Nova: Recursive Zero-Knowledge Arguments from Folding Schemes

Abhiram Kothapalli[†] Srinath Setty[⋆] Ioanna Tzialla[‡]

[†]Carnegie Mellon University [⋆]Microsoft Research [‡]New York University

Abstract. We introduce a new approach to realize incrementally verifiable computation (IVC), in which the prover recursively proves the correct execution of incremental computations of the form $y = F^{(\ell)}(x)$, where F is a (potentially non-deterministic) computation, x is the input, y is the output, and $\ell > 0$. Unlike prior approaches to realize IVC, our approach avoids succinct non-interactive arguments of knowledge (SNARKs) entirely and arguments of knowledge in general. Instead, we introduce and employ *folding schemes*, a weaker, simpler, and more efficiently-realizable primitive, which reduces the task of checking two instances in some relation to the task of checking a single instance. We construct a folding scheme for a characterization of NP and show that it implies an IVC scheme with improved efficiency characteristics: (1) the "recursion overhead" (i.e., the number of steps that the prover proves in addition to proving the execution of F) is a constant and it is dominated by two group scalar multiplications expressed as a circuit (this is the smallest recursion overhead in the literature), and (2) the prover's work at each step is dominated by two multiexponentiations of size $O(|F|)$, providing the fastest prover in the literature. The size of a proof is $O(|F|)$ group elements, but we show that using a variant of an existing zkSNARK, the prover can prove the knowledge of a valid proof succinctly and in zero-knowledge with $O(\log |F|)$ group elements. Finally, our approach neither requires a trusted setup nor FFTs, so it can be instantiated efficiently with any cycles of elliptic curves where DLOG is hard.

1 Introduction

We revisit the problem of realizing *incrementally-verifiable computation (IVC)* [43]: a cryptographic primitive that enables producing proofs of correct execution of "long running" computations such that a verifier can efficiently verify the correct execution of any prefix of the computation. IVC enables a wide variety of applications including verifiable delay functions [9,45], succinct blockchains [13,31], and incrementally-verifiable versions of verifiable state machines [33,39].

A well-known approach to construct IVC is to use succinct non-interactive arguments of knowledge (SNARKs) for NP [23,24,29,36]: at each incremental step i, the prover produces a SNARK proving that it has applied F correctly to the output of step $i-1$ and that the SNARK verifier *represented as a circuit* has accepted the SNARK from step $i-1$ [6,8]. However, it is well-known that this approach is impractical [6, 19]. Alternatively, one can use SNARKs without trusted setup [17, 21, 38, 41] but their verifiers are more expensive than those of SNARKs with trusted setup, both asymptotically and concretely. Recent works [10, 12, 15, 16] aim to address the inefficiency of SNARK-based IVC, with an innovative approach: at each step, the verifier circuit "defers" expensive steps in verifying a SNARK for NP instances (e.g., verifying polynomial evaluation proofs) by accumulating those steps into a single instance that is later checked efficiently. However, these works still require the prover to produce a SNARK at each step and the verifier circuit to partially verify that SNARK.

We introduce a new approach that avoids SNARKs (and more generally arguments of knowledge) entirely and relies purely on deferral to realize IVC. In a nutshell, instead of accumulating expensive steps of verifying a SNARK for NP instances, the verifier circuit in our approach accumulates the NP instances themselves. We formalize this technique as a new and minimal primitive, which we refer to as a *folding scheme*. A folding scheme is weaker, simpler, and far more efficient compared to arguments of knowledge including SNARKs. Indeed, realizing IVC via folding schemes results in improved efficiency over prior work (Figure 3): (1) the verifier circuit is constant-sized and its size is dominated by two group scalar multiplications; this is the smallest verifier circuit in the literature (in the context of recursive proof composition); and (2) the prover's work at each step is dominated by two multiexponentiations of size $O(|F|)$, providing the fastest prover in the literature, both asymptotically and concretely. Section 1.4 provides a detailed comparison between our approach and prior work.

1.1 Folding Schemes

A folding scheme is defined with respect to an NP relation, and it is a protocol between an untrusted *prover* and a *verifier*. Both entities hold two N-sized NP instances, and the prover in addition holds purported witnesses for both instances. The protocol enables the prover and the verifier to output a single N-sized NP instance, which we refer to as a *folded instance*. Furthermore, the prover privately outputs a purported witness to the folded instance using purported witnesses for the original instances. Informally, a folding scheme guarantees that the folded instance is satisfiable only if the original instances are satisfiable. A folding scheme is said to be *non-trivial* if the verifier's costs and the communication are lower in the case where the verifier participates in the folding scheme and then verifies a purported NP witness for the folded instance than the case where the verifier verifies purported NP witnesses for each of the original instances.

Several existing techniques exhibit the two-to-one reduction pattern of folding schemes. Examples include the sumcheck protocol [35] and the split-and-fold techniques in inner product arguments [11]. [30, App. A] provides further details.

Remark 1 (Folding Schemes vs. SNARKs). SNARKs for NP [7, 23, 24, 29, 36] trivially imply a folding scheme for NP: given two NP instances u_1 and u_2 and the corresponding witnesses, the prover proves u_1 by producing a SNARK. The verifier checks that SNARK and then sets u_2 to be the folded instance. However, we construct a folding scheme for NP without relying on SNARKs (or more generally arguments of knowledge). Specifically, our folding scheme is weaker than any argument of knowledge (succinct or otherwise) because it merely *reduces* the satisfiability of two NP instances to the satisfiability of a single NP instance.[1]

To design a folding scheme for NP, we start with a popular NP-complete language that generalizes arithmetic circuit satisfiability: R1CS (Definition 10). As we illustrate later, it is difficult to devise a folding scheme for R1CS. To address this, we introduce a variant of R1CS, called *relaxed R1CS*, which, like R1CS, not only characterizes NP, but, unlike R1CS, can support a folding scheme. The following theorem captures the cryptographic and efficiency characteristics of our folding scheme for relaxed R1CS.

Theorem 1. *There exists a constant-round, public-coin, zero-knowledge folding scheme for relaxed R1CS where for N-sized relaxed R1CS instances over a finite field $\mathbb{F}$ with the same "structure" (i.e., R1CS coefficient matrices), the prover's work is $O_\lambda(N)$, and the verifier's work and the communication are both $O_\lambda(1)$, assuming the existence of any additively-homomorphic commitment scheme that provides $O_\lambda(1)$-sized commitments to N-sized vectors over $\mathbb{F}$ (e.g., Pedersen's commitments), where λ is the security parameter.*

Because our folding scheme is public coin, it can be made non-interactive in the random oracle model using the Fiat-Shamir transform [22], and be instantiated (heuristically) in the standard model using a concrete hash function. We rely on such a non-interactive folding scheme to construct IVC.

1.2 IVC from Non-Interactive Folding Schemes

We show how to realize IVC using a non-interactive version of our folding scheme for relaxed R1CS. We refer to our construction as Nova.

Recall that an IVC is an argument of knowledge [29, 36][2] for incremental computations of the form $y = F^{(\ell)}(x)$, where F is a (possibly non-deterministic) computation, $\ell > 0$, x is a public input, and y is the public output. At each incremental step, the IVC prover produces a proof that the step was computed correctly *and* it has verified a proof for the prior step. In other words, at each incremental step, the IVC prover produces a proof of satisfiability for an augmented circuit that augments the circuit for F with a "verifier circuit" that verifies the proof of the prior step. Recursively, the final proof proves the correctness of the entire incremental computation. A key aspect of IVC is that neither the IVC verifier's work nor the IVC proof size depends on the number of

[1] This work realizes IVC using our folding scheme. As IVC implies SNARKs (e.g., see [6]), one might wonder whether folding schemes are in general weaker than SNARKs. However, existing constructions of IVC (including our own) rely on additional assumptions (§4.2), which the resulting IVC-based SNARK inherits.

[2] An *argument of knowledge* for circuit satisfiability enables an untrusted polynomial-time prover to prove to a verifier the knowledge of a witness w such that $\mathcal{C}(w, x) = y$, where $\mathcal{C}$ is a circuit, x is some public input, and y is some public output.

steps in the incremental computation. In particular, the IVC verifier only verifies the proof produced at the last step of the incremental computation.

In Nova, we consider incremental computations, where each step of the incremental computation is expressed with R1CS (all the steps in the incremental computation share the same R1CS coefficient matrices). At step i of the incremental computation, as in other approaches to IVC, Nova's prover proves that the step i was computed correctly. Furthermore, at step i, instead of verifying a proof for step $i-1$ (as in traditional approaches to IVC), Nova's approach treats the computation at step $i-1$ as an R1CS instance and folds that into a running relaxed R1CS instance. Specifically, at each step, Nova's prover proves that it has performed the step's computation and has folded its prior step represented as an R1CS instance into a running relaxed R1CS instance. In other words, the circuit satisfiability instance that the prover proves at each incremental step computes a step of the incremental computation and includes a circuit for the computation of the verifier in the non-interactive folding scheme for relaxed R1CS.

A distinctive aspect of Nova's approach to IVC is that it achieves the smallest "verifier circuit" in the literature. Since the verifier's costs in the non-interactive version of the folding scheme for relaxed R1CS is $O_\lambda(1)$, the size of the computation that Nova's prover proves at each incremental step is $\approx|F|$, assuming N-sized vectors are committed with an $O_\lambda(1)$-sized commitments (e.g., Pedersen's commitments). In particular, the verifier circuit in Nova is constant-sized and its size is dominated by two *group scalar multiplications*. Furthermore, Nova's prover's work at each step is dominated by two multiexponentiations of size $\approx|F|$. Note that Nova's prover does not perform any FFTs, so it can be instantiated efficiently using *any* cycles of elliptic curves where DLOG is hard.

With the description thus far, the size of an IVC proof (which is a purported witness for the running relaxed R1CS instance) is $O_\lambda(|F|)$. Instead of sending such a proof to a verifier, at any point in the incremental computation, Nova's prover can prove the knowledge of a satisfying witness to the running relaxed R1CS instance in zero-knowledge with an $O_\lambda(\log|F|)$-sized succinct proof using a zkSNARK that we design by adapting Spartan [38]. The following theorem summarizes our key result.

Theorem 2. *For any incremental function where each step of the incremental function applies a (non-deterministic) function F, there exists an IVC scheme with the following efficiency characteristics, assuming N-sized vectors are committed with an $O_\lambda(1)$-sized commitments.*

- *IVC proof sizes are $O(|F|)$ and the verifier's work to verify them is $O_\lambda(|F|)$. The prover's work at each incremental step is $\approx|F|$. Specifically, the prover's work at each step is dominated by two multiexponentiations of size $\approx|F|$.*
- *Succinct zero-knowledge proofs of valid IVC proofs are size $O_\lambda(\log|F|)$, and the verifier's work to verify them is either $O_\lambda(\log|F|)$ or $O_\lambda(|F|)$ depending on the commitment scheme for vectors. The prover's work to produce this succinct zero-knowledge proof is $O_\lambda(|F|)$.*

1.3 Implementation and Performance Evaluation

We implement Nova as a library in about 6,000 lines of Rust [3]. The library is generic over a cycle of elliptic curves and a hash function (used internally as the random oracle). The library provides candidate implementations with the Pasta cycle of elliptic curves [4] and Poseidon [2,26]. Finally, the library accepts F (i.e., a step of the incremental computation) as a bellperson gadget [1].

