf{u}.u = 1$ and $\mathsf{u}.\overline{E} = \mathsf{u}_\bot.\overline{E}$. This instance passes the verifier's check in step 3. Furthermore, the instance output by the verifier matches that of the prover. So, the output instance-witness pair is in HAC and is satisfying.

We prove knowledge soundness as follows. Consider an adversary $\mathcal{A}$ that adaptively picks the input instance and a malicious prover $\mathcal{P}^\star$ that succeds with probability ϵ. Let $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$. Suppose on input a random tape r, the adversary $\mathcal{A}$ picks an HASH instance (d, v) and some auxiliary state st. We construct an extractor $\mathcal{E}$ that succeds with probability $\epsilon - \mathsf{negl}(\lambda)$ in obtaining satisfying witness for the original instance.

On input r, $\mathcal{E}$ first obtains the following tuple from the adversary:

$$(\mathsf{s} = \bot, (d, v), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{r})$$

The extractor $\mathcal{E}$ then computes $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s} = \bot)$. Next, $\mathcal{E}$ runs

$$(\mathsf{u} = (\mathsf{u}_\bot.\overline{E}, 1, \overline{W}, (c, d, f)), \mathsf{w}) \leftarrow \langle \mathcal{P}^\star, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (d, v), \mathsf{st}).$$

The extractor outputs $\bot$ as the witness for the input instance. Since the extractor runs $\mathcal{P}^\star$ only once, it runs in expected polynomial time.

We must now argue that $\bot$ is a satisfying witness for the HASH instance (d, v) with probability $\epsilon - \mathsf{negl}(\lambda)$. We are given that w is a satisfying witness for the HAC instance $\mathsf{u} = (\mathsf{u}_\bot.\overline{E}, 1, \overline{W}, (c, d, f))$ with structure $\mathsf{s}_{\mathsf{HAC}}$. From this satisfying witness, parse out a value $v' \in \mathbb{F}^n$ such that $d = h(\mathsf{pp}, v')$ and $f = \sum_{i=0}^{n-1} v'_i \cdot c^i$. From the verifier's checks, we have that $f = \sum_{i=0}^{n-1} v_i \cdot c^i$. Since c is a random challenge and because h is a collision-resistant hash function, by the Schwartz-Zippel lemma over c, we have that $v = v'$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Since $h(\mathsf{pp}, v') = d$ and $v = v'$, we have that $d = h(\mathsf{pp}, v)$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Therefore, $(\mathsf{pp}, (d, v), \bot) \in \mathsf{HASH}$ with probability $\epsilon - \mathsf{negl}(\lambda)$. □

Non-interactivity. We make the above RoK non-interactive using Fiat-Shamir transform and instantiate the random oracle in the plain model using a concrete hash function. As in prior work [47], we assume that the resulting non-interactive RoK is knowledge sound. Let $\mathcal{H}$ denote this hash function.

In our concrete instantiation, we use an on-chain efficient hash function for $\mathcal{H}$, specifically Keccak.

4.2.2. Constructing IVC from MicroNova's folding scheme. Recall that an IVC scheme allows a prover to show that $z_n = F^{(n)}(z_0)$ for some count n, initial input z_0, and output z_n. We now show how to construct an IVC scheme for a non-deterministic, polynomial-time computable function F using our non-interactive folding scheme for committed relaxed R1CS (§4.1). In MicroNova's folding scheme, the "running" instance is a pair of committed relaxed R1CS instances, one encoding the correct execution of F' and another encoding the correct execution of a group scalar multiplication operation followed by a group point addition. So, in the construction below, for a trivially satisfying running instance, we use $(\mathsf{u}_\perp, \mathsf{u}_\perp)$.

Construction 2 (IVC). Let $\mathsf{NIFS} = (\mathsf{G}, \mathsf{K}, \mathsf{P}, \mathsf{V})$ be the non-interactive folding scheme for committed relaxed R1CS (§4.1). Consider a polynomial-time function F that takes non-deterministic input, and a cryptographic hash function hash. We define our augmented function F' as follows (all arguments to F' are taken as non-deterministic advice):

$F'(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi, r_i, r_{i+1}, c) \to \mathsf{x}$:
If $i = 0$,
(1) $m \leftarrow (\mathsf{vk}, i+1, z_0, F(z_0, \omega_0), (\mathsf{u}_\perp, \mathsf{u}_\perp), r_{i+1})$,
(2) compute $d \leftarrow \mathsf{hash}(m)$, and $f \leftarrow \sum_{i=0}^{n-1} m_i \cdot c^i$,
(3) output (c, d, f).
Otherwise,
(1) parse $\mathsf{u}_i.\mathsf{x}$ as (c, d, f), where $\mathsf{u}_i.\mathsf{x}$ is the public IO of u_i,
(2) check that $d = \mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i, r_i)$
(3) check that $(\mathsf{u}_i.\overline{E}, \mathsf{u}_i.u) = (\mathsf{u}_\perp.\overline{E}, 1)$,
(4) compute $\mathsf{U}_{i+1} \leftarrow \mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, \pi)$, and
(5) $m \leftarrow (\mathsf{vk}, i+1, z_0, F(z_i, \omega_i), \mathsf{U}_{i+1}, r_{i+1})$,
(6) compute $d \leftarrow \mathsf{hash}(m)$, and $f \leftarrow \sum_{i=0}^{n-1} m_i \cdot c^i$,
(7) output (c, d, f).

Because F' can be computed in polynomial time, it can be represented as a committed relaxed R1CS structure. We assume that there is a deterministic procedure, which we denote with $\mathsf{AUGMENT}$, that takes as input a function F and public parameters pp sampled by $\mathcal{G}$ of the IVC scheme, and outputs the committed relaxed R1CS structure corresponding to F', which we denote with $\mathsf{s}_{F'}$. Note that this assumes a canonical representation of the message m as a vector of n field elements (n is a constant). Let $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1}) \leftarrow \mathsf{trace}(F', (\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi, r_i, r_{i+1}, c))$ denote the satisfying committed relaxed R1CS instance-witness pair $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$ for the execution of F', as a committed Relaxed R1CS with structure $\mathsf{s}_{F'}$, on non-deterministic advice $(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi, r_i, r_{i+1}, c)$. Note that trace is a randomized algorithm that internally samples randomness to create hiding commitments inside u_{i+1}. Additionally, note that trace sets $\mathsf{u}_{i+1}.\overline{E} = \mathsf{u}_\perp.\overline{E}$ and that $\mathsf{u}_{i+1}.u = 1$.

We define the IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ as follows.

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

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

(1) compute $\mathsf{s}_{F'} \leftarrow \mathsf{AUGMENT}(\mathsf{pp}, F)$;
(2) compute $(\mathsf{pk}_{\mathsf{fs}}, \mathsf{vk}_{\mathsf{fs}}) \leftarrow \mathsf{NIFS.K}(\mathsf{pp}, \mathsf{s}_{F'})$;
(3) output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((\mathsf{pp}, F, \mathsf{pk}_{\mathsf{fs}}), (\mathsf{pp}, \mathsf{s}_{F'}, \mathsf{vk}_{\mathsf{fs}}))$.

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

(1) if $i = 0$, $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi) \leftarrow ((\mathsf{u}_\perp, \mathsf{u}_\perp), (\mathsf{w}_\perp, \mathsf{w}_\perp), \perp)$;
(2) otherwise, parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), r_i)$ and compute $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}, (\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$;
(3) sample $r_{i+1} \in \mathbb{F}$ randomly;
(4) compute $c \leftarrow \mathcal{H}(\mathsf{vk}, i+1, z_0, F(z_i, \omega_i), \mathsf{U}_{i+1}, r_{i+1})$;
(5) compute $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1}) \leftarrow \mathsf{trace}(F', (\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi, r_i, r_{i+1}, c))$, and
(6) output $\Pi_{i+1} \leftarrow ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}), r_{i+1})$.

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

If $i = 0$, check that $z_i = z_0$;
otherwise,
(1) parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), r_i)$,
(2) parse $\mathsf{u}_i.\mathsf{x}$ as (c, d, f), where $\mathsf{u}_i.\mathsf{x}$ is the public IO of u_i,
(3) check that $d = \mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i, r_i)$,
(4) check that $(\mathsf{u}_i.\overline{E}, \mathsf{u}_i.u) = (\mathsf{u}_\perp.\overline{E}, 1)$, and
(5) check that W_i and w_i are satisfying witnesses to U_i and u_i respectively using $\mathsf{vk.pp}$ and $\mathsf{vk.s}_{F'}$.

Lemma 4.2. *Construction 2 is an IVC scheme that satisfies completeness and knowledge soundness.*

Proof Intuition (Completeness). Given a satisfying IVC proof $\Pi_i = ((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), r_i)$ suppose that $\mathcal{P}$ outputs $\Pi_{i+1} = ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}), r_{i+1})$. Because Π_i is a valid IVC proof, $(\mathsf{u}_i, \mathsf{w}_i)$ and $(\mathsf{U}_i, \mathsf{W}_i)$ are satisfying instance-witness pairs. Because $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1})$ is obtained by folding $(\mathsf{u}_i, \mathsf{w}_i)$ and $(\mathsf{U}_i, \mathsf{W}_i)$, it must be satisfying by the folding scheme's completeness. By construction, $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$ is satisfying instance-witness pair that satisfies the IVC verifier's auxiliary checks including the ones that involve r_{i+1}. Thus, Π_{i+1} is satisfying. □

Proof Intuition (knowledge soundness). For function F, constant n, $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F)$, consider an adversary $\mathcal{P}^*$ that outputs (z_0, z, Π) such that $\mathcal{V}(\mathsf{vk}, (n, z_0, z), \Pi) = 1$ with probability ϵ. We construct an extractor $\mathcal{E}$ that with input (pp, z_0, z), outputs $(\omega_0, \ldots, \omega_{n-1})$ such that by computing $z_i \leftarrow F(z_{i-1}, \omega_{i-1})$ for all $i \in \{1, \ldots, n\}$ we have that $z_n = z$ with probability $\epsilon - \mathsf{negl}(\lambda)$. We show inductively that $\mathcal{E}$ can construct an extractor $\mathcal{E}_i$ that outputs $(z_i, \ldots, z_{n-1})$, $(\omega_i, \ldots, \omega_{n-1})$, and Π_i such that for all $j \in \{i+1, \ldots, n\}$, $z_j = F(z_{j-1}, \omega_{j-1})$, $\mathcal{V}(\mathsf{vk}, i, z_0, z_i, \Pi_i) = 1$, and $z_n = z$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Then, because in the base case when $i = 0$, $\mathcal{V}$ checks that $z_0 = z_i$, it is sufficient for $\mathcal{E}$ to run $\mathcal{E}_0$ to retrieve values $(\omega_0, \ldots, \omega_{n-1})$. Initially, $\mathcal{E}_n$ simply runs the assumed $\mathcal{P}^*$ to get a satisfying Π_n. Given extractor $\mathcal{E}_i$ that

satisfies the inductive hypothesis, we can construct extractor $\mathcal{E}_{i-1}$. Note that this proof is identical to the knowledge soundness proof of Nova's IVC scheme. □

4.3. Compressing MicroNova's IVC Proofs

MicroNova's IVC proofs are linear in the size of F, so they are not efficient to verify on a blockchain. We now discuss how to compress MicroNova's IVC proofs so they are exponentially smaller and faster to verify. In theory, one can address this problem with any SNARK for NP. Specifically, the prover can produce a SNARK proving that it knows Π_i such that IVC verifier $\mathcal{V}$ accepts for statement (i, z_0, z_i). Unfortunately, employing an off-the-shelf SNARK makes the overall solution impractical as the SNARK prover must prove, among other things, the knowledge of vectors whose commitments equal a particular value; this requires encoding a linear number of group scalar multiplications in R1CS. To address this, we design SNARKs tailored for our purpose and we describe it in Section 5. Below, we describe how to use a SNARK to prove the knowledge of a valid IVC proof. Formally, we design a SNARK for the following relation.

