Amortized Multi-Verifier Proofs from Reductions of Knowledge

Nikitas Paslis
nikitas.paslis@upf.edu
Universitat Pompeu Fabra

Carla Ràfols
carla.rafols@upf.edu
Universitat Pompeu Fabra

Alexandros Zacharakis
alexandros.zacharakis@hpi.de
Hasso Plattner Institute, University of Potsdam

October 26, 2026

Abstract

We study amortization of prover work in the multi-verifier setting, motivated by proof-as-a-service deployments in which a shared prover serves K independent clients holding distinct statements. Each verifier checks only its own statement and proof, with no inter-verifier communication. The challenge is therefore to amortize prover work across many proofs while preserving local verification.

We consider polynomial relations arising naturally in IOP-based proof systems, where verification reduces to polynomial identities and polynomial openings at verifier-chosen random points. Existing amortization techniques rely on shared verifier randomness, for example, to batch openings at a common evaluation point. However, under the standard Fiat–Shamir transform, independently verifiable proofs derive challenges from separate transcripts, preventing such amortization.

We address this obstacle through a *multi-verifier Fiat–Shamir transform* that correlates verifier challenges across independently verifiable proofs while preserving locality. We further introduce *promise local folding schemes*, which defer polynomial-constraint checks generated during folding and amortize them later. Together, these techniques provide a generic framework for amortization under local verification.

We apply this framework to the witness-independent component arising in R1CS- and CCS-based SNARK provers, which reduces to bivariate polynomial evaluation claims over the public constraint matrices. This component accounts for a substantial portion of the concrete proving cost. Applying our techniques to these claims reduces the server's cryptographic cost for this component from $\mathcal{O}(K \cdot s)$ to $\mathcal{O}(K \log K + s)$ group operations, where s denotes the sparsity of the public constraint matrices, with each verifier performing only $\mathcal{O}(\log K)$ cryptographic work and requiring no inter-verifier communication. Because the amortized component depends only on the public circuit description, our framework composes cleanly with collaborative zk-SNARK protocols, which target the complementary witness-dependent component.

Contents

1 Introduction **1**
- 1.1 Our Contributions . . . 2
- 1.2 Related Work . . . 4

2 Techniques Overview **5**
- 2.1 Reductions of Knowledge . . . 6
- 2.2 A Multi-Verifier Fiat-Shamir Transform . . . 7
- 2.3 Local Folding Schemes . . . 8
- 2.4 Compiling Reductions of Knowledge to Local Folding Schemes . . . 9
- 2.5 Application: Amortized Delegation of Bivariate Polynomial Evaluations . . . 12

3 Preliminaries **15**
- 3.1 Polynomial Commitments . . . 16
- 3.2 Merkle Trees . . . 16

4 Reductions of Knowledge **17**
- 4.1 (Public Coin) Reduction of Knowledge . . . 17
- 4.2 Non-interactive Reduction of Knowledge in the RO model . . . 19
- 4.3 Fiat-Shamir transform for Reductions of Knowledge . . . 20
- 4.4 Sequential and Parallel Composition of Reductions of Knowledge . . . 20

5 A Transformation for Amortized Multi-Verifier Reductions of Knowledge **23**

6 Promise Local Folding Schemes **28**

7 Local Folding Schemes Compiler **29**
- 7.1 $\mathcal{R}_{\mathsf{PCC}}$-Promise Local Folding Scheme Compiler . . . 30
- 7.2 From $\mathcal{R}_{\mathsf{PCC}}$-Promise to Plain Local Folding Schemes . . . 35

8 A Local Folding Scheme for Bivariate Polynomial Evaluation **38**
- 8.1 A Reduction of Knowledge for (Partially) Folding $\mathcal{R}_{\mathrm{M}}$ Statements . . . 39
- 8.2 A Local Folding Scheme for Bivariate Polynomial Evaluation . . . 42

A Deferred Proofs **52**
- A.1 Proof of Thm. 4.4.1 . . . 52

1 Introduction

Succinct non-interactive arguments of knowledge (SNARKs) [Kil92; Mic94] have become a central cryptographic primitive, and an extensive body of work targets their efficient construction. A complementary line of research studies *SNARK proof delegation*: rather than having the witness-holder perform the (often expensive) proof computation directly, proving is outsourced to one or more dedicated servers, optionally with privacy guarantees over the witness. Despite the breadth of this literature, almost all prior work studies the delegation of a *single* proof at a time. A natural extension is a single proving infrastructure serving K independent clients, each delegating the proof of a different statement. We refer to this as the *multi-verifier* setting: each verifier should be able to verify its own statement and proof independently, without assuming any coordination or communication with verifiers of other statements. This raises the following question:

Can a SNARK prover serving K independent verifiers amortize cryptographic work across all K proofs?