Recursion Overheads. We measure the size of Nova's verifier circuit, as it determines the *recursion overhead*: the number of additional constraints that the prover must prove at each incremental step besides proving an invocation of F.

We find that Nova's verifier circuit is ≈20,000 R1CS constraints. This is the smallest verifier circuit in the literature and hence Nova incurs the lowest recursion overhead. Specifically, Nova's recursion overhead is $> 10\times$ lower than in SNARK-based IVC [6] with state-of-the-art per-circuit trusted setup SNARK [27], and over $100\times$ smaller than with a SNARK without trusted setup [21]. Compared to recent works, Nova's recursion overhead is over $7\times$ lower than Halo's [12], and over $2\times$ lower than the scheme of Bunz et al. [15].

	Primary Curve	Secondary Curve
Scalar multiplications	12,362	12,362
Random oracle call	1,431	1,434
Collision-resistant hash	2,300	2,306
Non-native arithmetic	3,240	3,240
Glue code	1,251	1,782
Total	20,584	21,124

Fig. 1: A detailed breakdown of sub-routines in Nova's verifier's circuit and the associated number of R1CS constraints. The verifier circuit on each of the curves in the cycle are not identical as they have slightly different base cases. We find that a majority of constraints in the verifier circuit step from the group scalar multiplications.

Performance of Nova. We experiment with Nova on an Azure Standard F32s_v2 VM (16 physical CPUs, 2.70 GHz Intel(R) Xeon(R) Platinum 8168, and 64 GB memory). In our experiments, we vary the number of constraints in F. Our performance metrics are: the prover time, the verifier time, and proof sizes. We measure these for Nova's IVC scheme as well as its Spartan-based zkSNARK to compress IVC proofs. Figure 2 depicts our results, and we find the following.

– The prover's per-step cost to produce an IVC proof and compress it scale sub-linearly with the size of F (since the cost is dominated by two multiexponentiations, which scale sub-linearly due to the Pippenger algorithm and parallelize better at larger sizes). When $|F| \approx 2^{20}$ constraints, the prover's per-step cost to produce an IVC proof is $\approx 1\mu$s/constraint. For the same F, the cost to produce a compressed IVC proof is $\approx 24\mu$s/constraint.[3]

[3] If the prover produces a compressed IVC proof every ≈24 steps, the prover incurs at most $2\times$ overhead to compress IVC proofs. Similarly, if the prover compresses its IVC proof every ≈240 steps, the overhead drops to ≈20%.

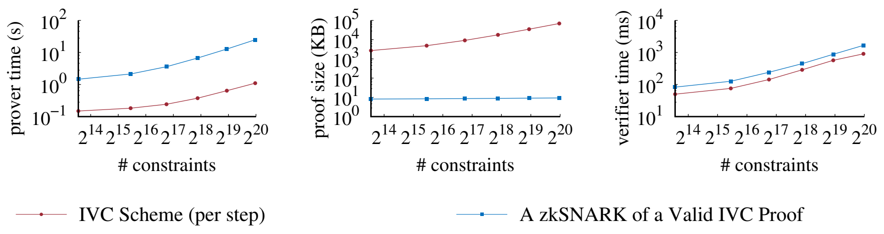


Fig. 2: Performance of Nova as a function of $|F|$. See the text for details.

- Compressed IVC proofs are $\approx$ 8–9 KB and are significantly shorter than IVC proofs (e.g., they are $\approx$7,400$\times$ shorter when $|F| \approx 2^{20}$ constraints).
- Verifying a compressed proof is only $\approx$2$\times$ higher costs than verifying a significantly longer IVC proof.

1.4 A More Detailed Comparison with Prior Work

Figure 3 compares Nova with prior approaches. Nova's approach can be viewed as taking Halo's approach to the extreme. Specifically:

- At each incremental step, Halo's verifier circuit verifies a "partial" SNARK. This still requires Halo's prover to perform $|F|$-sized FFTs and $O(|F|)$ exponentiations (i.e., *not* an $|F|$-sized multiexponentiation). Whereas, in Nova, the verifier circuit folds an entire NP instance representing computation at the prior step into a running relaxed R1CS instance. This only requires Nova's prover to commit to a satisfying assignment of an $\approx$$|F|$-sized circuit (which computes F and performs the verifier's computation in a folding scheme for relaxed R1CS), so at each step, Nova's prover only computes an $O(|F|)$-sized multiexponentiation and does not compute any FFTs. So, Nova's prover incurs lower costs than Halo's prover, both asymptotically and concretely.
- The verifier circuit in Halo is of size $O_\lambda(\log|F|)$ whereas in Nova, it is $O_\lambda(1)$. Concretely, the dominant operations in Halo's circuit is $O(\log|F|)$ group scalar multiplications, whereas in Nova, it is two group scalar multiplications.
- Halo and Nova have the same proof sizes $O_\lambda(\log|F|)$ and verifier time $O_\lambda(|F|)$.

Bünz et al. [16] apply Halo's approach to other polynomial commitment schemes. Halo Infinite [10] generalizes the approach in Halo [12] to any homomorphic polynomial commitment scheme; they also obtain PCD (and hence IVC) even when polynomial commitment schemes do not satisfy succinctness.

Bünz et al. [15] propose a variant of the approach in Halo, where they realize PCD (and hence IVC) without relying on succinct arguments. Specifically, they first devise a non-interactive argument of knowledge (NARK) for R1CS with $O_\lambda(N)$-sized proofs and $O_\lambda(N)$ verification times for N-sized R1CS instances. Then, they show that most of the NARK's verifier's computation can be deferred by performing $O_\lambda(1)$ work in the verifier circuit. For zero-knowledge, Nova relies on zero-knowledge arguments with succinct proofs, whereas their approach does

	"Verifier circuit" (dominant ops)	Prover (each step)	Proof size	Verifier	assumptions
BCTV14 [6] with [27][†]	3 $\mathbb{P}$	$O(C)$ FFT $O(C)$ MSM	$O_\lambda(1)$	$O_\lambda(1)$	q-type
Spartan [38]-based IVC	$O(\sqrt{C})$ $\mathbb{G}$	$O(C)$ MSM	$O_\lambda(\sqrt{C})$	$O_\lambda(\sqrt{C})$	DLOG, RO
Fractal [21]	$O_\lambda(\log^2 C)$ $\mathbb{F}$ $O(\log^2 C)$ $\mathbb{H}$	$O(C)$ FFT $O(C)$ MHT	$O_\lambda(\log^2 C)$	$O_\lambda(\log^2 C)$	RO
Halo [12]	$O(\log C)$ $\mathbb{G}$	$O(C)$ FFT $O(C)$ EXP	$O_\lambda(\log C)$	$O_\lambda(C)$	DLOG, RO
BCLMS [15][⋆]	8 $\mathbb{G}$	$O(C)$ FFT $O(C)$ MSM	$O_\lambda(C)$	$O_\lambda(C)$	DLOG, RO
Nova (this work)	2 $\mathbb{G}$	$O(C)$ MSM	$O_\lambda(\log C)$	$O_\lambda(C)$	DLOG, RO
Nova (this work)	2 $\mathbb{G}_T$	$O(C)$ MSM	$O_\lambda(\log C)$	$O_\lambda(\log C)$	SXDH, RO

[†] Requires per-circuit trusted setup and is undesirable in practice
$O(C)$ FFT: FFT over an $O(C)$-sized vector costing $O(C \log C)$ operations over $\mathbb{F}$
$O(C)$ MHT: Merkle tree over an $O(C)$-sized vector costing $O(C)$ hash computations
$O(C)$ EXP: $O(C)$ exponentiations in a cryptographic group
$O(C)$ MSM: $O(C)$-sized multi-exponentiation in a cryptographic group

Fig. 3: Asymptotic costs of Nova and its baselines to produce and verify a proof for an incremental computation where each incremental step applies a function F. C denotes the size of the computation at each incremental step, i.e., $|F| + |\mathcal{C}_\mathcal{V}|$, where $\mathcal{C}_\mathcal{V}$ is the "verifier circuit" in IVC. The "verifier circuit" column depicts the number of dominant operations in $\mathcal{C}_\mathcal{V}$, where $\mathbb{P}$ denotes a pairing in a pairing-friendly group, $\mathbb{F}$ denotes the number of finite field operations, $\mathbb{H}$ denotes a hash computation, and $\mathbb{G}$ denotes a scalar multiplication in a cryptographic group. The prover column depicts the cost to the prover for each step of the incremental computation, and proof sizes and verifier times refer respectively to the size of the proof of the incremental computation and the associated verification times. For Nova's proof sizes and verification times, we depict the compressed proof sizes (otherwise, they are $O_\lambda(C)$) and the time to verify a compressed proof (otherwise, they are $O_\lambda(C)$). Rows with RO require heuristically instantiating the random oracle with a concrete hash function in the standard model.

not rely on succinct arguments. However, Nova's approach has several conceptual and efficiency advantages over the work of Bünz et al [15]:

- Nova introduces a new primitive called a folding scheme, which is conceptually simpler and is easier to realize than prior notions such as (split) accumulation schemes used in prior work [15, 16]. Furthermore, a folding scheme for NP directly leads to IVC and is again easier to analyze than with prior notions.
- At each step, their prover performs an $O(|F|)$-sized FFT (which costs $O(|F| \log |F|)$ operations over $\mathbb{F}$). Whereas, Nova does not perform any FFTs.
- Their prover's work for multiexponentitions at each step and the size of their verifier circuit are both higher than in Nova by $\approx 4\times$.
- Proof sizes are $O_\lambda(|F|)$ in their work, whereas in Nova, they are $O_\lambda(\log |F|)$. We believe, in theory, they can also compress their proofs, using a succinct argument, but unlike Nova, they do not specify how to do so in a concretely efficient manner. Furthermore, using succinct arguments is inconsistent with their goal of not employing them.

Concurrent work. In an update concurrent with this work, Bünz et al. [15] provide an improved construction of their NARK for R1CS, which leads to an IVC that, like Nova, avoids FFTs. Furthermore, they improve the size of the verifier circuit by $\approx 2\times$, which is still larger than Nova's verifier circuit by $\approx 2\times$. The per-step computation of the prover remains $4\times$ higher than Nova.

1.5 An Overview of the Rest of the Paper

Section 2 provides the necessary background. Section 3 formally defines folding schemes and their properties. In Section 4, we introduce a variant of R1CS called relaxed R1CS for which we provide a folding scheme satisfying Theorem 1. Then, in Section 5, we use a non-interactive version of the folding scheme (§4.2) to construct an IVC scheme and a scheme to compress IVC proofs satisfying Theorem 2 by assuming the existence of a zkSNARK for relaxed R1CS with logarithmic-sized proofs. Finally, in Section 6, we construct such a zkSNARK.