Definition 4.4 (IVC proof validity relation). Let $\mathsf{IVC} = (\mathsf{G}, \mathsf{K}, \mathsf{P}, \mathsf{V})$ denote the IVC scheme described in Construction 2. We define the relation $\mathcal{R}_{\mathsf{VIVC}}$ over public parameter, structure, instance, and witness tuples as follows.

$$\mathcal{R}_{\mathsf{VIVC}} = \left\{ (\mathsf{pp}, F, (n, z_0, z_n), \Pi_n) \;\middle|\; \begin{array}{l} \mathsf{vk} \leftarrow \mathsf{IVC.K}(\mathsf{pp}, F), \\ \mathsf{IVC.V}(\mathsf{vk}, (n, z_0, z_n), \Pi) = 1 \end{array} \right\}$$

In a nutshell, we leverage the fact that Π contains three committed relaxed R1CS instance-witness pairs. So, $\mathcal{P}$ first folds the instance-witness pairs (u, w) and (U, W) in Π to produce a folded instance-witness pair $(\mathsf{U}', \mathsf{W}')$, using $\mathsf{NIFS.P}$. Next, $\mathcal{P}$ runs $\mathsf{SNARK.P}$ to produce a proof π that it knows a valid witness for U'. The compressed proof consists of $(\mathsf{U}, \mathsf{u}, \pi)$. When $\mathcal{V}$ verifies $\mathcal{P}$'s proof, it will have to perform a hash check correspondance between u and U, where the hash function used is the same as the one used in F' (i.e., Poseidon in our case). To avoid this, we employ an auxiliary RoK (Construction 1) allowing the verifier to instead use an on-chain-efficient hash function (e.g., Keccak).

Construction 3 (A SNARK of a Valid IVC Proof). Let $\mathsf{IVC} = (\mathsf{G}, \mathsf{K}, \mathsf{P}, \mathsf{V})$ denote the IVC scheme in Construction 2, let NIFS denote the non-interactive folding scheme (§4.1), and let hash and $\mathcal{H}$ denote two cryptographic hash functions, hash is circuit-friendly (e.g., Poseidon) and $\mathcal{H}$ is on-chain-friendly (e.g., Keccak). Let SNARK denote a SNARK for committed relaxed R1CS that has the same public parameter generator algorithm as the IVC scheme. We construct a SNARK $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for the relation $\mathcal{R}_{\mathsf{VIVC}}$ (Definition 4.4) as follows.

$\underline{\mathcal{G}(1^\lambda) \to \mathsf{pp}}$: Output $\mathsf{pp} \leftarrow \mathsf{SNARK.G}(1^\lambda)$

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

(1) Compute $(\mathsf{pk}_{\mathsf{IVC}}, \mathsf{vk}_{\mathsf{IVC}}) \leftarrow \mathsf{IVC.K}(\mathsf{pp}, F)$.
(2) Compute $\mathsf{s}_{F'} \leftarrow \mathsf{AUGMENT}(\mathsf{pp}, F)$.
(3) Compute $(\mathsf{pk}_{\mathsf{F}'}, \mathsf{vk}_{\mathsf{F}'}) \leftarrow \mathsf{SNARK.K}(\mathsf{pp}, \mathsf{s}_{F'})$.
(4) Compute $(\mathsf{pk}_{\mathsf{EC}}, \mathsf{vk}_{\mathsf{EC}}) \leftarrow \mathsf{SNARK.K}(\mathsf{pp}, \mathsf{s}_{\mathsf{EC}})$.
(5) Output $((\mathsf{pk}_{\mathsf{IVC}}, \mathsf{pk}_{\mathsf{F}'}, \mathsf{pk}_{\mathsf{EC}}), (\mathsf{vk}_{\mathsf{IVC}}, \mathsf{vk}_{\mathsf{F}'}, \mathsf{vk}_{\mathsf{EC}}))$.

$\underline{\mathcal{P}(\mathsf{pk}, (n, z_0, z_n), \Pi_n)) \to \pi}$:

If $n = 0$, output $\perp$; otherwise,
(1) parse Π_n as $((\mathsf{U}_n, \mathsf{W}_n), (\mathsf{u}_n, \mathsf{w}_n), r_n)$
(2) $(\mathsf{U}', \mathsf{W}', \pi_n) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}_{\mathsf{IVC}}, ((\mathsf{U}_n, \mathsf{W}_n), (\mathsf{u}_n, \mathsf{w}_n)))$
(3) parse $(\mathsf{U}', \mathsf{W}')$ as $((\mathsf{U}_{\mathsf{F}'}, \mathsf{U}_{\mathsf{EC}}), (\mathsf{W}_{\mathsf{F}'}, \mathsf{W}_{\mathsf{EC}}))$
(4) compute $\pi_{\mathsf{F}'} \leftarrow \mathsf{SNARK.P}(\mathsf{pk}_{\mathsf{F}'}, \mathsf{U}_{\mathsf{F}'}, \mathsf{W}_{\mathsf{F}'})$
(5) compute $\pi_{\mathsf{EC}} \leftarrow \mathsf{SNARK.P}(\mathsf{pk}_{\mathsf{EC}}, \mathsf{U}_{\mathsf{EC}}, \mathsf{W}_{\mathsf{EC}})$
(6) output $(\mathsf{U}_n, \mathsf{u}_n, r_n, \pi_n, \pi_{\mathsf{F}'}, \pi_{\mathsf{EC}})$.

$\underline{\mathcal{V}(\mathsf{vk}, (n, z_0, z_n), \pi) \to \{0, 1\}}$:

If $n = 0$, check that $z_0 = z_i$; otherwise,
(1) parse π as $(\mathsf{U}_n, \mathsf{u}_n, r_n, \pi_n, \pi_{\mathsf{F}'}, \pi_{\mathsf{EC}})$,
(2) parse public IO $\mathsf{u}_n.\mathsf{x}$ as (c, d, f),
(3) $m \leftarrow (\mathsf{vk}_{\mathsf{IVC}}, i, z_0, z_n, \mathsf{U}_n, r_n)$,
(4) check that $c = \mathcal{H}(m)$ and $f = \sum_{i=0}^{n-1} m_i \cdot c^i$,
(5) check that $(\mathsf{u}.\overline{E}, \mathsf{u}.u) = (\mathsf{u}_\perp.\overline{E}, 1)$,
(6) compute $\mathsf{U}' \leftarrow \mathsf{NIFS.V}(\mathsf{vk}_{\mathsf{IVC}}, \mathsf{U}_n, \mathsf{u}_n, \pi_n)$,
(7) parse U' as $(\mathsf{U}_{\mathsf{F}'}, \mathsf{U}_{\mathsf{EC}})$
(7) check that $\mathsf{SNARK.V}(\mathsf{vk}_{\mathsf{F}'}, \mathsf{U}_{\mathsf{F}'}, \pi_{\mathsf{F}'}) = 1$, and
(8) check that $\mathsf{SNARK.V}(\mathsf{vk}_{\mathsf{EC}}, \mathsf{U}_{\mathsf{EC}}, \pi_{\mathsf{EC}}) = 1$.

Theorem 4.1 (A SNARK of a Valid IVC Proof)**.** *Construction 3 is a SNARK of a valid IVC proof produced by Construction 2.*

Proof Intuition. Completeness and knowledge soundness hold due to the completeness and knowledge soundness of the underlying SNARK, the auxiliary RoK (Construction 1), and the non-interactive folding scheme. In more detail, by the knowledge soundness of the non-interactive auxiliary RoK (§4.2.1), $\mathcal{V}$'s check in step (3) and (4) is equivalent to checking that $d = \mathsf{hash}(\mathsf{vk}_{\mathsf{IVC}}, i, z_0, z_n, \mathsf{U}_n, r_n)$. Assuming that the non-interactive folding scheme satisfies succinctness (e.g., with Pedersen commitments), succinctness holds due to the fact that u, U, and π are succinct, and due to the succinctness of the underling SNARK. □

5. MicroNova's SNARKs for R1CS

MicroNova's IVC proof compression layer (§4.3) requires two SNARKs: (1) a SNARK to prove a committed relaxed R1CS instance encoding the correct execution of F', and (2) a SNARK to prove an instance in EC, which itself a subset of committed relaxed R1CS. In MicroNova, the first instance is defined over a pairing-friendly elliptic curve E_1 (e.g., BN254), and the second instance is defined over a non-pairing-friendly elliptic curve E_2 (e.g., Grumpkin), and E_1/E_2 is a 2-cycle of elliptic curves (i.e., the base field of E_1 equals the scalar field of E_2 and vice versa). We now

describe two different adaptations of Spartan [59]. We refer to these as MicroSpartan and DelegatedSpartan respectively.

5.1. MicroSpartan (for a pairing-friendly curve)

This section describes MicroSpartan, a SNARK that builds on Spartan [59] but minimizes its verifier time and proof sizes. Recall that Spartan runs several sequential invocations of the sum-check protocol and reduces the satisfiability of committed relaxed R1CS to proving evaluations of two multilinear polynomials encoding the witness W and error terms E and of three sparse multilinear polynomials encoding the R1CS structure (A, B, C).

In contrast, MicroSpartan reduces the satisfiability of committed relaxed R1CS to a collection of sum-check instances and two lookup checks. The lookup instances can be proven with a straightforward use of lookup arguments. A state-of-the-art sum-check-based option is Lasso [63], but it uses multiple invocations of the sum-check protocol. Instead, we employ a bit more expensive logUp protocol [24], [42], which requires the prover to send additional polynomial commitments. With those additional commitments, the lookup check reduces to a handful of sum-check instances. The prover then invokes the sum-check protocol [54] to prove all sum-check instances including the ones arising from lookup checks. At the end of the sum-check protocol, all committed polynomials are queried at the same point r, where r is chosen over the course of the sum-check protocol. Since all polynomials are evaluated at the same random point, by leveraging homomorphic properties of the polynomial commitments, the verifier requests an evaluation of a single polynomial at a single point. In other words, the prover and the verifier fold evaluation claims about all polynomial commitments into a single evaluation claim, so the prover has to produce a single polynomial evaluation argument about the folded polynomial. Overall, MicroNova makes a single invocation of the sum-check protocol followed by a single invocation of the polynomial evaluation argument. For the polynomial evaluation argument, MicroNova uses HyperKZG (§6), which reduces the single multilinear evaluation into a collection of univariate polynomial evaluation instances, which are proven with a batched version of KZG.

We recall the sum-check relation SC [51, Definition 6], and introduce additional relations. For $N \in \mathbb{N}$, let eq_τ, where $\tau \in \mathbb{F}^{\log N}$, denote the multilinear polynomial g such that for all $i \in \{0,1\}^{\log N}$, $g(i) = \mathsf{eq}(\tau, i)$.