2 Preliminaries

Let $\mathbb{F}$ denote a finite field with $|\mathbb{F}| = 2^{\Theta(\lambda)}$, where λ is the security parameter. Let $\cong$ denote computational indistinguishability with respect to a PPT adversary. We globally assume that generator algorithms that produce public parameters are additionally provided appropriate size bounds.

2.1 A Commitment Scheme for Vectors over $\mathbb{F}$

We require a commitment scheme for vectors over $\mathbb{F}$ that is additively homomorphic and succinct. We formally define these two properties and others noted below in [30, App. F]. Below, we define the syntax for commitment schemes.

Definition 1 (A Commitment Scheme for Vectors). *A commitment scheme for $\mathbb{F}^m$ is a tuple of three protocols with the following interface.*

- $\mathsf{Gen}(1^\lambda, m) \to \mathsf{pp}$*: takes length parameter m; produces public parameters* pp.
- $\mathsf{Com}(\mathsf{pp}, v, r) \to C$*: takes vector $v \in \mathbb{F}^m$ and $r \in \mathbb{F}$; produces commitment C.*
- $\mathsf{Open}(\mathsf{pp}, C, v, r) \to \{0, 1\}$*: verifies the opening of commitment C to $v \in \mathbb{F}^m$.*

A commitment scheme satisfies hiding (the commitment reveals no information), binding (a PPT adversary cannot open a commitment to two different values), and succinctness (the commitment size is logarithmic in the opening size).

2.2 Non-Interactive Arguments of Knowledge

Definition 2 (Non-Interactive Argument of Knowledge). *Consider a relation $\mathcal{R}$ over public parameters, structure, instance, and witness tuples. A non-interactive argument of knowledge for $\mathcal{R}$ consists of 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.*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$*: On input security parameter* λ*, samples public parameters* pp*.*
- $\mathcal{K}(\mathsf{pp}, \mathsf{s}) \to (\mathsf{pk}, \mathsf{vk})$*: On input structure* s*, representing common structure among instances, outputs the prover key* pk *and verifier key* vk*.*
- $\mathcal{P}(\mathsf{pk}, u, w) \to \pi$*: On input instance* u *and witness* w*, outputs a proof* π *proving that* $(\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}$*.*
- $\mathcal{V}(\mathsf{vk}, u, \pi) \to \{0, 1\}$*: Checks instance* u *given proof* π*.*

An argument of knowledge satisfies completeness if for any PPT adversary $\mathcal{A}$

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

An argument of knowledge satisfies knowledge soundness if for all PPT adversaries $\mathcal{A}$ *there exists a PPT extractor* $\mathcal{E}$ *such that for all randomness* ρ

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{vk}, u, \pi) = 1, \\ (\mathsf{pp}, \mathsf{s}, u, w) \notin \mathcal{R} \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, u, \pi) \leftarrow \mathcal{A}(\mathsf{pp}; \rho), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ w \leftarrow \mathcal{E}(\mathsf{pp}, \rho) \end{array}\right] = \mathsf{negl}(\lambda).$$

Definition 3 (Zero-Knowledge). *An argument of knowledge* $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ *for relation* $\mathcal{R}$ *satisfies zero-knowledge if there exists PPT simulator* $\mathcal{S}$ *such that for all PPT adversaries* $\mathcal{A}$

$$\left\{ (\mathsf{pp}, \mathsf{s}, u, \pi) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, (u, w)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ \pi \leftarrow \mathcal{P}(\mathsf{pk}, u, w) \end{array}\right\} \cong \left\{ (\mathsf{pp}, \mathsf{s}, u, \pi) \;\middle|\; \begin{array}{l} (\mathsf{pp}, \tau) \leftarrow \mathcal{S}(1^\lambda), \\ (\mathsf{s}, (u, w)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ \pi \leftarrow \mathcal{S}(\mathsf{pp}, u, \tau) \end{array}\right\}$$

Definition 4 (Succinctness). *A non-interactive argument system is succinct if the size of the proof* π *is polylogarithmic in the size of the witness* w*.*

2.3 Incrementally Verifiable Computation

Incrementally verifiable computation (IVC) [43] enables verifiable computation for repeated function application. Intuitively, for a function F, with initial input z_0, an IVC scheme allows a prover to produce a proof Π_i for the statement $z_i = F^{(i)}(z_0)$ (i.e., i applications of F on input z_0) given a proof Π_{i-1} for the statement $z_{i-1} = F^{(i-1)}(z_0)$. Formally, IVC schemes additionally permit F to take auxiliary input ω. We recall the definition of IVC using notational conventions of modern argument systems.

Definition 5 (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. An IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ satisfies perfect completeness if for any PPT adversary $\mathcal{A}$*

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

where F is a polynomial time computable function. Likewise, an IVC scheme satisfies knowledge-soundness if for any constant $n \in \mathbb{N}$, and expected polynomial time adversaries $\mathcal{P}^*$ there exists expected polynomial-time extractor $\mathcal{E}$ such that for any input randomness ρ

$$\Pr\left[\begin{array}{l} z_n \neq z, \\ \mathcal{V}(\mathsf{vk}, n, z_0, z, \Pi) = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ F, (z_0, z, \Pi) \leftarrow \mathcal{P}^*(\mathsf{pp}; \rho), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F), \\ (\omega_0, \ldots, \omega_{n-1}) \leftarrow \mathcal{E}(\mathsf{pp}, z_0, z; \rho), \\ z_i \leftarrow F(z_{i-1}, \omega_{i-1}) \quad \forall i \in \{1, \ldots, n\} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

An IVC scheme satisfies succinctness if the size of the IVC proof Π does not grow with the number of applications n.

We note that in the definition above, the number of steps n is treated as a fixed environment variable that characterizes the extractor. This model is required for all known general recursive techniques as they rely on recursive extractors that blowup polynomially for each additional recursive step [10, 13, 15, 16, 21]. Bitansky et al. [8] avoid such a restriction by making non-blackbox assumptions about the extractors runtime with respect to that of the malicious prover. In any case, there are no known attacks on arbitrary depth recursion.

3 Folding Schemes

This section formally defines folding schemes. Intuitively, a folding scheme for a relation $\mathcal{R}$ is a protocol that reduces the task of checking two instances in $\mathcal{R}$ to the task of checking a single instance in $\mathcal{R}$.

Definition 6 (Folding Scheme). *Consider a relation $\mathcal{R}$ over public parameters, structure, instance, and witness tuples. A folding scheme for $\mathcal{R}$ consists of a PPT generator algorithm $\mathcal{G}$, a deterministic encoder algorithm $\mathcal{K}$, and a pair of PPT algorithms $\mathcal{P}$ and $\mathcal{V}$ denoting the prover and verifier respectively, with the following interface:*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$*: On input security parameter λ, samples public parameters* pp.
- $\mathcal{K}(\mathsf{pp}, \mathsf{s}) \to (\mathsf{pk}, \mathsf{vk})$*: On input* pp*, and a common structure* s *between instances to be folded, outputs a prover key* pk *and a verifier key* vk.

– $\mathcal{P}(\mathsf{pk}, (u_1, w_1), (u_2, w_2)) \rightarrow (u, w)$: *On input instance-witness tuples* (u_1, w_1) *and* (u_2, w_2) *outputs a new instance-witness tuple* (u, w) *of the same size.*
– $\mathcal{V}(\mathsf{vk}, u_1, u_2) \rightarrow u$: *On input instances* u_1 *and* u_2*, outputs a new instance* u*.*

Let

$$(u, w) \leftarrow \langle \mathcal{P}(\mathsf{pk}, w_1, w_2), \mathcal{V}(\mathsf{vk})\rangle(u_1, u_2)$$

denote the the verifier's output instance u *and the prover's output witness* w *from the interaction of* $\mathcal{P}$ *and* $\mathcal{V}$ *on witnesses* (w_1, w_2)*, prover key* pk*, verifier key* vk *and instances* (u_1, u_2)*. Likewise, let*

$$\mathsf{tr} = \langle \mathcal{P}(\mathsf{pk}, w_1, w_2), \mathcal{V}(\mathsf{vk})\rangle(u_1, u_2)$$

denote the corresponding interaction transcript. A folding scheme satisfies perfect completeness if for all PPT adversaries $\mathcal{A}$

$$\Pr\left[(\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, (u_1, w_1), (u_2, w_2)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, \mathsf{s}, u_1, w_1), (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ (u, w) \leftarrow \langle \mathcal{P}(\mathsf{pk}, w_1, w_2), \mathcal{V}(\mathsf{vk})\rangle(u_1, u_2) \end{array}\right] = 1.$$

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

$$\Pr\left[\begin{array}{l} (\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}, \\ (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R} \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, (u_1, u_2)) \leftarrow \mathcal{P}^*(\mathsf{pp}, \rho), \\ (w_1, w_2) \leftarrow \mathcal{E}(\mathsf{pp}, \rho) \end{array}\right] \geq$$
$$\Pr\left[(\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, (u_1, u_2)) \leftarrow \mathcal{P}^*(\mathsf{pp}, \rho), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ (u, w) \leftarrow \langle \mathcal{P}^*(\mathsf{pk}, \rho), \mathcal{V}(\mathsf{vk})\rangle(u_1, u_2) \end{array}\right] - \mathsf{negl}(\lambda)$$

where ρ *denotes arbitrary input randomness for* $\mathcal{P}^*$*. We call a transcript an accepting transcript if* $\mathcal{P}$ *outputs a satisfying folded witness* w *for the folded instance* u*. We consider a folding scheme non-trivial if the communication costs and* $\mathcal{V}$*'s computation are lower in the case where* $\mathcal{V}$ *participates in the folding scheme and then checks a witness sent by* $\mathcal{P}$ *for the folded instance than the case where* $\mathcal{V}$ *checks witnesses sent by* $\mathcal{P}$ *for each of the original instances.*

Definition 7 (Non-Interactive). *A folding scheme* $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ *is non-interactive if the interaction between* $\mathcal{P}$ *and* $\mathcal{V}$ *consists of a single message from* $\mathcal{P}$ *to* $\mathcal{V}$*. This single message is denoted as an output of* $\mathcal{P}$*, and an input to* $\mathcal{V}$*.*

Definition 8 (Zero-Knowledge). *A folding scheme* $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ *satisfies zero-knowledge for relation* $\mathcal{R}$ *if there exists a PPT simulator* $\mathcal{S}$ *such that for all PPT* *adversaries $\mathcal{A}$, and $\mathcal{V}^*$, and input randomness ρ*

$$\left\{ \mathsf{tr} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, (u_1, w_1), (u_2, w_2)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ (\mathsf{pp}, \mathsf{s}, u_1, w_1), (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}, \\ \mathsf{tr} = \langle \mathcal{P}(\mathsf{pk}, w_1, w_2), \mathcal{V}^*(\mathsf{vk}, \rho) \rangle (u_1, u_2) \end{array} \right\} \cong$$

$$\left\{ \mathsf{tr} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, (u_1, w_1), (u_2, w_2)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, \mathsf{s}, u_1, w_1), (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ \mathsf{tr} \leftarrow \mathcal{S}^{\mathcal{V}^*(\mathsf{vk}, \rho)}(\mathsf{pk}, u_1, u_2) \end{array} \right\}$$

Definition 9 (Public Coin). *A folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is called public coin if all the messages sent from $\mathcal{V}$ to $\mathcal{P}$ are sampled from a uniform distribution.*

Typically, knowledge soundness is difficult to prove directly. To assist these proofs, prior works employ the forking lemma [11], which abstracts away much of the probabilistic reasoning. The original forking lemma shows that to prove knowledge soundness it is sufficient to construct a PPT extractor that takes as input a "tree" of accepting transcripts and outputs a satisfying witness. However, in our setting, this extractor must *additionally* take as input the prover's output (i.e., the folded instance and witness) for each of these transcripts, which contains information needed to reconstruct the original witness. So, we introduce a small variant of the forking lemma that captures this modification.

Lemma 1 (Forking Lemma for Folding Schemes). *Consider a $(2\mu + 1)$-move folding scheme $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$. Π satisfies knowledge soundness if there exists a PPT $\mathcal{X}$ such that for all input instance pairs u_1, u_2, outputs satisfying witnesses w_1, w_2 with probability $1 - \mathsf{negl}(\lambda)$, given public parameters pp, structure s, and an $(n_1, \ldots, n_\mu)$-tree of accepting transcripts and corresponding folded instance-witness pairs (u, w). This tree comprises n_1 transcripts (and corresponding instance-witness pairs) with fresh randomness in $\mathcal{V}$'s first message; and for each such transcript, n_2 transcripts (and corresponding instance-witness pairs) with fresh randomness in $\mathcal{V}$'s second message; etc., for a total of $\prod_{i=1}^{\mu} n_i$ leaves bounded by $\mathsf{poly}(\lambda)$.*

Proof Intuition. A proof for our variant of the forking lemma is similar to that of Bootle et al. [11]. We present a formal proof in [30, App. F]. □

4 A Folding Scheme for NP

In this section, we describe a public-coin, zero-knowledge interactive folding scheme for NP. We additionally discuss how to make it non-interactive. We leverage the non-interactivity property to realize IVC in the next section, and the zero-knowledge property to achieve zero-knowledge IVC proof compression.

4.1 A Public-Coin, Zero-Knowledge Folding Scheme

To design a folding scheme for NP, we need an NP-complete language. While theoretically any NP-complete language is a viable candidate, we focus on R1CS,[4] a popular algebraic representation that generalizes arithmetic circuit satisfiability.

Definition 10 (R1CS). *Consider a finite field $\mathbb{F}$. Let the public parameters consist of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$. The R1CS structure consists of sparse matrices $A, B, C \in \mathbb{F}^{m \times m}$ with at most $n = \Omega(m)$ non-zero entries in each matrix. An instance $\mathsf{x} \in \mathbb{F}^{\ell}$ consists of public inputs and outputs and is satisfied by a witness $W \in \mathbb{F}^{m-\ell-1}$ if $(A \cdot Z) \circ (B \cdot Z) = C \cdot Z$, where $Z = (W, \mathsf{x}, 1)$.*

As we show in the next section, to realize IVC, we only need a folding scheme that can fold two R1CS instances with the same R1CS matrices (A, B, C). Specifically, given R1CS matrices (A, B, C), and two corresponding instance-witness pairs (x_1, W_1) and (x_2, W_2), we would like to devise a scheme that reduces the task of checking both instances into the task of checking a single new instance-witness pair (x, W) against the same R1CS matrices (A, B, C). Unfortunately, as we illustrate now, it is difficult to devise a folding scheme for R1CS such that it satisfies completeness, let alone knowledge soundness.

First Attempt. As R1CS is an algebraic system, the most direct approach would be to take a random linear combination. Ignoring efficiency concerns, suppose that the prover sends witnesses W_1 and W_2 in the first step. The verifier responds with a random $r \in \mathbb{F}$; the prover and the verifier both compute

$$\begin{aligned} \mathsf{x} &\leftarrow \mathsf{x}_1 + r \cdot \mathsf{x}_2 \\ W &\leftarrow W_1 + r \cdot W_2, \end{aligned}$$

and set the new instance-witness pair to be (x, W). However, for non-trivial $Z_1 = (W_1, \mathsf{x}_1, 1)$ and $Z_2 = (W_2, \mathsf{x}_2, 1)$, and $Z = (W, \mathsf{x}, 1)$, we *roughly* have that

$$\begin{aligned} AZ \circ BZ &= A(Z_1 + r \cdot Z_2) \circ B(Z_1 + r \cdot Z_2) \\ &= AZ_1 \circ BZ_1 + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2 \cdot (AZ_2 \circ BZ_2) \\ &\neq CZ. \end{aligned}$$

The failed attempt exposes three issues. First, we must account for an additional cross-term, $r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1)$. Second, the terms excluding the cross-term combine to produce a term that does not equal CZ:

$$AZ_1 \circ BZ_1 + r^2 \cdot (AZ_2 \circ BZ_2) = CZ_1 + r^2 \cdot CZ_2 \neq CZ_1 + r \cdot CZ_2 = CZ.$$

Third, we do not even have that $Z = Z_1 + r \cdot Z_2$ because $Z_1 + r \cdot Z_2 = (W, \mathsf{x}, 1 + r \cdot 1)$.

[4] R1CS is implicit in the QAPs formalism of GGPR [23], but it was made explicit in subsequent work [40]; they refer to it as a "constraint system in quadratic form".

Second Attempt. To handle the first issue, we introduce a "slack" (or error) vector $E \in \mathbb{F}^m$ which absorbs the cross terms generated by folding. To handle the second and third issues, we introduce a scalar u, which absorbs an extra factor of r in $CZ_1 + r^2 \cdot CZ_2$ and in $Z = (W, \mathsf{x}, 1 + r \cdot 1)$. We refer to a variant of R1CS with these additional terms as *relaxed* R1CS.

Definition 11 (Relaxed R1CS). *Consider a finite field $\mathbb{F}$. Let the public parameters consist of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$. The 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 relaxed R1CS instance consists of an error vector $E \in \mathbb{F}^m$, a scalar $u \in \mathbb{F}$, and public inputs and outputs $\mathsf{x} \in \mathbb{F}^\ell$. An instance (E, u, x) is satisfied by a witness $W \in \mathbb{F}^{m-\ell-1}$ if $(A \cdot Z) \circ (B \cdot Z) = u \cdot (C \cdot Z) + E$, where $Z = (W, \mathsf{x}, u)$.*

Note that any R1CS instance can be expressed as a relaxed R1CS instance by augmenting it with $u = 1$ and $E = 0$, so relaxed R1CS retains NP-completeness.

Building on the first attempt, the prover and verifier can now use E to accumulate the cross-terms. In particular, for $Z_i = (W_i, \mathsf{x}_i, u_i)$, the prover and verifier additionally compute

$$\begin{aligned}
&u \leftarrow u_1 + r \cdot u_2 \\
&E \leftarrow E_1 + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1 - u_1 CZ_2 - u_2 CZ_1) + r^2 \cdot E_2,
\end{aligned}$$

and set the new instance-witness pair to be $((E, u, \mathsf{x}), W)$. Conveniently, updating u in this manner also keeps track of how the constant term in Z should be updated, which motivates our choice to use u in $Z = (W, \mathsf{x}, u)$ rather than introducing a new variable. Now, for $Z = (W, \mathsf{x}, u)$, and for random $r \in \mathbb{F}$,

$$\begin{aligned}
AZ \circ BZ &= AZ_1 \circ BZ_1 + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2 \cdot (AZ_2 \circ BZ_2) \\
&= (u_1 CZ_1 + E_1) + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2 \cdot (u_2 CZ_2 + E_2) \\
&= (u_1 + r \cdot u_2) \cdot C(Z_1 + rZ_2) + E \\
&= uCZ + E.
\end{aligned}$$

This implies that, for R1CS matrices (A, B, C), the folded witness W is a satisfying witness for the folded instance (E, u, x) as promised. A few issues remain: in the above scheme, the prover sends witnesses (W_1, W_2) for the verifier to compute E. As a result, the folding scheme is *not* non-trivial; it is also not zero-knowledge.

Final Protocol. To circumvent these issues, we use succinct and hiding additively homomorphic commitments to W and E in the instance, and treat both W and E as the witness. We refer to this variant of relaxed R1CS as *committed relaxed R1CS*. Below, we describe a folding scheme for committed relaxed R1CS, where the prover sends a single commitment to aid the verifier in computing commitments to the folded witness (W, E).

Definition 12 (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 $Z = (W, \mathsf{x}, u)$.

Construction 1 (A Folding Scheme for Committed Relaxed R1CS). Consider a finite field $\mathbb{F}$ and a succinct, hiding, and homomorphic commitment scheme Com over $\mathbb{F}$. We define the generator and the encoder as follows.

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

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

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

Theorem 3 (A Folding Scheme for Committed Relaxed R1CS). *Construction 1 is a public-coin folding scheme for committed relaxed R1CS with perfect completeness, knowledge soundness, and zero-knowledge.*

Proof Intuition. With textbook algebra, we can show that if witnesses $(E_1, r_{E_1}, W_1, r_{W_1})$ and $(E_2, r_{E_2}, W_2, r_{W_2})$ are satisfying witnesses, then the folded witness (E, r_E, W, r_W) must be a satisfying witness. We prove knowledge soundness via the forking lemma (Lemma 1) by showing that the extractor can produce the initial witnesses given three accepting transcripts and the corresponding folded witnesses. Specifically, the extractor uses all three transcripts to compute E_i and r_{E_i}, and any two transcripts to compute W_i and r_{W_i} for $i \in \{1, 2\}$. The choice of which two transcripts does not matter due to the binding property of the commitment scheme. We present a formal proof in [30, App. B]. □

4.2 Achieving Non-Interactivity via the Fiat-Shamir Transform

To design Nova's IVC scheme, we require our folding scheme for committed relaxed R1CS to be non-interactive in the standard model. To do so we first achieve non-interactivity in the random oracle model using the (strong) Fiat-Shamir transform [22]. Next, we heuristically instantiate the random oracle using a cryptographic hash function. As a result, we can only heuristically argue the security of the resulting non-interactive folding scheme. Note that all existing IVC constructions in the standard model require instantiating the random oracle with a cryptographic hash function [12, 15, 21, 43].

Construction 2 (A Non-Interactive Folding Scheme). We achieve non-interactivity in the random oracle model using the strong Fiat-Shamir transform [22]. Let ρ denote a random oracle sampled during parameter generation and provided to all parties. Let $(\mathsf{G}, \mathsf{K}, \mathsf{P}, \mathsf{V})$ represent our interactive folding scheme (Construction 1). We construct a non-interactive folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ as follows:

- $\mathcal{G}(1^\lambda)$: output $\mathsf{pp} \leftarrow \mathsf{G}(1^\lambda)$.
- $\mathcal{K}(\mathsf{pp}, (A, B, C))$: $\mathsf{vk} \leftarrow \rho(\mathsf{pp}, \mathsf{s})$ and $\mathsf{pk} \leftarrow (\mathsf{pp}, (A, B, C), \mathsf{vk})$; output $(\mathsf{vk}, \mathsf{pk})$.
- $\mathcal{P}(\mathsf{pk}, (u_1, w_1), (u_2, w_2))$: runs $\mathsf{P}((\mathsf{pk.pp}, \mathsf{pk}.(A, B, C))$ to retrieve its first message $\overline{T}$, and sends $\overline{T}$ to $\mathcal{V}$; computes $r \leftarrow \rho(\mathsf{vk}, u_1, u_2, \overline{T})$, forwards this to P, and outputs the resulting output.
- $\mathcal{V}(\mathsf{vk}, u_1, u_2, \overline{T})$: runs V with $\overline{T}$ as the message from the prover and with randomness $r \leftarrow \rho(\mathsf{vk}, u_1, u_2, \overline{T})$, and outputs the resulting output.