Definition 5.1 (Structured lookup relation). Let $(\mathsf{Gen}, \mathsf{Com})$ denote an additively-homomorphic commitment scheme for vectors over finite field $\mathbb{F}$. Consider size parameters $n, m \in \mathbb{N}$. We define the lookup relation SLKP over public parameter, instance, witness tuples as follows

$$\mathsf{SLKP} = \left\{ (\mathsf{pp}, (\overline{a}, \overline{v}), (\tau, a, v)) \;\middle|\; \begin{array}{l} v, a \in \mathbb{F}^m, \mathsf{eq}_\tau \in \mathbb{F}^n, \\ \forall i \in [m].\ v_i = \tau_{a_i}, \\ \mathsf{Com}(\mathsf{pp}, a) = \overline{a}, \\ \mathsf{Com}(\mathsf{pp}, v) = \overline{v}, \end{array} \right\}.$$

Definition 5.2 (Piece-wise lookup relation). Let $(\mathsf{Gen}, \mathsf{Com})$ denote an additively-homomorphic commitment scheme for vectors over finite field $\mathbb{F}$. Consider size parameters $n, m, \ell \in \mathbb{N}$. We define the lookup relation PLKP over public parameter, instance, witness tuples as follows. $\mathsf{PLKP} =$

$$\left\{ (\mathsf{pp}, (\overline{w}, \overline{a}, \overline{v}), (\mathsf{x}, u, a, v)) \;\middle|\; \begin{array}{l} z = (w, u, \mathsf{x}), \\ v, a \in \mathbb{F}^m, \mathsf{x} \in \mathbb{F}^\ell, \\ u \in \mathbb{F}, z \in \mathbb{F}^n, \\ \forall i \in [m].\ v_i = z_{a_i}, \\ \mathsf{Com}(\mathsf{pp}, w) = \overline{w}, \\ \mathsf{Com}(\mathsf{pp}, a) = \overline{a}, \\ \mathsf{Com}(\mathsf{pp}, v) = \overline{v}, \end{array} \right\}.$$

Construction 4 (MicroSpartan: A RoK $\mathsf{CRR1CS} \to \mathsf{LKP}$). Let $\Pi = (\mathsf{Gen}, \mathsf{Com})$ denote an additively-homomorphic commitment scheme for vectors over finite field $\mathbb{F}$. Let m denote the number of constraints and n denote the number of witness variables for members of $\mathsf{CRR1CS}$. Let N_A, N_B, N_C denote the number of non-zero entries in R1CS matrices. Let $N = N_A + N_B + N_C$. WLOG, N is a power of 2. Suppose that matrices are represented in the COO format as a vector of tuples. Let M denote the concatenation of sparse representations of A, B, C. We construct a RoK as follows.

- $\mathsf{Gen}(1^\lambda) \to \mathsf{pp}$: output $\Pi.\mathsf{Gen}(1^\lambda, N)$
- $\mathcal{K}(\mathsf{pp}, \mathsf{s}) \to (\mathsf{pk}, \mathsf{vk})$: compute $\mathsf{row}, \mathsf{col}, \mathsf{val}_A, \mathsf{val}_B, \mathsf{val}_C \in \mathbb{F}^N$ as follows.
  - for $(i, (r, c, v)) \in M$, $\mathsf{row}[i] = r$, $\mathsf{col}[i] = c$.
  - for $(i, (r, c, v)) \in A$, $\mathsf{val}_A[i] = v$.
  - for $(i, (r, c, v)) \in B$, $\mathsf{val}_B[i + n_A] = v$.
  - for $(i, (r, c, v)) \in C$, $\mathsf{val}_C[i + n_A + n_B] = v$.

  $\mathsf{pk} \leftarrow (\mathsf{row}, \mathsf{col}, \mathsf{val}_A, \mathsf{val}_B, \mathsf{val}_C)$

  $\mathsf{vk} \leftarrow (\mathsf{Com}(\mathsf{pp}, \mathsf{row}), \mathsf{Com}(\mathsf{pp}, \mathsf{col}), \mathsf{Com}(\mathsf{pp}, \mathsf{val}_A),$

  $\mathsf{Com}(\mathsf{pp}, \mathsf{val}_B), \mathsf{Com}(\mathsf{pp}, \mathsf{val}_C))$

  Output $(\mathsf{pk}, \mathsf{vk})$

The prover and the verifier are given an instance in $\mathsf{CRR1CS}$: $(\overline{E}, \overline{W}, u, \mathsf{x})$. The prover is additionally provided with a witness: (E, W). Let $z = (w, u, \mathsf{x})$. WLOG, we assume that all vectors are padded with zeros to be of size N.

- $\mathcal{P} \to \mathcal{V}$: The prover sends commitments $(\overline{a}, \overline{b}, \overline{c})$, where $a = Az, b = Bz, c = Cz$,$\overline{a} = \mathsf{Com}(\mathsf{pp}, a), \overline{b} = \mathsf{Com}(\mathsf{pp}, b), \overline{c} = \mathsf{Com}(\mathsf{pp}, c)$.
- $\mathcal{V} \to \mathcal{P}$: Sample and send $\tau \in_R \mathbb{F}^{\log N}$.
- $\mathcal{P} \to \mathcal{V}$: The prover sends $(v_a, v_b, v_c, \overline{L_\mathsf{r}}, \overline{L_\mathsf{c}})$, where $v_a = \widetilde{a}(\tau), v_b = \widetilde{b}(\tau), v_c = \widetilde{c}(\tau)$, and for all $i \in [N]$, $L_\mathsf{r}[i] = \mathsf{eq}_\tau[\mathsf{row}[i]]$ and $L_\mathsf{c}[i] = z[\mathsf{col}[i]]$.
- $\mathcal{V} \to \mathcal{P}$: Sample and send a random value $c \in \mathbb{F}$
- The prover and the verifier output the following instance-witness pairs (the verifier only outputs instances):

– $(\mathsf{pp}, (F_1, G_1), (\overline{a}, \overline{b}, \overline{c}, \overline{E}, u, \tau), (a, b, c, E)) \in \mathsf{SC}$,

$$G_1((a, b, c, E), (u, \tau)) = (a, b, c, E, \mathsf{eq}_\tau)$$
$$F_1(a(x), b(x), c(x), E(x), \mathsf{eq}_\tau) = \mathsf{eq}_\tau(x) \cdot (a(x) \cdot b(x) - u \cdot c(x) - E(x))$$

– $(\mathsf{pp}, (F_2, G_2), (v_a, \overline{a}, \tau), (a)) \in \mathsf{SC}$, where

$$G_2(a, \tau) = (a, \mathsf{eq}_\tau)$$
$$F_2(a(x), \mathsf{eq}_\tau(x)) = \mathsf{eq}_\tau(x) \cdot a(x)$$

– $(\mathsf{pp}, (F_3, G_3), (v_b, \overline{b}, \tau), (b)) \in \mathsf{SC}$, where

$$G_3(b, \tau) = (b, \mathsf{eq}_\tau)$$
$$F_3(b(x), \mathsf{eq}_\tau(x)) = \mathsf{eq}_\tau(x) \cdot b(x)$$

– $(\mathsf{pp}, (F_4, G_4), (v_c, \overline{c}, \tau), (c)) \in \mathsf{SC}$, where

$$G_4(c, \tau) = (c, \mathsf{eq}_\tau)$$
$$F_4(c(x), \mathsf{eq}_\tau(x)) = \mathsf{eq}_\tau(x) \cdot c(x)$$

– $(\mathsf{pp}, (F_5, G_5), \mathsf{u}, \mathsf{w}) \in \mathsf{SC}$, where

$$T = v_a + c \cdot v_b + c^2 \cdot v_c$$
$$\mathsf{u} = (T, (\overline{L_\mathsf{r}}, \overline{L_\mathsf{c}}, \overline{w}, \overline{\mathsf{val}_A}, \overline{\mathsf{val}_B}, \overline{\mathsf{val}_C}), (\mathsf{x}, u, c))$$
$$\mathsf{w} = (L_\mathsf{r}, L_\mathsf{c}, w, \mathsf{val}_A, \mathsf{val}_B, \mathsf{val}_C)$$
$$G_5(\mathsf{w}, (\mathsf{x}, u, c)) = (L_\mathsf{r}, L_\mathsf{c}, z = (w, u, \mathsf{x}), \mathsf{val})$$
$$\mathsf{val} = \mathsf{val}_A + c \cdot \mathsf{val}_B + c^2 \cdot \mathsf{val}_C$$
$$F_5(L_\mathsf{r}(x), L_\mathsf{c}(x), z(x), \mathsf{val}(x)) = L_\mathsf{r}(x) \cdot \mathsf{val}(x) \cdot L_\mathsf{c}(x)$$

– $(\mathsf{pp}, (\overline{\mathsf{row}}, \overline{L_\mathsf{r}}), (\tau, \mathsf{row}, \mathsf{L_r})) \in \mathsf{SLKP}$

– $(\mathsf{pp}, (\overline{w}, \overline{\mathsf{col}}, \overline{L_\mathsf{c}}), (\mathsf{x}, u, \mathsf{col}, \mathsf{L_c})) \in \mathsf{PLKP}$

Proof Intuition. The construction above essentially streamlines the checks done in Spartan [59]. The first sum-check instance checks if the constraints are satisfying assuming that polynomials a, b, c are correct. The last sum-check instance recomputes $(\widetilde{Az}(\tau), \widetilde{Bz}(\tau), \widetilde{Cz}(\tau))$ assuming the validity of L_r and L_c. By the construction of val polynomials, this sum-check instance recomputes the claimed values via preprocessed R1CS matrices and w. The two lookup checks validate the correctness of $(L_\mathsf{r}, L_\mathsf{c})$ using preprocessed address vectors. We still need to validate the correctness of the provided (v_a, v_b, v_c) values. One option is to invoke the polynomial evaluation argument, but this ends up querying polynomials at two different locations: some polynomials at τ and others at r, where r is chosen over the course of the sum-check protocol. Instead, we express the multilinear polynomial evaluation as a degree-2 sum-check instance. At the end of the single sum-check, all polynomials including (a, b, c) are queried at r. □

5.2. DelegatedSpartan: SNARKs over a non-pairing-friendly curve

Spartan [59, §5.1] can be phrased as a RoK from CRR1CS to MPE^2. There are two instances in the output relation because there are two commitments in the input instance.

Lemma 5.1 (Spartan's NARK [59])**.** *For instances in* CRR1CS*, let m denote the number of constraints and n denote the number of variables. There exists a RoK from* CRR1CS *to* MPE^2 *with the following efficiency characteristics. The prover's work is $O(m + n)$ field operations; the verifier's work is $O(\log m + \log n)$ field operations, plus the cost to evaluate the multilinear extension (MLE) of R1CS matrices at a random point; and the communication complexity is $O(\log m + \log n)$ finite field elements.*

To prove two MPE instances $(\mathsf{u}_1, \mathsf{u}_2)$, we do the following. We cannot use HyperKZG because E_2 is not pairing-friendly.

(1) Reduce polynomial evaluations to circuit satisfiability. We design an R1CS circuit, polyeval, over the scalar field of E_1 that checks the purported instance-witness pairs and places the corresponding instances in the public IO. All inputs are taken non-deterministically and the circuit hardcodes the public parameters pp. Note that all steps except for steps (3) and (4) are represented without any field emulation.

$\underline{\mathsf{polyeval}(Q_1, Q_2, x_1, x_2) \to (q_1, q_2, x, y_1, y_2)}$
(1) compute $q_1 \leftarrow \mathsf{Com}(\mathsf{pp}, Q_1)$
(2) compute $q_2 \leftarrow \mathsf{Com}(\mathsf{pp}, Q_2)$
(3) compute $y_1 \leftarrow Q_1(x_1)$
(4) compute $y_2 \leftarrow Q_2(x_2)$
(5) output $(q_1, q_2, x_1, x_2, y_1, y_2)$

Using u_1 and u_2, the verifier constructs a committed relaxed R1CS instance $\mathsf{u}_{\mathsf{polyeval}}$. Then, it is easy to see that the satisfiability of u_1, u_2 is tantamount to the knowledge of satisfying witnesses to $\mathsf{u}_{\mathsf{polyeval}}$.

(2) Prove the circuit with MicroSpartan. Since the R1CS circuit is defined over the scalar field of E_1, we invoke MicroSpartan with HyperKZG to produce a succinct proof with an efficient on-chain verifier.