Assumption 1 (RO instantiation). Construction 2 is a non-interactive folding scheme that satisfies completeness, knowledge soundness, and zero-knowledge in the standard model when ρ is instantiated with a cryptographic hash function.

5 Nova: An IVC Scheme with Proof Compression

This section describes Nova, an IVC scheme designed from a non-interactive folding scheme, which when instantiated with any additively-homomorphic commitment scheme with succinct commitments achieves the claimed efficiency (Lemma 4). In addition, Nova incorporates an efficient zkSNARK to prove the knowledge of valid IVC proofs succinctly and in zero-knowledge, providing a succinct, zero-knowledge proof of knowledge of a valid IVC proof.

In Nova, at each incremental step, the prover folds a particular step of the incremental computation (represented as a committed relaxed R1CS instance-witness pair) into a running committed relaxed R1CS instance-witness pair. At any step in the incremental computation, a valid "IVC proof", in a nutshell, is a satisfying witness of the running committed relaxed R1CS instance (which an honest prover can compute by folding witnesses associated with each step of the incremental computation) along with the running committed relaxed R1CS instance. Furthermore, at any incremental step, Nova's prover can prove in zero-knowledge and with a succinct proof—using a variant of an existing zkSNARK [38] (Section 6)—that it knows a valid IVC proof (i.e., a satisfying witness) to the running committed relaxed R1CS instance (Construction 4).

Note that Nova is *not* a zero-knowledge IVC scheme, as that would additionally require an IVC proof to be zero-knowledge (in Nova's case, an IVC proof does *not* hide witnesses associated with steps of the incremental computation). This difference is immaterial in the context of a single prover since it can use Nova's auxiliary zkSNARK to provide a zero-knowledge proof of knowledge of a valid IVC proof; we leave it to future work to achieve zero-knowledge IVC.

5.1 Constructing IVC from a Folding Scheme for NP

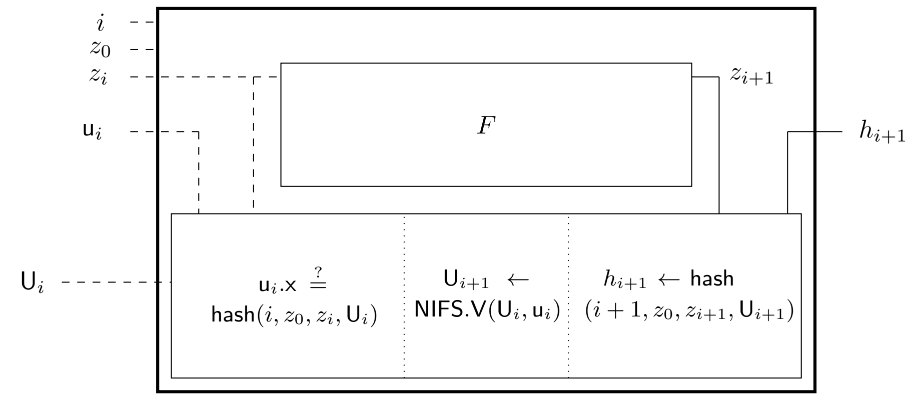
Recall that an IVC scheme allows a prover to show that $z_n = F^{(n)}(z_0)$ for some count n, initial input z_0, and output z_n. We now show how to construct an IVC scheme for a non-deterministic, polynomial-time computable function F using our non-interactive folding scheme for committed relaxed R1CS (Construction 2).[5]

In our construction, as in a SNARK-based IVC, the prover uses an augmented function F' (Figure 4), which, in addition to invoking F, performs additional bookkeeping to fold proofs of prior invocations of itself.

We first describe a simplified version of F', to provide intuition. F' takes as non-deterministic advice two committed relaxed R1CS instances u_i and U_i. Suppose that U_i represents the correct execution of invocations $1, \ldots, i-1$ of F' so long as u_i represents the correct execution of invocation i of F'. F' performs two tasks. First, it executes a step of the incremental computation: instance u_i contains z_i which F' uses to output $z_{i+1} = F(z_i)$. Second, F' invokes the verifier of the non-interactive folding scheme to fold the task of checking u_i and U_i into the task of checking a single instance U_{i+1}. The IVC prover then computes a new instance u_{i+1} which attests to the correct execution of invocation $i+1$ of F', thereby attesting that $z_{i+1} = F(z_i)$ and U_{i+1} is the result of folding u_i and U_i. Now, we have that U_{i+1} represents the correct execution of invocations $1, \ldots, i$ of F' so long as u_{i+1} represents the correct execution of invocation $i+1$ of F'.

The above description glossed over a subtle discrepancy: Because F' must output the running instance U_{i+1} for the next invocation to use, it is contained

[5] While, in theory, we can use any folding scheme for NP, we specifically invoke our construction for committed relaxed R1CS for a simpler presentation.



i
z_0
z_i
u_i
U_i
F
z_{i+1}
h_{i+1}
$\mathsf{u}_i.\mathsf{x} \overset{?}{=}$ $\mathsf{hash}(i, z_0, z_i, \mathsf{U}_i)$
$\mathsf{U}_{i+1} \leftarrow$ $\mathsf{NIFS.V}(\mathsf{U}_i, \mathsf{u}_i)$
$h_{i+1} \leftarrow \mathsf{hash}$ $(i+1, z_0, z_{i+1}, \mathsf{U}_{i+1})$

Fig. 4: Overview of F'. F' represented as a committed relaxed R1CS instance u_{i+1} encodes the statement that there exists $((i, z_0, z_i, \mathsf{u}_i, \mathsf{U}_i), \mathsf{U}_{i+1}, \overline{T})$ such that $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i)$, $h_{i+1} = \mathsf{hash}(\mathsf{vk}, i+1, z_0, F(z_i), \mathsf{U}_{i+1})$, $\mathsf{U}_{i+1} = \mathsf{NIFS}.V(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, \overline{T})$, and that F' outputs h_{i+1}. The diagram omits depicting vk, ω, and $\overline{T}$.

in $\mathsf{u}_{i+1}.\mathsf{x}$ (i.e., the public IO of u_{i+1}). But, in the next iteration, F' must fold $\mathsf{u}_{i+1}.\mathsf{x}$ into $\mathsf{U}_{i+1}.\mathsf{x}$, meaning that F' is stuck trying to squeeze U_{i+1} into $\mathsf{U}_{i+1}.\mathsf{x}$. To handle this inconsistency, we modify F' to output a *collision-resistant hash* of its public IO rather than producing it directly (this ensures that the public IO of F' is a constant number of finite field elements). The next invocation of F' then additionally takes the preimage of this hash as non-deterministic advice. We assume that the hash function takes an additional random input (which provides hiding) but for notational convenience we do not explicitly depict this.

Producing IVC Proofs. Let $(\mathsf{u}_\perp, \mathsf{w}_\perp)$ be the trivially satisfying instance-witness pair, 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.

Now, in iteration $i+1$, the IVC prover runs F' and computes u_{i+1} and U_{i+1} as well as the corresponding witnesses w_{i+1} and W_{i+1}. Because u_{i+1} and U_{i+1} together attest to the correctness of $i+1$ invocations of F' (which indirectly attests to $i+1$ invocations of F) the IVC proof Π_{i+1} is $((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}))$. Moreover, succinctness is maintained by the properties of the underlying folding scheme. We formally describe our construction below.

Construction 3 (IVC). Let $\mathsf{NIFS} = (\mathsf{G}, \mathsf{K}, \mathsf{P}, \mathsf{V})$ be the non-interactive folding scheme for committed relaxed R1CS (Construction 2). 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):

$\underline{F'(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \overline{T}) \to \mathsf{x}}$:

If i is 0, output $\mathsf{hash}(\mathsf{vk}, 1, z_0, F(z_0, \omega_i), \mathsf{u}_\perp)$;

otherwise,

(1) check that $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i)$, where $\mathsf{u}_i.\mathsf{x}$ is the public IO of u_i,
(2) check that $(\mathsf{u}_i.\overline{E}, \mathsf{u}_i.u) = (\mathsf{u}_\perp.\overline{E}, 1)$,
(3) compute $\mathsf{U}_{i+1} \leftarrow \mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}, \mathsf{u}, \overline{T})$, and
(4) output $\mathsf{hash}(\mathsf{vk}, i+1, z_0, F(z_i, \omega_i), \mathsf{U}_{i+1})$.

Because F' can be computed in polynomial time, it can be represented as a committed relaxed R1CS structure $\mathsf{s}_{F'}$. Let

$$(\mathsf{u}_{i+1}, \mathsf{w}_{i+1}) \leftarrow \mathsf{trace}(F', (\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \overline{T}))$$

denote the satisfying committed relaxed R1CS instance-witness pair $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$ for the execution of F' on non-deterministic advice $(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \overline{T})$.

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

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

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

Compute $(\mathsf{pk_{fs}}, \mathsf{vk_{fs}}) \leftarrow \mathsf{NIFS.K}(\mathsf{pp}, \mathsf{s}_{F'})$ and output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((F, \mathsf{pk_{fs}}), (F, \mathsf{vk_{fs}}))$.

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

Parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$ and then

(1) if i is 0, compute $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \overline{T}) \leftarrow (\mathsf{u}_\perp, \mathsf{w}_\perp, \mathsf{u}_\perp.\overline{E})$;
otherwise, compute $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \overline{T}) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}, (\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$,
(2) compute $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1}) \leftarrow \mathsf{trace}(F', (\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \overline{T}))$, and
(3) output $\Pi_{i+1} \leftarrow ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}))$.

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

If i is 0, check that $z_i = z_0$;
otherwise,

(1) parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$,
(2) check that $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i)$,
(3) check that $(\mathsf{u}_i.\overline{E}, \mathsf{u}.u) = (\mathsf{u}_\perp.\overline{E}, 1)$, and
(4) check that W_i and w_i are satisfying witnesses to U_i and u_i respectively.

Lemma 2 (Completeness). *Construction 3 is an IVC scheme that satisfies completeness.*

Proof Intuition. Given a satisfying IVC proof $\Pi_i = ((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$ suppose that $\mathcal{P}$ outputs $\Pi_{i+1} = ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}))$. 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. Thus, Π_{i+1} is satisfying. [30, App. C] provides a formal proof. □

Lemma 3 (Knowledge Soundness). *Construction 3 is an IVC scheme that satisfies knowledge soundness.*

Proof Intuition. 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 $(\omega_0, \dots, \omega_{n-1})$ such that by computing $z_i \leftarrow F(z_{i-1}, \omega_{i-1})$ for all $i \in \{1, \dots, n\}$ 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 $(z_i, \dots, z_{n-1})$, $(\omega_i, \dots, \omega_{n-1})$, and Π_i such that for all $j \in \{i+1, \dots, n\}$, $z_j = F(z_{j-1}, \omega_{j-1})$, $\mathcal{V}(\mathsf{vk}, i, z_0, z_i, \Pi_i) = 1$, 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 $(\omega_0, \dots, \omega_{n-1})$. 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 $\mathcal{E}_{i-1}$. [30, App. C] provides a formal proof. □

Lemma 4 (Efficiency). *When instantiated with the Pedersen commitment scheme, we have that* $|F'| = |F| + o(2 \cdot \mathsf{G} + 2 \cdot \mathsf{H} + \mathsf{R})$*, where* $|F|$ *denotes the number of R1CS constraints to encode a function* F*,* G *is the number of constraints required to encode a group scalar multiplication,* H *is the number of constraints required to encode* hash*, and* R *is the number of constraints to encode the RO* ρ*.*

Proof. On input instances U and u, $\mathsf{NIFS.V}$ computes $\overline{E} \leftarrow \mathsf{U}.\overline{E} + r \cdot \overline{T} + r^2 \cdot \mathsf{u}.\overline{E}$ and $\overline{W} \leftarrow \mathsf{U}.\overline{W} + r \cdot \mathsf{u}.\overline{W}$. However, by construction, $\mathsf{u}.\overline{E} = \mathsf{u}_\perp.\overline{E} = \overline{0}$. So, $\mathsf{NIFS.V}$ computes two group scalar multiplications, as it does not need to compute $r^2 \cdot \mathsf{u}.\overline{E}$. $\mathsf{NIFS.V}$ additionally invokes the RO once to obtain a random scalar. Finally, F' makes two additional calls to hash (details are in the description of F'). □

5.2 Compressing IVC Proofs with zkSNARKs

To prove a statement about an incremental computation, the prover can produce an IVC proof using the construction in the prior section and send the IVC proof to the verifier. However, this does not satisfy zero-knowledge (as the IVC proof described in the prior section does not hide the prover's non-deterministic inputs) and succinctness (as the IVC proof size is linear in the size F). In theory, one can address this problem with any zkSNARK for NP. Specifically, $\overline{\mathcal{P}}$ can produce a zkSNARK proving that it knows Π_i such that IVC verifier $\mathcal{V}$ accepts for statement (i, z_0, z_i). Naturally, the proof sent to the verifier is succinct and zero-knowledge due to the corresponding properties of the zkSNARK.

Unfortunately, employing an off-the-shelf zkSNARK makes the overall solution impractical as the zkSNARK 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 the programming model of zkSNARKs (e.g., R1CS or circuits). To address this, we design a zkSNARK tailored for our particular purpose and we describe it in Section 6. Below, we describe how to use a zkSNARK to prove the knowledge of a valid IVC proof.

Construction 4 (A zkSNARK of a Valid IVC Proof). Let IVC denote the IVC scheme in Construction 3, let NIFS denote the non-interactive folding scheme in Construction 2, and let hash denote a randomized cryptographic hash function. Assume a zero-knowledge succinct non-interactive argument of knowledge (Definition 2), $\mathsf{zkSNARK}$, for committed relaxed R1CS. That is, given public parameters pp, structure s, and instance u, $\mathsf{zkSNARK.P}$ can convince $\mathsf{zkSNARK.V}$ in zero-knowledge and with a succinct proof (e.g., $O_\lambda(\log N)$-sized proof) that it knows a corresponding witness w such that $(\mathsf{pp}, \mathsf{s}, \mathsf{u}, \mathsf{w})$ is a satisfying committed relaxed R1CS tuple.

Consider a polynomial-time computable function F. Suppose $\mathsf{pp} \leftarrow \mathsf{IVC.G}(1^\lambda)$ and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{IVC.K}(\mathsf{pp}, F)$. Suppose the prover $\mathcal{P}$ and verifier $\mathcal{V}$ are provided an instance (i, z_0, z_i). We construct a zkSNARK that allows the prover to show that it knows an IVC proof Π_i such that $\mathsf{IVC.V}(\mathsf{vk}, i, z_0, z_i, \Pi_i) = 1$.

In a nutshell, we leverage the fact that Π is two committed relaxed R1CS instance-witness pairs. So, $\mathcal{P}$ first folds instance-witness pairs (u, w) and (U, W) to produce a folded instance-witness pair $(\mathsf{U}', \mathsf{W}')$, using $\mathsf{NIFS.P}$. Next, $\mathcal{P}$ runs $\mathsf{zkSNARK.P}$ to prove that it knows a valid witness for U'. In more detail, for polynomial-time computable function F and corresponding function F' as defined in Construction 3 (and instantiated with hash), we define $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ as follows.

$\underline{\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}}$:

(1) Compute $\mathsf{pp}_{\mathsf{NIFS}} \leftarrow \mathsf{NIFS.G}(1^\lambda)$
(2) Compute $\mathsf{pp}_{\mathsf{zkSNARK}} \leftarrow \mathsf{zkSNARK.G}(1^\lambda)$
(3) Output $(\mathsf{pp}_{\mathsf{NIFS}}, \mathsf{pp}_{\mathsf{zkSNARK}})$

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

(1) Compute $(\mathsf{pk}_{\mathsf{NIFS}}, \mathsf{vk}_{\mathsf{NIFS}}) \leftarrow \mathsf{NIFS.K}(\mathsf{pp.pp}_{\mathsf{NIFS}}, \mathsf{s}_{F'})$.
(2) Compute $(\mathsf{pk}_{\mathsf{zkSNARK}}, \mathsf{vk}_{\mathsf{zkSNARK}}) \leftarrow \mathsf{zkSNARK.K}(\mathsf{pp.pp}_{\mathsf{zkSNARK}}, \mathsf{s}_{F'})$.
(3) Output $((\mathsf{pk}_{\mathsf{NIFS}}, \mathsf{pk}_{\mathsf{zkSNARK}}), (\mathsf{vk}_{\mathsf{NIFS}}, \mathsf{vk}_{\mathsf{zkSNARK}}))$.

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

```
If $i$ is 0, output $\perp$;
otherwise,
    (1) parse $\Pi$ as $((\mathsf{U}, \mathsf{W}), (\mathsf{u}, \mathsf{w}))$
    (2) compute $(\mathsf{U}', \mathsf{W}', \overline{T}) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}_{\mathsf{NIFS}}, (\mathsf{U}, \mathsf{W}), (\mathsf{u}, \mathsf{w}))$
    (3) compute $\pi_{\mathsf{U}'} \leftarrow \mathsf{zkSNARK.P}(\mathsf{pk}_{\mathsf{zkSNARK}}, \mathsf{U}', \mathsf{W}')$
    (4) output $(\mathsf{U}, \mathsf{u}, \overline{T}, \pi_{\mathsf{U}'})$.
```

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

```
If $i$ is 0, check that $z_0 = z_i$;
otherwise,
```

(1) parse π as $(\mathsf{U}, \mathsf{u}, \overline{T}, \pi_{\mathsf{U}'})$,
(2) check that $\mathsf{u}.\mathsf{x} = \mathsf{hash}(\mathsf{vk}_{\mathsf{NIFS}}, i, z_0, z_i, \mathsf{U})$,
(3) check that $(\mathsf{u}.\overline{E}, \mathsf{u}.u) = (\mathsf{u}_{\perp}.\overline{E}, 1)$,
(4) compute $\mathsf{U}' \leftarrow \mathsf{NIFS.V}(\mathsf{vk}_{\mathsf{NIFS}}, \mathsf{U}, \mathsf{u}, \overline{T})$, and
(5) check that $\mathsf{zkSNARK.V}(\mathsf{vk}_{\mathsf{zkSNARK}}, \mathsf{U}', \pi_{\mathsf{U}'}) = 1$.

Theorem 4. *Construction 4 is a zkSNARK of a valid IVC proof produced by Construction 3.*

Proof Intuition. Completeness and knowledge soundness hold due to the completeness and knowledge soundness of the underlying zkSNARK and the non-interactive folding scheme. Assuming the non-interactive folding scheme satisfies succinctness (e.g., by using the Pedersen commitment scheme), succinctness holds due to the fact that u, U, and $\overline{T}$ are succinct, and due to the succinctness of the underling zkSNARK.

To prove zero-knowledge, we construct a simulator $\mathcal{S}$ that first iteratively simulates $(\mathsf{U}_i, \mathsf{u}_i)$ for all $i \in \{1, \ldots, n\}$. Specifically, given a simulated proof $(\mathsf{U}_i, \mathsf{u}_i)$, $\mathcal{S}$ first uses the simulator of the non-interactive folding scheme to simulate $\overline{T}_i$. $\mathcal{S}$ then folds U_i and u_i using $\overline{T}_i$ to produce U_{i+1}. $\mathcal{S}$ simulates u_i using the observation that all terms are randomized. In the final round, $\mathcal{S}$ folds u_n and U_n (again using a simulated $\overline{T}_n$) to produce an instance U', and then uses the simulator of the zkSNARK to produce $\pi_{\mathsf{U}'}$. $\mathcal{S}$ outputs $(\mathsf{U}_n, \mathsf{u}_n, \overline{T}_n, \pi_{\mathsf{U}'})$. We provide a formal proof in [30, App. D]. □

6 A zkSNARK for Committed Relaxed R1CS

As described in Section 5.2, Nova needs a zkSNARK for committed relaxed R1CS to prove the knowledge of a valid IVC proof succinctly and in zero-knowledge. This section presents such a zkSNARK by adapting Spartan [38]. We build on Spartan [38] to avoid FFTs and a trusted setup.

6.1 Background

We assume familiarity with polynomials. We provide background in [30, App. G].

Definition 13 (Polynomial Extension). *Suppose $f : \{0,1\}^{\ell} \to \mathbb{F}$ is a function that maps ℓ-bit strings to an element of $\mathbb{F}$. A polynomial extension of f is a low-degree ℓ-variate polynomial $\widetilde{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ such that $\widetilde{f}(x) = f(x)$ for all $x \in \{0,1\}^{\ell}$. A multilinear extension (MLE) of a function $f : \{0,1\}^{\ell} \to \mathbb{F}$ is a low-degree polynomial extension where the extension is a multilinear polynomial.*

Every function $f : \{0,1\}^{\ell} \to \mathbb{F}$ has a unique MLE, and conversely every ℓ-variate multilinear polynomial over $\mathbb{F}$ extends a unique function mapping $\{0,1\}^{\ell} \to \mathbb{F}$. Below, we use $\widetilde{f}$ to denote the unique MLE of f.