5.3. Implemented optimizations

Efficiently evaluating the MLE of R1CS matrices in EC**.** Recall that DelegatedSpartan is run to prove the satisfiability of a small circuit containing about 1,300 constraints. This circuit computes a scalar multiplication and a point addition. Since DelegatedSpartan relies on Lemma 5.1, the verifier has to evaluate the MLE of R1CS matrices. The time-optimal algorithm takes time linear in the number of non-zero entries in R1CS matrices [59], [64], [67]. In our context, R1CS matrices have ≈6,000 non-zero entries. While a linear number of field operations is relatively inexpensive on-chain, this requires storing R1CS matrices on-chain, which is prohibitive (we attempted to do a clever encoding of R1CS matrix entries and the costs were still prohibitive).

We observe that this circuit can be made structured given that it implements many iterations of a double-add loop to compute a scalar multiplication followed by some constraints to do point addition. We introduce dummy constraints so that the R1CS matrices contain repeated copies of the same sub-matrix, one for each loop iteration, followed by some entries in the end. Now, to evaluate the MLE of these structured matrices, the verifier evaluates the MLE of the three sub-matrices (there are at most 30 non-zero entries) and the MLEs of the non-uniform portion of the matrices (which also only has a few tens of non-zero entries). Using these values, the verifier computes the MLE of the three structured matrices with only $O(\log k)$ finite field operations ($k = 128$ in our case). Overall, the on-chain verifier stores a succinct representation of these matrices and evaluates MLEs efficiently on the fly. Although this idea is folklore and described in different forms in prior work [62], implementing it required substantial circuit engineering.

Prove $\mathsf{u}_{\mathsf{polyeval}}$ with $\mathsf{U}_{F'}$ in a batch. MicroSpartan naturally provides a batched variant that proves multiple R1CS instances at once (i.e., by batching all the sum-check protocol invocations). For ease of reference, we refer to this as BatchedMicroSpartan. To prove $\mathsf{U}_{F'}$, MicroNova invokes MicroSpartan already, so we use BatchedMicroSpartan to prove $\mathsf{u}_{\mathsf{polyeval}}$ and $\mathsf{U}_{F'}$ in a single batch, saving proof sizes and verifier costs by $\approx 2\times$.

Reduce the size of polyeval circuit. A natural commitment scheme over a non-pairing-friendly curve E_2 is Pedersen's commitment scheme. In our context, the instances in MPE commit to vectors of size $n = 2,048$. This means that the polyeval circuit performs that many group scalar multiplications and finite field operations. To reduce this work, we employ a matrix commitment scheme [68]: a commitment to a vector of size n is n_1 group elements each committing to a vector of size n_2, where $n = n_1 \cdot n_2$. This reduces the size of the circuit from $O(n_1 \cdot n_2)$ to $O(n_1 + n_2)$. However, the trade-off is that the recursive verifier circuit has to perform $O(n_1)$ group scalar multiplications, rather than $O(1)$, to fold instances in E_2. To balance the two costs, we pick $n_1 = 8$ and $n_2 = 256$. We provide additional details in Appendix B.

6. HyperKZG: Proving multilinear evaluations

This section describes HyperKZG, a RoK from a multilinear polynomial evaluation instance to a collection of univariate polynomial evaluation instances. We then prove those univariate polynomial evaluation instances with univariate KZG. Together, the verifier's work is dominated by two pairing operations and a logarithmic number of group scalar multiplications.[4] Our starting point here is the reduction from Gemini [21] that reduces the task of checking a multilinear polynomial in *coefficient* form over $\log n$ variables to the task of checking evaluations of $\log n$ univariate polynomials over n coefficients. We make a small—but crucial—modification to this reduction that allows committing to multilinear polynomials in evaluation form. This in turn makes it possible to use the commitment scheme with MicroSpartan without requiring a change of basis (which entails superlinear and expensive operations such as FFTs).

6.1. Reducing multilinear to univariate evaluations

Construction 5 (An RoK from MPE to UPE). Consider a finite field $\mathbb{F}$, a commitment scheme com for vectors over $\mathbb{F}$, and a size parameter n. Let $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, n)$. Let $\ell = \log n$. We construct a reduction of knowledge from MPE to $\mathsf{UPE}^{3\log n - 1}$. That is, the prover and verifier reduce the task of checking the evaluation of a multilinear polynomial over $\log n$ variables to the task of checking $3\log n - 1$ evaluations of univariate polynomials with n coefficients.

For a vector $P \in \mathbb{F}^n$, let $P_{\mathsf{evens}} \in \mathbb{F}^{n/2}$ and $P_{\mathsf{odds}} \in \mathbb{F}^{n/2}$ denote vectors with elements from P at the even indices and odd indices respectively.

Consider an arbitrary instance in MPE $(\overline{P}, x, y)$, and suppose the prover claims that it knows polynomial P such that $(\mathsf{pp}, (\overline{P}, x, y), P) \in \mathsf{MPE}$.

1) Let $P^{(0)} = P$. The prover computes $\forall i \in \{1, \ldots, \ell - 1\}$
$$P^{(i)} = (1 - x_i) \cdot P^{(i-1)}_{\mathsf{evens}} + x_i \cdot P^{(i-1)}_{\mathsf{odds}}$$
The prover sends commitments $(\overline{P}^{(1)}, \ldots, \overline{P}^{(\ell-1)})$.[5]
2) The verifier sends a random challenge $r \in \mathbb{F}$.
3) The prover sends claimed evaluations, treating each vector as coefficients of a univariate polynomial of appropriate degree. For all $i \in \{1, \ldots, \ell\}$:
$$y^{(i)} = P^{(i)}(r^2)$$
$$y^{(i-1)}_{\mathsf{pos}} = P^{(i-1)}(r)$$
$$y^{(i-1)}_{\mathsf{neg}} = P^{(i-1)}(-r)$$
4) The verifier checks that for all $i \in \{1, \ldots, \ell\}$
$$y^{(i)} = (1 - x_i) \cdot \frac{y^{(i-1)}_{\mathsf{pos}} + y^{(i-1)}_{\mathsf{neg}}}{2} + x_i \cdot \frac{y^{(i-1)}_{\mathsf{pos}} - y^{(i-1)}_{\mathsf{neg}}}{2 \cdot r}$$
The verifier checks that $y^{(\ell)} = y$.
5) Let $\overline{P}^{(0)} = \overline{P}$. The prover and the verifier output a collection of instance-witness pairs in UPE (the verifier only outputs instances): for $i \in \{1, \ldots, \ell - 1\}$
$$(\mathsf{pp}, (\overline{P}^{(i)}, r^2, y^{(i)}), P^{(i)}) \in \mathsf{UPE},$$

4. There are prior KZG-type commitment schemes for multilinear polynomials [29], [57] but they require a logarithmic number of pairings. A pairing consumes $10\times$ higher gas costs than a scalar multiplication, resulting in a verifier that is about $3\times$ higher gas costs than HyperKZG's on Ethereum.

5. Observe that $P^{(i)}$ corresponds to partially evaluating P as a multilinear polynomial at $(x_1, \ldots, x_i)$. Each step i corresponds to a step of the evaluation algorithm from [67, Section 5.1] (also described in [65, Lemma 3.8]), for polynomials in evaluation form.

and for $i \in \{0, \ldots, \ell - 1\}$

$$(\mathsf{pp}, (\overline{P}^{(i)}, r, y^{(i)}_{\mathsf{pos}}), P^{(i)}) \in \mathsf{UPE}$$

$$(\mathsf{pp}, (\overline{P}^{(i)}, -r, y^{(i)}_{\mathsf{neg}}), P^{(i)}) \in \mathsf{UPE}$$

Lemma 6.1. *For a size bound n, Construction 5 is a reduction of knowledge from* MPE *to* $\mathsf{UPE}^{3\log n - 1}$ *with the following efficiency characteristics. The prover's cost is dominated by $\sum_{i=0}^{\ell-1} |\mathsf{com}(2^i)|_{\mathsf{p}}$ where $|\mathsf{com}(2^i)|_{\mathsf{p}}$ denotes the cost of committing to a vector of size 2^i, and the cost of computing $P^{(i)}$ for all $i \in \{1, \ldots, \ell - 1\}$ The communication cost is dominated by $(\sum_{i=0}^{\ell-1} |\mathsf{com}(2^i)|_{\mathsf{c}}) + 3\log n \cdot \mathsf{F}$ where $|\mathsf{com}(2^i)|_{\mathsf{c}}$ denotes the size of a commitment for a vector of size 2^i and F denotes the size of a field element, The verifier's cost is dominated by $O(\log n)$.*

Proof (sketch). The claimed efficiency is easy to check. Perfect completeness is also easy to check. This follows immediately from the correctness of the multilinear polynomial evaluation algorithm in the Lagrange basis (i.e., multilinear polynomials represented in their evaluation form over a Boolean hypercube) on which the protocol is based (see [65, Lemma 3.8]), and the correctness of the verifier's checks in the RoK, which holds for any random challenge $r \in \mathbb{F}$ when the prover is honest.

We prove knowledge soundness as follows. Consider an adversary $\mathcal{A}$ that adaptively picks the input instance and a malicious prover $\mathcal{P}^\star$ that succeds with probability ϵ. Let $n \in \mathbb{N}$ denote a size parameter. Let $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, n)$. Suppose on input a random tape r, the adversary $\mathcal{A}$ picks an MPE instance $(\overline{P}, x, y)$ and some auxiliary state st. We construct an extractor $\mathcal{E}$ that succeds with probability $\epsilon - \mathsf{negl}(\lambda)$ in obtaining satisfying witness for the original instance.

On input r, $\mathcal{E}$ first obtains the following tuple from the adversary:

$$((\overline{P}, x, y), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{r})$$

The extractor $\mathcal{E}$ runs $\langle \mathcal{P}^\star, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (d, v), \mathsf{st})$ to obtain the following instance-witness pairs, where $i \in \{1, \ldots, \log n$.

$$(\mathsf{pp}, (\overline{P}^{(i)}, r^2, y^{(i)}), P^{(i)}) \in \mathsf{UPE}$$

$$(\mathsf{pp}, (\overline{P}^{(i-1)}, r, y^{(i-1)}_{\mathsf{pos}}), P^{(i-1)}) \in \mathsf{UPE}$$

$$(\mathsf{pp}, (\overline{P}^{(i-1)}, -r, y^{(i-1)}_{\mathsf{neg}}), P^{(i-1)}) \in \mathsf{UPE}$$

The extractor outputs $P^{(0)}$ as the witness for the input instance. Since the extractor runs $\mathcal{P}^\star$ only once, it runs in expected polynomial time. We must now argue that $P^{(0)}$ is a satisfying witness for the MPE instance $(\overline{P}, x, y)$ with probability $\epsilon - \mathsf{negl}(\lambda)$. This follows from an anlysis similar to our base protocol's analysis [21, Lemma 5.4]. □

6.2. Proving UPE instances with batched KZG

To prove instances in UPE, we invoke batched KZG [19], [44]. For uniformity, we batch $3\log n$ evaluations rather than $3\log n - 1$ evaluations as depicted i.e., we evaluate each of the $\log n$ univariate polynomials at $r, -r$, and r^2. Our approach to batching leverages the particular characteristics of our setting. Suppose $p_1, \ldots, p_k$ are polynomials in $\mathbb{F}[X]$ and $(u_1, \ldots, u_t)$ are values in $\mathbb{F}$ and $v_{i,j}$ are the corresponding evaluations such that $p_i(u_j) = v_{i,j}$. In our context, $t = 3$ (since each polynomial is evaluated at three points) and $k = \log n$.