Lemma 5 (The Sum-Check Protocol [35]). *For ℓ-variate polynomial G over $\mathbb{F}$ with degree at most μ in each variable, there exists a public-coin interactive proof protocol (known as the sum-check protocol) to reduce the task of checking $\sum_{x \in \{0,1\}^\ell} G(x) = T$ to the task of checking $G(r) = e$ for $r \in \mathbb{F}^\ell$. The interaction consists of a total of ℓ rounds, where in each round the verifier sends a single element of $\mathbb{F}$ and the prover responds with $\mu + 1$ elements of $\mathbb{F}$.*

6.2 A Polynomial IOP for Idealized Relaxed R1CS

Our exposition below is based on Spartan [38] and its recent recapitulation [34]. The theorem below and its proof is a verbatim adaptation of Spartan's polynomial IOP for R1CS to relaxed R1CS.

Recall that an interactive proof (IP) [25] for a relation $\mathcal{R}$ is an interactive protocol between a prover and a verifier where the prover proves the knowledge of a witness w for a prescribed instance u such that $(u, w) \in \mathcal{R}$. An interactive oracle proof (IOP) [5, 37] generalizes interactive proofs where in each round the prover may send an oracle (e.g., a string) and the verifier may query a previously-sent oracle during the remainder of the protocol. A polynomial IOP [17] is an IOP in which the oracle sent by the prover is a polynomial and the verifier may query for an evaluation of the polynomial at a point in its domain. We consider a (minor) variant of polynomial IOPs, where the verifier has oracle access to polynomials in the R1CS structure and instance.

We first construct a polynomial IOP for an idealized version of relaxed R1CS (Definition 14) where the instance contains a purported witness. We then compile it into a zkSNARK for committed relaxed R1CS (Definition 12).

Definition 14 (Idealized Relaxed R1CS). *Consider a finite field $\mathbb{F}$. Let the public parameters consist of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$. The idealized 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 idealized relaxed R1CS instance consists of an error vector $E \in \mathbb{F}^m$, a scalar $u \in \mathbb{F}$, witness vector $W \in \mathbb{F}^m$, and public inputs and outputs $\mathsf{x} \in \mathbb{F}^\ell$. An instance (E, u, W, x) is satisfying if $(A \cdot Z) \circ (B \cdot Z) = u \cdot (C \cdot Z) + E$, where $Z = (W, \mathsf{x}, u)$.*

Construction 5 (Polynomial IOP for Idealized Relaxed R1CS). Consider an idealized relaxed R1CS statement φ consisting of public parameters (m, n, ℓ), structure (A, B, C), and instance (E, u, W, x), Without loss of generality, we assume that m and n are powers of 2 and that $m = 2 \cdot (\ell + 1)$.

Let $s = \log m$. We interpret the matrices A, B, C as functions with signature $\{0,1\}^{\log m} \times \{0,1\}^{\log m} \to \mathbb{F}$ in a natural manner. In particular, an input in $\{0,1\}^{\log m} \times \{0,1\}^{\log m}$ is interpreted as the binary representation of an index $(i, j) \in [m] \times [m]$, where $[m] := \{1, \ldots, m\}$ and the function outputs (i, j)th entry of the matrix. As such, let $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$ denote multilinear extensions of A, B, and C interpreted as functions, so they are $2 \log m$-variate sparse multilinear polynomials of size n. Similarly, we interpret E and W as functions with respective signatures $\{0,1\}^{\log m} \to \mathbb{F}$ and $\{0,1\}^{\log m - 1} \to \mathbb{F}$. Furthermore, let $\widetilde{E}$ and $\widetilde{W}$

denote the multilinear extensions of E and W interpreted as functions, so they are multilinear polynomials in $\log m$ and $\log m - 1$ variables respectively.

As noted earlier, the verifier has an oracle access to the following polynomials: $\widetilde{A}$, $\widetilde{B}$, $\widetilde{C}$, $\widetilde{E}$, and $\widetilde{W}$. Additionally, the verifier reads u and x in entirety.

Let $Z = (W, \mathsf{x}, u)$. Similar to how we interpret matrices as functions, we interpret Z and (x, u) as functions with the following respective signatures: $\{0,1\}^s \to \mathbb{F}$ and $\{0,1\}^{s-1} \to \mathbb{F}$. Observe that the MLE $\widetilde{Z}$ of Z satisfies

$$\widetilde{Z}(X_1, \ldots, X_s) = (1 - X_1) \cdot \widetilde{W}(X_2, \ldots, X_s) + X_1 \cdot \widetilde{(\mathsf{x}, u)}(X_2, \ldots, X_s) \tag{1}$$

Similar to [38, Theorem 4.1], checking if φ is satisfiable is equivalent, except for a soundness error of $\log m/|\mathbb{F}|$ over the choice of $\tau \in \mathbb{F}^s$, to checking if the following identity holds:

$$0 \overset{?}{=} \sum_{x \in \{0,1\}^s} \widetilde{\mathsf{eq}}(\tau, x) \cdot F(x), \tag{2}$$

where

$$F(x) = \left(\sum_{y \in \{0,1\}^s} \widetilde{A}(x,y) \cdot \widetilde{Z}(y) \right) \cdot \left(\sum_{y \in \{0,1\}^s} \widetilde{B}(x,y) \cdot \widetilde{Z}(y) \right) - \left(u \cdot \sum_{y \in \{0,1\}^s} \widetilde{C}(x,y) \cdot \widetilde{Z}(y) + \widetilde{E}(x) \right),$$

and $\widetilde{\mathsf{eq}}$ is the multilinear extension of $\mathsf{eq} : \{0,1\}^s \times \{0,1\}^s \to \mathbb{F}$ where $\mathsf{eq}(x, e) = 1$ if $x = e$ and 0 otherwise.

That is, if φ is satisfiable, then Equation (2) holds with probability 1 over the choice of τ, and if not, then Equation (2) holds with probability at most $O(\log m/|\mathbb{F}|)$ over the random choice of τ.

To compute the right-hand side in Equation (2), the prover and the verifier apply the sum-check protocol to the following polynomial: $g(x) \coloneqq \widetilde{\mathsf{eq}}(\tau, x) \cdot F(x)$ From the verifier's perspective, this reduces the task of computing the right-hand side of Equation (2) to the task of evaluating g at a random input $r_x \in \mathbb{F}^s$. Note that the verifier can locally evaluate $\widetilde{\mathsf{eq}}(\tau, r_x)$ in $O(\log m)$ field operations via $\widetilde{\mathsf{eq}}(\tau, r_x) = \prod_{i=1}^{s} (\tau_i r_{x,i} + (1 - \tau_i)(1 - r_{x,i}))$. With $\widetilde{\mathsf{eq}}(\tau, r_x)$ in hand, $g(r_x)$ can be computed in $O(1)$ time given the four quantities: $\sum_{y \in \{0,1\}^s} \widetilde{A}(r_x, y) \cdot \widetilde{Z}(y)$, $\sum_{y \in \{0,1\}^s} \widetilde{B}(r_x, y) \cdot \widetilde{Z}(y)$, $\sum_{y \in \{0,1\}^s} \widetilde{C}(r_x, y) \cdot \widetilde{Z}(y)$, and $\widetilde{E}(r_x)$.

The last quantity can be computed with a single query to polynomial $\widetilde{E}$. Furthermore, the first three quantities can be computed by applying the sum-check protocol three more times in parallel, once to each of the following three polynomials (using the same random vector of field elements, $r_y \in \mathbb{F}^s$, in each of the three invocations): $\widetilde{A}(r_x, y) \cdot \widetilde{Z}(y)$, $\widetilde{B}(r_x, y) \cdot \widetilde{Z}(y)$, and $\widetilde{C}(r_x, y) \cdot \widetilde{Z}(y)$.

To perform the verifier's final check in each of these three invocations of the sum-check protocol, it suffices for the verifier to evaluate each of the above three polynomials at the random vector r_y, which means it suffices for the verifier to evaluate $\widetilde{A}(r_x, r_y)$, $\widetilde{B}(r_x, r_y)$, $\widetilde{C}(r_x, r_y)$, and $\widetilde{Z}(r_y)$. The first three evaluations can be obtained via the verifier's assumed query access to $(\widetilde{A}, \widetilde{B}, \widetilde{C})$. $\widetilde{Z}(r_y)$ can be computed (via Equation (1)) from a query to $\widetilde{W}$ and from computing $\widetilde{(\mathsf{x}, u)}$.

In summary, we have the following polynomial IOP.

1. $\mathcal{V} \rightarrow \mathcal{P}$: $\tau \in_R \mathbb{F}^s$
2. $\mathcal{V} \leftrightarrow \mathcal{P}$: run the sum-check protocol to reduce the check in Equation (2) to checking if the following hold, where r_x, r_y are vectors in $\mathbb{F}^s$ chosen at random by the verifier over the course of the sum-check protocol:
   - $\widetilde{A}(r_x, r_y) \stackrel{?}{=} v_A$, $\widetilde{B}(r_x, r_y) \stackrel{?}{=} v_B$, and $\widetilde{C}(r_x, r_y) \stackrel{?}{=} v_C$;
   - $\widetilde{E}(r_x) \stackrel{?}{=} v_E$; and
   - $\widetilde{Z}(r_y) \stackrel{?}{=} v_Z$.
3. $\mathcal{V}$:
   - check if $\widetilde{A}(r_x, r_y) \stackrel{?}{=} v_A$, $\widetilde{B}(r_x, r_y) \stackrel{?}{=} v_B$, and $\widetilde{C}(r_x, r_y) \stackrel{?}{=} v_C$, with a query to $\widetilde{A}, \widetilde{B}, \widetilde{C}$ at (r_x, r_y);
   - check if $\widetilde{E}(r_x) \stackrel{?}{=} v_E$ with an oracle query to $\widetilde{E}$; and
   - check if $\widetilde{Z}(r_y) \stackrel{?}{=} v_Z$ by checking if: $v_Z = (1 - r_y[1]) \cdot v_W + r_y[1] \cdot \widetilde{(\mathsf{x}, u)}(r_y[2..])$, where $r_y[2..]$ refers to a slice of r_y without the first element of r_y, and $v_W \leftarrow \widetilde{W}(r_y[2..])$ via an oracle query (see Equation (1)).

Theorem 5. *Construction 5 is a polynomial IOP for idealized relaxed R1CS defined over a finite field $\mathbb{F}$, with the following parameters, where m denotes the dimension of the R1CS matrices, and n denotes the number of non-zero entries in the matrices: Soundness error is $O(\log m)/|\mathbb{F}|$; round complexity is $O(\log m)$; The verifier has query access to $2\log m$-variate multilinear polynomials $\widetilde{A}, \widetilde{B}, \widetilde{C}$ in the structure, and $(\log m)$-variate multilinear polynomial $\widetilde{E}$, and $(\log m - 1)$-variate multilinear polynomial $\widetilde{W}$ in the instance; the verifier issues a single query to polynomials $\widetilde{A}$, $\widetilde{B}$, $\widetilde{C}$, and $\widetilde{W}$, $\widetilde{E}$, and otherwise performs $O(\log m)$ operations over $\mathbb{F}$; the prover performs $O(n)$ operations over $\mathbb{F}$ to compute its messages in the polynomial IOP and to respond to the verifier's queries to $(\widetilde{W}, \widetilde{E}, \widetilde{A}, \widetilde{B}, \widetilde{C})$.*

Proof. Perfect completeness follows from perfect completeness of the sum-check protocol and the fact that Equation (2) holds with probability 1 over the choice of τ if φ is satisfiable. Applying a standard union bound to the soundness error introduced by probabilistic check in Equation (2) with the soundness error of the sum-check protocol [35], we conclude that the soundness error for the depicted polynomial IOP as at most $O(\log m)/|\mathbb{F}|$.