Let $C_1, \ldots, C_k$ be commitments to p_i. Call this a (k, t)-polynomial relation. A prover can convince a verifier that the relation holds, more efficiently than the $O(kt)$ direct solution, by using two types of batching (in order):

1) Fold the k polynomials into a single polynomial, by taking a random linear combination, thus creating a $(1, t)$-polynomial relation instance.
2) Batch the t evaluations for a single polynomial directly.

Step 1. The verifier sends a random value $q \in \mathbb{F}$ to the prover, who computes the following polynomial

$$B(X) = p_1(X) + q \cdot p_2(X) + q^2 \cdot p_3(X) + \ldots + q^{k-1} \cdot p_k(X)$$

The verifier computes $C_B = \sum_{i=1}^{k} q^{i-1} C_i$ locally (since the commitments are linearly homomorphic). Similarly, for the evaluation of B at any of the points u_i, we have

$$\begin{aligned} B(u_i) &= q^0 \cdot p_1(u_i) + q^1 \cdot p_2(u_i) + \ldots + q^{k-1} \cdot p_k(u_i) \\ &= v_{1,i} + q \cdot v_{2,i} + \ldots + q^{k-1} \cdot v_{k,i} \end{aligned}$$

which $\mathcal{V}$ can also compute locally. Now $(\mathcal{P}, \mathcal{V})$ have a $(1, t)$ instance, where $B(X)$ is the committed polynomial and $(u_i)_{i=1}^t$ are the t points. The cost to the verifier in of Step 1 is $t - 1$ group scalar multiplications.

Step 2. In this step we batch the verifier's work to check the t openings of $B(X)$ using the details of verifying KZG polynomial commitments [44]. Compute the t witnesses $W_i = \mathsf{KZG.Open}(\mathsf{ck}, C_B, u_i, B(u_i))$. Checking these individually entails checking:

$$e(C_B - B(u_i)G + u_i W_i, H) = e(W_i, \tau H) \tag{1}$$

for all $i \in [t]$. A well-known batching technique [14] applied to pairing-based signatures [36, Technique 3] is as follows. Suppose the verification of t items is of the form

$$e(L_1, H) = e(R_1, H') \ \wedge \ \ldots \ \wedge \ e(L_t, H) = e(R_t, H') \ .$$

$\mathcal{V}$ instead samples a random $(d_2, \ldots, d_t)$ and checks

$$\begin{aligned} e(L_1, H) e(L_2, H)^{d_2} \cdots e(L_t, H)^{d_t} = \\ e(R_1, H') e(R_2, H')^{d_2} \cdots e(R_t, H')^{d_t} \\ e(L_1, H) e(d_2 L_2, H) \cdots e(d_t L_t, H) = \\ e(R_1, H') e(d_2 R_2, H') \cdots e(d_t R_t, H') \\ e(L_1 + d_2 L_2 + \ldots + d_t L_t, H) = \\ e(R_1 + d_2 R_2 + \ldots + d_t R_t, H') \ . \end{aligned}$$

When we set L_i and R_i as in Equation (1), the verifier's cost in Step 2 is $4 \cdot (t-1)$ scalar multiplications in $\mathbb{G}_1$ and two pairings. In HyperKZG, we have $k = \ell$ polynomials and $t = 3$ points so Step 1 costs $\ell - 1$ scalar multiplications and Step 2 costs 8 scalar multiplications and two pairings, for a total of $\ell + 7$ scalar multiplications and two pairings. In Equation (1) we re-arranged the KZG verification equation (following [43]) so that all scalar multiplications are in $\mathbb{G}_1$, and a single pairing is required for verification, allowing us to make use of Ethereum's pre-compiled contracts.

Additional optimizations. For Step 2, we could employ the approach of Boneh et al. [19] that batches multiple polynomials at multiple (possibly different) points. However, in our context, the number of evaluations is only three, so there is limited gain to the prover and the verifier (the prover would compute two MSMs of size n instead of three in our current version) and the verifier's work will be about the same. The verifier is a bit more complex with their approach. Given the limited benefits, we do not implement this optimization.

7. Implementation

We implement MicroNova on top of Nova [6] as a library in about 11,000 lines of Rust. For a curve cycle, we use BN254/Grumpkin [5]; it also supports Pallas/Vesta [8] and Secp/Secq curve cycles [5]. For a circuit-friendly hash function, we use Poseidon [40]. For a on-chain-efficient hash function, we use Keccak256. The APIs of MicroNova accept a step function F as a circuit expressed with `bellpepper` [1]. The library also implements BatchedMicroSpartan and DelegatedSpartan to compress an IVC proof produced by MicroNova, as well as implementations of (non-batched) MicroSpartan and the baseline Spartan to compress an IVC proof of Nova. For polynomial commitment schemes, the library implements HyperKZG for pairing-friendly curves (e.g., BN254), and Pedersen commitments that can serve all curves (e.g., Grumpkin).

We also implement MicroSpartan as a *stand-alone* proof system alongside MicroNova and Nova, targeting scenarios that does not require IVC. To build such stand-alone prover and verifier, we design a straightforward augmented circuit that, given step F and z_{i-1}, compute $z_i \leftarrow F(z_{i-1})$, and outputs $\mathsf{x} \leftarrow (z_i, z_{i-1})$ into R1CS public IO.

On-chain verifier. We implement MicroNova's on-chain verifier as a library in about 3,300 lines of Solidity. The library also provides an on-chain verifier for the stand-alone MicroSpartan and HyperKZG. Both MicroNova and the stand-alone MicroSpartan are over BN254. We use Ethereum's EC precompiles to support EC operations (i.e., `ecAdd`, `ecMul` and `ecPairing`) [2]. Fiat-Shamir transcript is built on Keccak256.

To send proofs on-chain, we use `serde` [9]. We wrote a deserializer in Solidity to reconstruct appropriate data structures. We customize the serialization so the elliptic curve points are in affine form, prime field elements are in big-endian format, and other values are encoded in fixed size byte arrays. This avoids expensive on-chain conversions. The on-chain verifier also accepts a serialized verifier key as an argument. In more detail, the on-chain verifier receives a byte sequence containing a verifier key, a proof, and the public IO (e.g., z_0, z_n, number of steps). After receiving the sequence, the on-chain verifier inexpensively deserializes it into data structures, and then verifies the proof. The verifier reverts if any step in the verification fails.

8. Experimental evaluation

In this section, we experimentally evaluate MicroNova. To evaluate the cost of achieving a succinct on-chain verifier, we compare MicroNova with Nova. In addition to evaluating the full system, we also provide microbenchmarks for the underlying components of MicroNova: HyperKZG and stand-alone MicroSpartan with HyperKZG as its polynomial commitment scheme. These microbenchmarks will provide enough context to understand the performance of MicroNova, and are in Section 8.4.

8.1. Metrics and testbed

Our principal evaluation metrics are: (1) the prover's cost to produce a proof; (2) the verifier's cost to verify a proof on a CPU; (3) the verifier's cost to verify a proof on-chain; (4) the size of a proof on a CPU; and (5) the size of a proof on-chain. For (1) and (2), we use the wall clock time, and for (3), we measure gas costs on the Ethereum virtual machine (EVM). For (4) and (5), we report length in bytes, but by serializing proof data structures differently. For example, proofs verified on a CPU have elliptic curve points in compressed form (decompression of points is relatively inexpensive on a CPU, but expensive on-chain).

For our measurements, we use an Azure Fsv2-series VM of size Standard F64s_v2 (64 vCPUs, 2.70 GHz Intel(R) Xeon(R) Platinum 8168, and 128 GiB memory).

8.2. Methodology and parameters

We report performance with varying step circuit sizes $|F|$. We use a synthetic circuit, from prior work [52], [59], that contains $|F| = n_c$ R1CS constraints and n_v R1CS variables, and maintains $n_c/n_v \approx 1$. The circuit computes $y = x^{2^{n_c}}$ naively with repeated multiplications on a prime field with n_c constraints. We use a public IO of length 1, i.e., $|\mathsf{x}| = 1$. We set the number of IVC steps proven to 10.

We use `cargo bench` to measure metrics (1) and (2). For metric (4), we serialize the proof structure with `serde` [9] and then measure the number of bytes after compressing the serialized proof string with `ZlibEncoder` [3]. For metric (5), we serialize proofs as described in Section 7

To compile MicroNova's Solidity verifier, we use Foundry [4] with Solidity command-line compiler `solc` version 0.8.25 on EVM version `cancun`. We configure `solc` optimizer

	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}
Nova	0.91	1.77	3.52	6.62	12.5	23.0	46.4
MicroNova	2.45	3.21	4.77	7.75	12.9	24.6	48.7

Figure 1: Prover's total cost on a CPU (in seconds) to produce an IVC proof of 10 steps of varying step-circuit sizes.

	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}
Nova[1]	1.30	2.39	4.44	8.49	16.7	33.2	66.1
Nova[2]	0.45	0.68	1.08	1.93	3.51	6.73	12.9
Nova[3]	5.31	10.4	20.3	39.9	78.5	155	310
Nova[4]	2.02	3.87	7.18	13.5	26.1	50.1	100
MicroNova	28.5	29.9	30.5	33.7	40.8	66.7	119

Figure 2: Prover's cost on a CPU (in seconds) to compress an IVC proof of varying step-circuit sizes.

with flags `--via-ir` and `--optimizer-runs=50000`. To measure gas costs, we use `gasleft()`.

8.3. Evaluation results of MicroNova and Nova

We first measure the size of MicroNova's verifier circuit, as it determines *recursion overheads*: the number of additional constraints that the prover must prove at each incremental step besides proving an invocation of F. We find that MicroNova's verifier circuit is $\approx 74,000$ R1CS constraints on the pairing-friendly curve, and $\approx 1,300$ on the non-pairing-friendly curve. Precisely, the circuit is of 74,352 constraints on BN254 and 2,037 on Grumpkin, while the latter includes dummy constraints (§5.3), which does not cost the prover or the verifier, to make the circuit uniform. As a comparison, Nova's verifier circuit is $\approx 10,000$ R1CS constraints on each curve in the cycle (9,949 and 10,502 constraints on BN254 and Grumpkin curve respectively). That is, MicroNova's recursion overhead is about $3\times$ higher than Nova's, but this overhead is negligible when $|F|$ is sufficiently large, which is the case in our target applications (e.g., a Rollup).

When varying the step circuit size $|F|$, we pick the number of constraints n_c so that when augmented with MicroNova's verifier circuit, the total number of constraints in F' equals to a power of 2. For ease of depiction, we use these powers of 2 to denote $|F|$ e.g., when F' has 2^{20} constraints, $|F| = n_c = 2^{20} - 74,352 = 974,224$, and we will use $|F| \approx 2^{20}$ in figures to represent $|F| = 974,224$. Note that MicroNova and Nova *prove* the same F in our experiments (sizes of augmented circuits F' of MicroNova and Nova are different due to their different recursion overheads).

We instantiate Nova over BN254/Grumpkin. For proof compression, Nova library implements non-preprocessing Spartan [59], which does not provide succinct verification, let alone on-chain verification (the verifier's work is linear in the circuit size). We refer to this version of Spartan as SpartanNARK. To provide a full context, we experiment with different configurations. For its IVC proof component on Grumpkin curve, Nova compresses it with SpartanNARK and with an IPA-based polynomial commitment scheme. For its proof of IVC on BN254 curve, Nova can compress it with either SpartanNARK or MicroSpartan, and with either IPA-PC or HyperKZG as the polynomial commitment scheme. We use the following notation to refer to different variants of Nova for brevity: Nova[1]—Nova with SpartanNARK and IPA-PC; Nova[2]—Nova with SpartanNARK and HyperKZG; Nova[3]—Nova with MicroSpartan and IPA-PC; and Nova[4]—Nova with MicroSpartan and HyperKZG.

	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}
Nova[1]	78.0	125	204	397	768	1.4s	2.8s
Nova[2]	25.4	26.8	30.6	43.6	77.0	148	288
Nova[3]	188	383	690	1.3s	2.5s	5.0s	10s
Nova[4]	20.9	21.2	20.7	21.5	21.5	21.5	21.0
MicroNova	13.6	13.7	13.7	13.7	13.8	13.6	13.7

Figure 3: Verifier's cost on a CPU (in milliseconds) to verify a compressed IVC proof of varying step-circuit sizes. Entries with "s" are in seconds.

Prover. Figure 1 depicts the prover's *total* cost to produce a proof of a ten-step IVC (four variants of Nova are exactly the same for the IVC prover). Figure 2 depicts the prover's cost to compress an IVC proof.

MicroNova's prover's cost to produce and to compress an IVC proof scale roughly linearly with $|F|$. When $|F|$ is small, MicroNova incurs higher overheads than Nova, but the overhead drops as $|F|$ increases. This is because (1) Nova's verifier's circuit on both curves combined is smaller than MicroNova's, and (2) when MicroNova compresses an IVC proof the BatchedMicroSpartan proves a constant-sized R1CS polyeval of 1.7M constraints that proves 2 MPE instances on Grumpkin curve (§5.3). For MicroNova's prover to compress an IVC proof, such constant cost is dominating until $|F| \approx 2^{21}$, then the R1CS of IVC is no longer smaller than the R1CS of polyeval. At $|F| \approx 2^{21}$, MicroNova's IVC prover overhead over Nova is only 3.2%.

Note that Nova using SpartanNARK has a better prover time than with MicroSpartan because the former incurs linear verification costs whereas the latter does not. These results also confirm that HyperKZG is significantly more efficient than IPA-based polynomial commitment scheme.

Verifier and proof sizes. Figure 3 depicts the verifier's cost to verify a compressed IVC proof on a CPU. The cost under the first three Nova instantiation scale close to linearly with $|F|$. MicroNova's verifier cost is the lowest: < 14 ms (Nova with MicroSpartan and HyperKZG benefits from techniques in this work and performs better than other Nova variants).

Figure 4 depicts the on-chain verifier's costs for MicroNova (we do not depict Nova's costs as it is impractical to verify Nova's proof on-chain). We can see that both the gas cost and proof length remains (almost) unchanged up to $|F| \approx 2^{21}$ (this is due to fixed-sized polyeval circuit proven by MicroNova). Then, they have a logarithmic growth with $|F|$: when $|F|$ increases from 2^{21} to 2^{22}, or from 2^{22} to

	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}
proof size	12.8	12.8	12.8	12.8	12.8	13.1	13.4
gas cost	2.23	2.22	2.22	2.22	2.22	2.25	2.28

Figure 4: MicroNova's on-chain proof size (in KB) and the verifier's on-chain cost (in M gas) to verify a compressed IVC proof of varying step-circuit sizes. The length of the verifier's key on-chain is 1,368 bytes regardless of $|F|$.

	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}
Nova[1]	10.3	10.6	10.8	11.1	11.4	11.7	12.0
Nova[2]	11.4	11.8	12.1	12.5	12.8	13.2	13.5
Nova[3]	9.04	9.20	9.37	9.53	9.70	9.86	10.0
Nova[4]	10.3	10.5	10.8	11.0	11.2	11.4	11.7
MicroNova	10.8	10.8	10.8	10.8	10.8	11.0	11.2

Figure 5: MicroNova's and Nova's proof sizes on a CPU (in KB).

2^{23}, the gas cost of the on-chain verifier only increases by 32K ($< 2\%$), and the proof length only grows by 264 bytes ($\approx 2\%$) each time.

Figure 5 depicts the size of compressed IVC proofs of MicroNova and Nova on a CPU. MicroNova's proof is about 10.8 KB, until $|F| \approx 2^{21}$, and then has a logarithmic growth with $|F|$, but still remains small. The different Nova variants have similar proof sizes despite using IPA-based polynomial commitment scheme because proof sizes are similar under both IPA-PC and HyperKZG.

8.4. Additional benchmarks

Decomposing MicroNova's on-chain costs. Figure 6 depicts the on-chain gas costs of four components of MicroNova's verifier: (1) deserializer; (2) DelegatedSpartan's verifier; (3) BatchedMicroSpartan's verifier; and (4) HyperKZG verifier, which is included in (3).

First, the deserializer's cost scales linearly with the proof length (in our implementation, the deserializer also handles verifier's key and other parameters, which are constant-sized). DelegatedSpartan verifier has a constant cost. BatchedMicroSpartan and HyperKZG's verifiers' costs exhibit the same trend as MicroNova's verifier—costs nearly unchanged until $|F| \approx 2^{21}$ and then have a logarithmic scaling. Second, BatchedMicroSpartan contributes ≈65% of the cost of MicroNova's verifier; it also contributes a vast majority of MicroNova's gas cost increase when the circuit grows. Within BatchedMicroSpartan, ≈31% is incurred by HyperKZG verifier. The DelegatedSpartan verifier contributes $< 25\%$ of MicroNova's cost, while the deserializer only contributes 6%.

Stand-alone MicroSpartan.

When varying the size of the step circuit $|F|$, we use the same set of $|F| = n_c$ we have picked when evaluating MicroNova (§8.3). We have the stand-alone MicroSpartan prove and verify 10 continuous steps of F one after another.

	2^{17}	2^{18}-2^{21}	2^{22}	2^{23}
MicroNova	2.23	2.22	2.25	2.28
Deserializer	0.126	0.126	0.128	0.130
DelegatedSpartan	0.552	0.552	0.552	0.552
BatchedMicroSpartan	1.44	1.43	1.46	1.49
HyperKZG	0.447	0.447	0.459	0.471

Figure 6: Cost of four components of MicroNova's on-chain verifier (in M gas). HyperKZG is included in BatchedMicroSpartan while the deserializer or DelegatedSpartan is not. Four columns of $|F| \approx 2^{18}$, $|F| \approx 2^{19}$, $|F| \approx 2^{20}$ and $|F| \approx 2^{21}$ are merged into one as data is (nearly) the same.

	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}
Prove	1.18	3.94	7.36	13.9	26.6	51.9	104
Verify[1]	5.09	5.12	5.15	5.20	5.23	5.26	5.29
proof[1]	5.18	5.63	5.86	6.08	6.31	6.53	6.76
Verify[2]	0.81	0.86	0.88	0.90	0.93	0.95	0.98
proof[2]	6.25	6.78	7.04	7.30	7.57	7.83	8.10

Figure 7: Performance results of stand-alone MicroSpartan per-step average, from the first row to the last: (1) prover's cost (in seconds); (2) verifier's cost on CPU (in milliseconds); (3) proof length on CPU (in KB); (4) verifier's cost on chain (in M gas); and (5) proof length on chain (in KB). The length of the verifier's key on-chain is always 824 bytes.

	2^{15}	2^{17}	2^{19}	2^{21}	2^{23}	2^{25}
commit	0.02	0.05	0.15	0.53	1.8	7.1
prove	0.08	0.24	0.75	2.6	9.7	36
verify on CPU	2.7	2.9	2.9	2.9	3.0	3.1
proof on CPU	2.0	2.3	2.6	2.8	3.1	3.3
verify on-chain	0.35	0.37	0.39	0.42	0.44	0.46
proof on-chain	2.6	2.9	3.2	3.5	3.9	4.2

Figure 8: HyperKZG's performance for polynomials of various length: (1) commit cost on CPU (in seconds); (2) prover's cost (in seconds); (3) verifier's cost on CPU (in milliseconds); (4) proof length on CPU (in KB); (5) verifier's cost on-chain (in M gas); and (6) proof length on chain (in KB). Length of the verifier's key on-chain is always 320 bytes.

Figure 7 shows the stand-alone MicroSpartan's *average* performance results of 10 steps. Throughout the 10 steps of the same $|F|$, MicroSpartan's performance fluctuates negligibly ($< 2\%$ of the average). The cost of the stand-alone MicroSpartan prover scales close to linearly with $|F|$. On- and off-chain verifier cost, the proof length on a CPU, and the proof length on-chain have a logarithmic scaling with $|F|$.

HyperKZG. We vary the size of polynomials from 2^{15} to 2^{25}, which is the range of the size of committed polynomials when evaluating MicroNova, stand-alone MicroSpartan, and Nova with HyperKZG in §8.3. As shown in Figure 8, HyperKZG's cost to commit and to prove an evalution both grow close to linearly with the size of the polynomial, and the HyperKZG's verifier's cost grows logarithmically.

9. Discussion of naive solutions

This section discusses naive solutions to the problem that MicroNova addresses: a folding-based IVC scheme with an efficient verifier. In theory, we can compose an existing folding-based IVC scheme (e.g., Nova) with an off-the-shelf SNARK [38], [41]: we represent Nova's verifier as a circuit and then prove the knowledge of a valid witness to that circuit. This achieves an efficient verifier, but it makes the prover multiple orders of magnitude slower than MicroNova's prover.

Recall that Nova's IVC proof contains an instance-witness pair on pairing curve (e.g., BN254) and an instance-witness pair on a non-pairing curve (e.g., Grumpkin). With Groth16 or Plonk (instantiated with BN254), we will be using them to prove a circuit that checks if *both* instances are satisfying. For a step circuit used in Nova with a 10^6 gates, the Nova verifier as a circuit, will be at least a billion to a trillion gates due to the need to emulate BN254 scalar multiplication operations (at least one per gate in the step circuit) inside a circuit defined over BN254.[6] Note that half-pairing cycles do not directly help as Groth16 and Plonk need pairing-friendly curves. Known techniques can bring this to a few hundred million gates. However, the prover times are impractical.

MicroNova contains a set of techniques to make the above naive solution practical. Indeed, MicroNova's techniques make the circuit proven with a SNARK significantly smaller ($\approx$1.7 million gates). Additionally, we use more efficient techniques (e.g., sum-checks) than Plonk's (e.g., quotient-checks) to make the prover concretely efficient. Indeed, the prover times for compression are only tens of seconds (Figure 2).

9.1. Concurrent work

Sonobe [10] is a library that implements Nova [52] and other folding schemes. Sonobe's focus is on modularity, so it is not fully optimized for performance. Like MicroNova, Sonobe also focuses on producing Nova proofs that are efficiently verifiable on-chain. Sonobe crucially relies on our prior work CycleFold [49], which was discovered while designing MicroNova.

However, there are several differences between our approach and their approach. First, Sonobe employs Groth16 [41] to compress IVC proofs given Groth16 provides an efficient on-chain verifier. So, Sonobe's requires a separate trusted setup for each application proven via Sonobe. In contrast, MicroNova requires only a universal trusted setup. In fact, MicroNova does not require any new trusted setup: it can use any existing trusted setup material created for other popular proof systems such as Plonk [38]. Second, Sonobe is more gas efficient: verifying a proof requires about 800,000 gas on Ethereum virtual machine. Whereas, MicroNova's proofs require 2.2 M gas since its proofs are longer and more complex to verify than the proof generated by Sonobe. However, one can apply further compression to MicroNova's proofs using Groth16 to further compress MicroNova's proofs—while retaining the trusted setup property of MicroNova. Third, to produce a compressed proof, Sonobe's approach requires proving a circuit of size $\approx$10 M constraints via Groth16. In contrast, MicroNova's approach only requires proving a circuit of size $\approx$1 M constraints via MicroSpartan. We leave it to the future work to provide a more detailed comparison.