The sum-check protocol is applied four times (although three of the invocations occur in parallel and in practice combined into one [38]). In each invocation, the polynomial to which the sum-check protocol is applied has degree at most 3 in each variable, and the number of variables is $s = \log m$. Hence, the round complexity of the polynomial IOP is $O(\log m)$. Since each polynomial has degree at most 3 in each variable, the total communication cost is $O(\log m)$ field elements.

The claimed verifier runtime is immediate from the verifier's runtime in the sum-check protocol, and the fact that $\widetilde{\mathsf{eq}}$ can be evaluated at any input $(\tau, r_x) \in \mathbb{F}^{2s}$ in $O(\log m)$ field operations. As in Spartan [38], the prover's work in the polynomial IOP in $O(n)$ operations over $\mathbb{F}$ using prior techniques [42,46]. $\square$

6.3 Compiling Polynomial IOPs to zkSNARKs

As in prior works [17,20,38], we compile our polynomial IOP into a zkSNARK using a polynomial commitment scheme [28] and the Fiat-Shamir transform [22].

Interpreting commitments to vectors as polynomial commitments. It is well known that commitments to m-sized vectors over $\mathbb{F}$ are commitments to $\log m$-variate multilinear polynomials represented with evaluations over $\{0,1\}^m$ [32,38, 44,47]. Furthermore, there is a polynomial commitment scheme for $\log m$-variate multilinear polynomials if there exists an argument protocol to prove an inner product computation between a committed vector and an m-sized public vector $((r_1, 1-r_1) \otimes \ldots \otimes (r_{\log m}, 1-r_{\log m}))$, where $r \in \mathbb{F}^{\log m}$ is an evaluation point. There are two candidate constructions in the literature. Note that the primary difference between two schemes is in the verifier's time.

1. $\mathsf{PC}_{\mathsf{BP}}$. If the commitment scheme for vectors over $\mathbb{F}$ is Pedersen's commitments, as in prior work [44], Bulletproofs [14] provides a suitable inner product argument protocol. The polynomial commitment scheme here achieves the following efficiency characteristics, assuming the hardness of the discrete logarithm problem. For a $\log m$-variate multilinear polynomial, committing takes $O_\lambda(m)$ time to produce an $O_\lambda(1)$-sized commitment; the prover incurs $O_\lambda(m)$ costs to produce an evaluation proof of size $O_\lambda(\log m)$ that can be verified in $O_\lambda(m)$. Note that $\mathsf{PC}_{\mathsf{BP}}$ is a special case of Hyrax's polynomial commitment scheme [44].
2. $\mathsf{PC}_{\mathsf{Dory}}$. If vectors over $\mathbb{F}$ are committed with a two-tiered "matrix" commitment (see for example, [18,32]), which provides $O_\lambda(1)$-sized commitments to m-sized vectors under the SXDH assumption. With this commitment scheme, Dory [32] provides the necessary inner product argument. The polynomial commitment here achieves the following efficiency characteristics, assuming the hardness of SXDH. For a $\log m$-variate multilinear polynomial, committing takes $O_\lambda(m)$ time to produce an $O_\lambda(1)$-sized commitment; the prover incurs $O_\lambda(m)$ costs to produce an evaluation proof of size $O_\lambda(\log m)$ that can be verified in $O_\lambda(\log m)$.

Polynomial commitments for sparse multilinear polynomials. In our constructions below, we require polynomial commitment schemes that can efficiently handle sparse multilinear polynomials. Spartan [38, §7] (and its optimization [41, §6]) provides a generic compiler to transform existing polynomial commitment schemes for multilinear polynomials into those that can efficiently handle sparse multilinear polynomials. Specifically, we apply [34, Theorem 5]) (which captures Spartan's compiler in a generic manner) to $\mathsf{PC}_{\mathsf{BP}}$ and $\mathsf{PC}_{\mathsf{Dory}}$ to obtain their variants that can efficiently handle sparse multilinear polynomials; we refer to them as "Sparse-$\mathsf{PC}_{\mathsf{BP}}$" and "Sparse-$\mathsf{PC}_{\mathsf{Dory}}$" respectively.

Theorem 6 (A zkSNARK from $\mathsf{PC}_{\mathsf{BP}}$). *Assuming the hardness of the discrete logarithm problem, there exists a zkSNARK in the random oracle model for committed relaxed R1CS with the following efficiency characteristics, where m denotes the dimensions of R1CS matrices and n denotes the number of non-zero entries in the matrices: The encoder runs in time $O_\lambda(n)$; The prover runs in time $O_\lambda(n)$; The proof length is $O_\lambda(\log n)$; and the verifier runs in time $O_\lambda(n)$.*[6]

Proof. For R1CS structure (A, B, C), we first have the encoder directly provide $(\widetilde{A}, \widetilde{B}, \widetilde{C})$ in the prover key, and additionally provide sparse polynomial commitments to $\widetilde{A}, \widetilde{B}, \widetilde{C}$ using Sparse-$\mathsf{PC}_{\mathsf{BP}}$ in both the prover and verifier keys. Next, we apply the compiler of [17] using $\mathsf{PC}_{\mathsf{BP}}$ to the polynomial IOP from Construction 5. At a high level, this replaces all of the oracles provided to the verifier with $\mathsf{PC}_{\mathsf{BP}}$ commitments, which the prover and verifier then use to simulate ideal queries to a committed oracle. By [17, Theorem 6] this provides a public-coin honest-verifier zero-knowledge interactive argument of knowledge. In particular, we can treat the resulting protocol as an argument for committed relaxed R1CS because the verifier is now provided with (polynomial) commitments to E and W. Applying the Fiat-Shamir transform [22] achieves non-interactivity and zero-knowledge in the random oracle model.

The claimed efficiency follows from the efficiency of the polynomial IOP, $\mathsf{PC}_{\mathsf{BP}}$, and Sparse-$\mathsf{PC}_{\mathsf{BP}}$. In more detail, using Sparse-$\mathsf{PC}_{\mathsf{BP}}$, the encoder takes $O_\lambda(n)$ time to create commitments $2 \log m$-variate sparse multilinear polynomials $\widetilde{A}, \widetilde{B}, \widetilde{C}$. The prover's costs in the polynomial IOP is $O(n)$. Furthermore, proving the evaluations of two $O(\log m)$-variate multilinear polynomials using $\mathsf{PC}_{\mathsf{BP}}$, it takes $O_\lambda(m)$ time. And, to prove the evaluations of three $2 \log m$-variate sparse multilinear polynomials of size n, using Sparse-$\mathsf{PC}_{\mathsf{BP}}$, it takes $O_\lambda(n)$ time. In total, the prover time is $O_\lambda(n)$. The proof length in the polynomial IOP is $O(\log m)$, and the proof sizes in the polynomial evaluation proofs is $O_\lambda(\log n)$, so the proof length is $O_\lambda(\log n)$. The verifier's time in the polynomial IOP is $O(\log m)$. In addition, it verifies five polynomial evaluations, which costs $O_\lambda(n)$ time: the two polynomial in the instance take $O_\lambda(m)$ time using $\mathsf{PC}_{\mathsf{BP}}$, and the three polynomials in the structure takes $O_\lambda(n)$ time using Sparse-$\mathsf{PC}_{\mathsf{BP}}$. So, in total, the verifier time is $O_\lambda(n)$. □

Corollary 1 (A zkSNARK from $\mathsf{PC}_{\mathsf{Dory}}$). *Assuming the hardness of the SXDH problem, there exists a zkSNARK in the random oracle model for committed relaxed R1CS with the following efficiency characteristics, where m denotes the dimensions of R1CS matrices and n denotes the number of non-zero entries in the matrices: The encoder runs in time $O_\lambda(n)$; The prover runs in time $O_\lambda(n)$; The proof length is $O_\lambda(\log n)$; and the verifier runs in time $O_\lambda(\log n)$.*

[6] [30, App. H] describes a minor optimization and a corresponding Corollary.

References

[1] bellperson. https://github.com/filecoin-project/bellperson

[2] neptune. https://github.com/filecoin-project/neptune

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

[4] Pasta curves. https://github.com/zcash/pasta

[5] Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive Oracle Proofs. In: TCC (2016)

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

[7] Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: ITCS (2012)

[8] Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKs and proof-carrying data. In: STOC (2013)

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

[10] Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo Infinite: Recursive zk-SNARKs from any Additive Polynomial Commitment Scheme. Cryptology ePrint Archive, Report 2020/1536 (2020)

[11] Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: EUROCRYPT (2016)

[12] Bowe, S., Grigg, J., Hopwood, D.: Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021 (2019)

[13] Bowe, S., Grigg, J., Hopwood, D.: Halo2 (2020), https://github.com/zcash/halo2

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

[15] Bünz, B., Chiesa, A., Lin, W., Mishra, P., Spooner, N.: Proof-carrying data without succinct arguments. Cryptology ePrint Archive, Report 2020/1618 (2020)

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

[17] Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: EUROCRYPT (2020)

[18] Bünz, B., Maller, M., Mishra, P., Vesely, N.: Proofs for inner pairing products and applications. Cryptology ePrint Archive, Report 2019/1177 (2019)

[19] Chen, W., Chiesa, A., Dauterman, E., Ward, N.P.: Reducing participation costs via incremental verification for ledger systems. Cryptology ePrint Archive, Report 2020/1522 (2020)

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

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

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

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

[24] Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: STOC. pp. 99–108 (2011)

[25] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: STOC (1985)

[26] 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)

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

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

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

[30] Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. Cryptology ePrint Archive, Paper 2021/370 (2021)

[31] Labs, O.: Mina cryptocurrency (2020), https://minaprotocol.com

[32] Lee, J.: Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. Cryptology ePrint Archive, Report 2020/1274 (2020)

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

[34] Lee, J., Setty, S., Thaler, J., Wahby, R.: Linear-time zero-knowledge SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/030 (2021)

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

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

[37] Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. In: STOC. pp. 49–62 (2016)

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

[39] Setty, S., Angel, S., Gupta, T., Lee, J.: Proving the correct execution of concurrent services in zero-knowledge. In: OSDI (Oct 2018)

[40] Setty, S., Braun, B., Vu, V., Blumberg, A.J., Parno, B., Walfish, M.: Resolving the conflict between generality and plausibility in verified computation. In: EuroSys (Apr 2013)

[41] Setty, S., Lee, J.: Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275 (2020)
[42] Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: CRYPTO (2013)
[43] Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: TCC. pp. 552–576 (2008)
[44] Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: S&P (2018)
[45] Wesolowski, B.: Efficient verifiable delay functions. In: EUROCRYPT. pp. 379–407 (2019)
[46] Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: CRYPTO (2019)
[47] Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In: S&P (2017)