6. Each non-native scalar multiplication takes a million gates and a native scalar multiplication requires a few thousand gates.

Acknowledgments

We thank Craig Costello and Michael Naehrig for helpful discussions on various aspects of elliptic curves. We thank Abhiram Kothapalli for conversations about some of the RoKs introduced in this paper. We thank Sebastian Angel for his contributions during the early stages of this project. Finally, we thank Wilson Nguyen, Carlos Pérez, Jay White, and the anonymous IEEE S&P reviewers for comments on a prior version of this paper.

References

[1] bellpepper. https://crates.io/crates/bellpepper

[2] EVM precompiled contracts. https://www.evm.codes/precompiled

[3] flate2. https://crates.io/crates/flate2

[4] Foundry. https://getfoundry.sh/

[5] halo2curves. https://crates.io/crates/halo2curves

[6] Nova: Recursive SNARKs without trusted setup. https://github.com/Microsoft/Nova

[7] Ova: A slightly better Nova. https://hackmd.io/V4838nnlRKal9ZiTHiGYzw

[8] Pasta curves. https://crates.io/crates/pasta_curves

[9] Serde. https://crates.io/crates/serde

[10] sonobe. https://github.com/privacy-scaling-explorations/sonobe/

[11] Awesome Folding. https://github.com/lurk-lab/awesome-folding (2023)

[12] Arun, A., Setty, S.: Nebula: Efficient read-write memory and switchboard circuits for folding schemes. Cryptology ePrint Archive (2024)

[13] Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: STOC (1991)

[14] Bellare, M., Garay, J.A., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: EUROCRYPT. pp. 236–250 (1998)

[15] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: CRYPTO (2014)

[16] Boneh, D.: ZKP MOOC Lecture 10: Recursive SNARKs. https://www.youtube.com/watch?v=0LW-qeVe6QI (2023)

[17] Boneh, D., Bünz, B., Fisch, B.: A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712 (2018)

[18] Boneh, D., Chen, B.: LatticeFold: A lattice-based folding scheme and its applications to succinct proof systems. Cryptology ePrint Archive, Paper 2024/257 (2024)

[19] Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive (2020)

[20] 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)

[21] Bootle, J., Chiesa, A., Hu, Y., Orrú, M.: Gemini: Elastic SNARKs for diverse environments. In: EUROCRYPT. pp. 427–457 (2022)

[22] Bootle, J., Chiesa, A., Sotiraki, K.: Sumcheck arguments and their applications. In: CRYPTO. pp. 742–773 (2021)

[23] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: S&P (2018)

[24] Bünz, B., Chen, B.: Protostar: Generic efficient accumulation/folding for special sound protocols. In: ASIACRYPT (2023)

[25] Bünz, B., Chiesa, A., Lin, W., Mishra, P., Spooner, N.: Proof-carrying data without succinct arguments. In: CRYPTO (2021)

[26] Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Proof-carrying data from accumulation schemes. In: TCC (2020)

[27] Bünz, B., Chen, J.: Proofs for deep thought: Accumulation for large memories and deterministic computations. Cryptology ePrint Archive, Paper 2024/325 (2024)

[28] Bünz, B., Mishra, P., Nguyen, W., Wang, W.: Accumulation without homomorphism. Cryptology ePrint Archive, Paper 2024/474 (2024)

[29] Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In: EUROCRYPT (2023)

[30] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: EUROCRYPT (2020)

[31] Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: EUROCRYPT (2020)

[32] Dimitriou, N., Garreta, A., Manzur, I., Vlasov, I.: Mova: Nova folding without committing to error terms. Cryptology ePrint Archive, Paper 2024/1220 (2024)

[33] Drake, J.: ZK Whiteboard Sessions - Module Fourteen: Nova Crash Course with Justin Drake. https://www.youtube.com/watch?v=SwonTtOQzAk (2022)

[34] Eagen, L., Gabizon, A.: Protogalaxy: Efficient ProtoStar-style folding of multiple instances. Cryptology ePrint Archive, Paper 2023/1106 (2023)

[35] Eagen, L., Gabizon, A., Sefranek, M., Towa, P., Williamson, Z.J.: Stackproofs: Private proofs of stack and contract execution using protogalaxy. Cryptology ePrint Archive, Paper 2024/1281 (2024)

[36] Ferrara, A.L., Green, M., Hohenberger, S., Pedersen, M.Ø.: Practical short signature batch verification. In: Fischlin, M. (ed.) Topics in Cryptology – CT-RSA 2009. pp. 309–324. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

[37] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO. pp. 186–194 (1986)

[38] Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. ePrint Report 2019/953 (2019)

[39] Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: EUROCRYPT (2013)

[40] Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for zero-knowledge proof systems. Cryptology ePrint Archive, Paper 2019/458 (2019)

[41] Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT (2016)

[42] Haböck, U.: Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Paper 2022/1530 (2022)

[43] Jie, K.W.: A minimum-viable KZG polynomial commitment scheme implementation. https://ethresear.ch/t/a-minimum-viable-kzg-polynomial-commitment-scheme-implementation/7675 (July 2020)

[44] Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: ASIACRYPT. pp. 177–194 (2010)

[45] Khovratovich, D., Maller, M., Tiwari, P.R.: MinRoot: candidate sequential function for Ethereum VDF. Cryptology ePrint Archive, Paper 2022/1626 (2022)

[46] Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: STOC (1992)

[47] Kothapalli, A., Parno, B.: Algebraic reductions of knowledge. In: CRYPTO (2023)

[48] Kothapalli, A., Setty, S.: SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive (2022)

[49] Kothapalli, A., Setty, S.: Cyclefold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. Cryptology ePrint Archive, Paper 2023/1192 (2023), https://eprint.iacr.org/2023/1192, https://eprint.iacr.org/2023/1192

[50] Kothapalli, A., Setty, S.: HyperNova: Recursive arguments for customizable constraint systems. In: CRYPTO (2024)

[51] Kothapalli, A., Setty, S.: NeutronNova: Folding everything that reduces to zero-check. Cryptology ePrint Archive, Paper 2024/1606 (2024), https://eprint.iacr.org/2024/1606

[52] Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. In: CRYPTO (2022)

[53] Lee, J., Nikitin, K., Setty, S.: Replicated state machines without replicated execution. In: S&P (2020)

[54] Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. In: FOCS (Oct 1990)

[55] Micali, S.: CS proofs. In: FOCS (1994)

[56] Nguyen, W., Boneh, D., Setty, S.: Revisiting the Nova proof system on a cycle of curves. Cryptology ePrint Archive, Paper 2023/969 (2023)

[57] Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: TCC (2013)

[58] Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. J. ACM **27**(4) (1980)

[59] Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: CRYPTO (2020)

[60] Setty, S., Lee, J.: Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275 (2020)

[61] Setty, S., Thaler, J.: BabySpartan: Lasso-based SNARK for non-uniform computation. Cryptology ePrint Archive, Paper 2023/1799 (2023), https://eprint.iacr.org/2023/1799

[62] Setty, S., Thaler, J., Wahby, R.: Customizable constraint systems for succinct arguments. Cryptology ePrint Archive (2023)

[63] Setty, S., Thaler, J., Wahby, R.: Unlocking the lookup singularity with lasso. In: EUROCRYPT (2024)

[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., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: S&P (2018)

[69] Wesolowski, B.: Efficient verifiable delay functions. In: EUROCRYPT. pp. 379–407 (2019)

[70] 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)

Appendix A.
Additional background

A.1. Polynomials and multilinear extensions

We adapt this section from prior work [59]. We recall several definitions and results regarding multivariate polynomials.

Definition A.1 (Multilinear polynomial). A multivariate polynomial is called a multilinear polynomial if the degree of the polynomial in each variable is at most one.

Definition A.2 (Multilinear polynomial extension). Given a vector $v \in \mathbb{F}^n$ a *multilinear polynomial extension* of v is an $(\log n)$-variate multilinear polynomial, denoted $\widetilde{v}$, such that $\widetilde{v}(x) = v_x$ for all $x \in \{0,1\}^{\log n}$. Specifically, $\widetilde{v}$ can be computed as follows.

$$\widetilde{v}(x) = \sum_{y \in \{0,1\}^\ell} v_y \cdot \mathsf{eq}(x, y)$$

where $\mathsf{eq}(x, y) = \prod_{i=1}^{\ell}(x_i \cdot y_i + (1 - x_i) \cdot (1 - y_i))$, outputs 1 if $x = y$ and 0 otherwise for $x, y \in \{0,1\}^{\log n}$.

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

Lemma A.1 (Schwartz-Zippel [58])**.** *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.2. Commitment Schemes

Definition A.3 (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 $\mathsf{Com} : P \times M \times R \to C$ 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

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

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

Definition A.4 (Homomorphic). The commitment scheme $(\mathsf{Gen}, \mathsf{Com})$ 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 $r_1, r_2 \in R$

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

Definition A.5 (Succinct Commitments). A commitment scheme $(\mathsf{Gen}, \mathsf{Com})$, 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 $|\mathsf{Com}(\mathsf{pp}, m, r)| = O_\lambda(\mathsf{polylog}(|m|))$.

We modify the original definition to account for the preprocessed setting, in which a deterministic *encoder* algorithm preprocesses a portion of the statement called the structure (typically encoding a circuit or set of constraints) *once* and outputs a prover and verifier key which can be used to verify any number of witnesses (e.g., variable assignments) against this structure. We enable the encoder to additionally output a new structure to check the output instance-witness pair against (this can be preprocessed by a subsequent encoder).

Definition A.6 (Reduction of Knowledge [47]). Consider relations $\mathcal{R}_1$ and $\mathcal{R}_2$ over public parameters, structure, instance, and witness tuples. A reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$ is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic algorithm $\mathcal{K}$, called the generator, the prover, the verifier and the encoder respectively:

- $\mathcal{G}(\lambda, n) \to \mathsf{pp}$: Takes as input security parameter λ and size parameters n. Outputs public parameters pp.
- $\mathcal{K}(\mathsf{pp}, \mathsf{s}_1) \to (\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2)$: Takes as input public parameters pp and structure s_1. Outputs prover key pk, verifier key vk, and updated structure s_2.
- $\mathcal{P}(\mathsf{pk}, u_1, w_1) \to (u_2, w_2)$: Takes as input pk, and an instance-witness pair (u_1, w_1). Reduces the task of checking $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$ to checking $(\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}_2$.

- $\mathcal{V}(\mathsf{vk}, u_1) \to u_2$: Takes as input vk, and an instance u_1 in $\mathcal{R}_1$. Reduces the task of checking instance u_1 to the task of checking a new instance u_2 in $\mathcal{R}_2$.

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 $((\mathsf{pk}, \mathsf{vk}), u_1, w_1)$ and runs the interaction on the prover's input (pk, u_1, w_1) and the verifier's input (vk, u_1). At the end of the interaction, $\langle \mathcal{P}, \mathcal{V} \rangle$ outputs the verifier's instance u_2 and the prover's witness w_2. A reduction of knowledge $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ satisfies the following conditions.

- Completeness: For any PPT adversary $\mathcal{A}$, given $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n)$, $(\mathsf{s}_1, u_1, w_1) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$ and $(\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}_1)$ we have that the prover's output instance is equal to the verifier's output instance u_2, and that

$$(\mathsf{pp}, \mathsf{s}_2, \langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, w_1)) \in \mathcal{R}_2.$$

- 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 $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n)$, $(\mathsf{s}_1, u_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$, and $(\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}_1)$, we have that

$$\Pr[(\mathsf{pp}, \mathsf{s}_1, u_1, \mathcal{E}(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st})) \in \mathcal{R}_1] \approx \Pr[(\mathsf{pp}, \mathsf{s}_2, \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, \mathsf{st})) \in \mathcal{R}_2].$$

- Public reducibility: There exists a deterministic polynomial-time function φ such that for any PPT adversary $\mathcal{A}$ and expected polynomial-time adversary $\mathcal{P}^*$, given

$$\begin{aligned} \mathsf{pp} &\leftarrow \mathcal{G}(\lambda, n), \\ (\mathsf{s}_1, u_1, \mathsf{st}) &\leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2) &\leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}_1), \end{aligned}$$

and $(u_2, w_2) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, \mathsf{st})$ with the interaction transcript tr, we have that $\varphi(\mathsf{pp}, \mathsf{s}_1, u_1, \mathsf{tr}) = u_2$.

We can define various additional properties for RoKs such as succinctness (the communication is sublinear in the witness size), non-interactivity (the interaction is a single message from the prover), public-coin (the verifier only sends random challenges), and tree-extractability (there exists an extractor that can produce a satisfying witness given a tree of accepting transcripts). We define these properties in Appendix A.3.

Typically, we are interested in reducing several relations at once. We can interpret several relations as a single relation using the following product operator.

Definition A.7 (Relation product). For relations $\mathcal{R}_1$ and $\mathcal{R}_2$ over public parameter, structure, instance, and witness pairs we define the relation product as follows.

$$\mathcal{R}_1 \times \mathcal{R}_2 = \left\{ \ (\mathsf{pp}, \mathsf{s}, (u_1, u_2), (w_1, w_2)) \ \middle| \ (\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1, (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}_2 \ \right\}.$$

We let $\mathcal{R}^n$ denote $\mathcal{R} \times \ldots \times \mathcal{R}$ for n times.

A motivating property of RoKs is that they are composable, allowing us to build complex reductions by stitching together simpler ones. In particular, given reductions $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$ and $\Pi_2 : \mathcal{R}_2 \to \mathcal{R}_3$ we have that $\Pi_2 \circ \Pi_1$ (that is, running Π_1 first and then running Π_2 on the outputs) is a reduction of knowledge from R_1 to $\mathcal{R}_3$. Similarly, given reductions $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$ and $\Pi_2 : \mathcal{R}_3 \to \mathcal{R}_4$ we have that $\Pi_1 \times \Pi_2$ (that is, independently running Π_1 and Π_2 on pairs of inputs) is a reduction of knowledge from $\mathcal{R}_1 \times \mathcal{R}_3$ to $\mathcal{R}_2 \times \mathcal{R}_4$.

A.3. Reductions of Knowledge

First, we define additional properties of reductions of knowledge.

Definition A.8 (Succinctness). A reduction 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 A.9 (Non-interactivity). A reduction 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 A.10 (Public-coin). A reduction of knowledge is public-coin if the verifier only sends uniformly random challenges during the interaction.

Next, we define the semantics of the sequential and parallel composition operators.

Lemma A.2 (Sequential composition [47])**.** *For reductions* $\Pi_1 = (\mathcal{G}, \mathcal{K}_1, \mathcal{P}_1, \mathcal{V}_1) : \mathcal{R}_1 \to \mathcal{R}_2$ *and* $\Pi_2 = (\mathcal{G}, \mathcal{K}_2, \mathcal{P}_2, \mathcal{V}_2) : \mathcal{R}_2 \to \mathcal{R}_3$*, we have that* $\Pi_2 \circ \Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}) : \mathcal{R}_1 \to \mathcal{R}_3$ *where* $\mathcal{K}(\mathsf{pp}, \mathsf{s}_1)$ *computes* $(\mathsf{pk}_1, \mathsf{vk}_1, \mathsf{s}_2) \leftarrow \mathcal{K}_1(\mathsf{pp}, \mathsf{s}_1)$*,* $(\mathsf{pk}_2, \mathsf{vk}_2, \mathsf{s}_3) \leftarrow \mathcal{K}_2(\mathsf{pp}, \mathsf{s}_2)$ *and outputs* $((\mathsf{pk}_1, \mathsf{pk}_2), (\mathsf{vk}_1, \mathsf{vk}_2), \mathsf{s}_3)$ *and where*

$$\mathcal{P}((\mathsf{pk}_1, \mathsf{pk}_2), u_1, w_1) = \mathcal{P}_2(\mathsf{pk}_2, \mathcal{P}_1(\mathsf{pk}_1, u_1, w_1))$$
$$\mathcal{V}((\mathsf{vk}_1, \mathsf{vk}_2), u_1) = \mathcal{V}_2(\mathsf{vk}_2, \mathcal{V}_1(\mathsf{vk}_1, u_1, w_1))$$

Lemma A.3 (Parallel composition [47])**.** *Consider relations* $\mathcal{R}_1$*,* $\mathcal{R}_2$*,* $\mathcal{R}_3$*, and* $\mathcal{R}_4$*. For reductions of knowledge* $\Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_1, \mathcal{V}_1) : \mathcal{R}_1 \to \mathcal{R}_2$ *and* $\Pi_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_2, \mathcal{V}_2) : \mathcal{R}_3 \to \mathcal{R}_4$ *we have that* $\Pi_1 \times \Pi_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}) : \mathcal{R}_1 \times \mathcal{R}_3 \to \mathcal{R}_2 \times \mathcal{R}_4$ *where*

$$\mathcal{P}(\mathsf{pk}, (u_1, u_3), (w_1, w_3)) = (\mathcal{P}_1(\mathsf{pk}, u_1, w_1), \mathcal{P}_2(\mathsf{pk}, u_3, w_3))$$
$$\mathcal{V}(\mathsf{vk}, (u_1, u_3)) = (\mathcal{V}_1(\mathsf{vk}, u_1), \mathcal{V}_2(\mathsf{vk}, u_3))$$

Next, we can define arguments of knowledge as a special type of reduction of knowledge.

Definition A.11 (Argument of Knowledge). Consider the boolean relation $\mathsf{TRUE} = \{(\mathsf{true}, \bot)\}$. A proof of knowledge for relation $\mathcal{R}$ is a reduction of knowledge of type $\mathcal{R} \to \mathsf{TRUE}$.

When proving the security of protocols, reasoning about knowledge soundness directly is typically cumbersome. To alleviate this issue, Bootle et al. [20] show that to prove knowledge soundness for the vast majority of public-coin interactions, it is sufficient to show that there exists an extractor that can produce a satisfying witness when provided a tree of accepting transcripts with refreshed verifier randomness at each layer. This property is known as *tree-extractability* and we formally state the corresponding result for reductions of knowledge (proven by Kothapalli and Parno [47]).

Definition A.12 (Tree of transcripts). Consider an m-round public-coin interactive protocol $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ that satisfies the interface described in Definition A.6. For a public parameter, structure, instance tuple $(\mathsf{pp}, \mathsf{s}, u_1)$, a $(n_1, \ldots, n_m)$-tree of accepting transcripts is a tree of depth m where each vertex at layer i has n_i outgoing edges such that (1) each vertex in layer $i \in [m]$ is labeled with a prover message for round i; (2) each outgoing edge from layer $i \in [m]$ is labeled with a different choice of verifier randomness for round i; (3) each leaf is labeled with an accepting statement-witness pair output by the prover and verifier corresponding to the interaction along the path.

Lemma A.4 (Tree extraction [47])**.** *Consider an* m*-round public-coin interactive protocol* $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ *that satisfies the interface described in Definition A.6 and satisfies completeness. Then* $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ *is a reduction of knowledge if there exists a PPT extractor* χ *that outputs a satisfying witness* w_1 *with probability* $1 - \mathsf{negl}(\lambda)$*, given an* $(n_1, \ldots, n_m)$*-tree of accepting transcripts for public parameter, structure, instance tuple* $(\mathsf{pp}, \mathsf{s}, u_1)$ *where the verifier's randomness is sampled from space* Q *such that* $|Q| = O(2^\lambda)$*, and* $\prod_i n_i = \mathsf{poly}(\lambda)$*.*

A.4. Incrementally Verifiable Computation

Recall that incrementally verifiable computation (IVC), enables a prover to to produce a proof π_{i+1} for $i+1$ steps of a computation given a proof π_i for i steps of the computation in a way that ensures that the proof size remains independent of the total steps of computation. As IVC is a major application of folding schemes [25], [50], [52], for completeness, we recall the formal definition below.

Definition A.13 (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 $\mathcal{A}$

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

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

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

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.

Appendix B. Details on polyeval with matrix commitment

Suppose that we have witness vectors (Q_1, Q_2) for instances in MPE^2 i.e., each has n field elements of E_2. Suppose that their commitments are q_1 and q_2 respectively, each containing n_1 curve points on E_2.

To evaluate both Q_1 and Q_2 at the same point x (i.e., $x = x_1 = x_2$), the verifier produces a challenge c (e.g., using a transcript in the non-interactive version), and combine two matrices into one by $Q \leftarrow Q_1 + c \cdot Q_2$. We split x with $\ell = \log n$ scalars of E_2 into $(r_1, r_2) \leftarrow x$, where r_1 gets the first $\log n_1$ scalars and r_2 gets the rest $\log n_2$ scalars (both n_1 and n_2 are powers of 2 in our context). Then, we fold n_1 rows of Q into one row—a vector Q' of n_2 scalars —by taking a weighted sum, where the weight of i-th row of Q equals to $\mathsf{eq}(r_1, i)$. This effectively evaluates the polynomial using the tensor structure of multilinear polynomial evaluations [68].

We now describe a more refined polyeval that checks the purported instance-witness pairs of MPE^2, and output the instance. All arguments are taken as non-deterministic advice, and hash is a hash function (e.g., Poseidon).

$\underline{\mathsf{polyeval}(\mathsf{pp}, q_1, q_2, c, x, Q') \to (d, c, x, y)}$

(1) parse pp as (n_1, n_2, ck), where ck is the Pedersen's commitment key.
(2) compute $o_1 \leftarrow \mathsf{Com}(\mathsf{ck}, Q')$.
(3) parse x as (r_1, r_2), which is described above.
(4) compute $T \leftarrow (\mathsf{eq}(r_1, 0), \ldots, \mathsf{eq}(r_1, n_1 - 1))$,
(5) compute $q \leftarrow q_1 + c \cdot q_2$,
(6) compute $o_2 \leftarrow \mathsf{Com}(q, T)$ // an MSM.
(7) check that $o_1 = o_2$.
(8) compute $y \leftarrow \widetilde{Q'}(r_2)$, which is the evaluation.
(9) compute $d \leftarrow \mathsf{hash}((q_1, q_2))$.
(10) output (d, c, x, y).

As before, polyeval can be efficiently represented by an R1CS circuit over the base field of E_2 (i.e., the scalar field of E_1). The circuit hardcodes the public parameters pp and places the instances checked in the public IO.

Appendix C.
Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

C.1. Summary

MicroNova improves on prior designs and implementations of incrementally verifiable computation. The final proof output after any number of iterations of a function F is of size $O(\log n)$ group elements and can be verified with $O(\log n)$ group scalar multiplications and two pairing operations where n is the constraint complexity of F.

There is a well-known approach to compressing IVC proofs using SNARKs (proof composition). MicroNova is an optimized instantiation of this general idea for Nova. Previously, Nova used Spartan with IPA-based polynomial commitment scheme to compress the final IVC proof. The time to verify is $O(n)$. This paper both modifies Nova and builds a verifier-efficient version of Spartan using as its polynomial commitment scheme a new KZG variant for multilinear polynomials. It then uses this in place of Spartan to compress Nova IVC proofs. It demonstrates that verification can reasonably be executed via a smart contract on Ethereum.

C.2. Scientific Contributions

- provides a valuable step forward in an established field

C.3. Reasons for Acceptance

The paper is well written, has a thorough experimental section, achieves practical results, and presents a number of techniques that may be of independent interest.