This setting arises naturally in proof-as-a-service deployments, where a shared proving infrastructure serves many independent clients. Depending on the application, the proving service may either learn the witness directly or interact with clients through privacy-preserving protocols such as MPC- or FHE-based delegation [SVV16; WZC+18; OB22; CLMZ23; GGJ+23; ABVT26; GGJ+25; LZW+25; HMW+25; GBK+25; ACGS24; ZWL+25].

The witness-independent bottleneck. Fully amortizing private SNARK proving across independent clients appears difficult, since witness-dependent computation typically requires MPC or other privacy-preserving machinery to hide the client's witness from the server. Rather than amortizing this privacy-preserving layer directly, we focus on a witness-independent cryptographic component determined entirely by the public circuit description and therefore amenable to amortization independently of the underlying privacy mechanism.

SNARKs derived from R1CS [CHM+20] or its generalization CCS [STW23] are particularly amenable to this approach: their arithmetization cleanly separates the public circuit description from the witness assignment, so a substantial portion of the prover's cryptographic work depends only on the circuit. Concretely, the constraint system is encoded by a small set of matrices $\mathbf{M}_1, \ldots, \mathbf{M}_\ell$ (e.g. $\ell = 3$ for R1CS), which are represented as bivariate polynomials $P_i(X, Y)$ (e.g., over a Lagrange basis, $P(X, Y) = \sum_{a,b} M_{a,b}\, \lambda_a(X)\lambda_b(Y)$). The SNARK prover must, as part of any proof, convince the verifier of evaluations of the form $y_i = P_i(\alpha, \beta)$ at challenges (α, β) derived during the protocol. Crucially, the P_i depend only on the public circuit description and not on the witness.

This *witness-independent* component is the only part of the prover's work whose cost scales with the sparsity s of the constraint system (the number of non-zero entries of the $\mathbf{M}_i$) rather than only with the number of multiplication gates[1]. In known instantiations [CHM+20; CFF+21; RZ21], this component requires $\mathcal{O}(s \log s)$ field operations dominated by $\mathcal{O}(s)$-sized multi-scalar multiplications over a pairing-friendly group; it also determines the size of the structured reference string.

The naive cost and the limits of existing techniques. Handling this witness-independent polynomial-evaluation component independently for each client incurs $\mathcal{O}(K \cdot s)$ cryptographic operations on the server side. Several lines of work target related problems, but each falls short in our setting: *Proof aggregation* [BMM+21; GMN22; ABST23; GPPS26; NPR26] and *Single-Instruction, Multiple Data* (SIMD) constructions [Tha13; WJB+17; WTs+18; TKPS22; ZLW+21] compress K SNARK proofs into

[1] While asymptotically $s = O(n)$, sparse circuits in practice often satisfy $s \approx 1.5n$–$3n$, making this component a substantial part of the prover's concrete cost.

	With Our Amortization	Without Amortization
Prover Time	$(K \cdot s + s \cdot \log s)\ \mathbb{F} + (K \cdot \log K + s)\ \mathbb{G}$	$K \cdot s \cdot \log s\ \mathbb{F} + K \cdot s\ \mathbb{G}$
Verifier Time	$\log K \cdot (\mathbb{F} + \mathbb{G} + \rho) + 1 \cdot \mathsf{P}$	$1 \cdot (\mathbb{F} + \mathbb{G} + \rho + \mathsf{P})$
Proof Size	$\lambda \cdot \log K$	λ

Table 1: Asymptotic comparison of the witness-independent polynomial-evaluation component when serving K independent clients, measured in group ($\mathbb{G}$), field ($\mathbb{F}$), random oracle queries (ρ), and pairing (P) operations. Here s denotes the sparsity of the underlying constraint system and λ the security parameter. The non-amortized costs correspond to standard R1CS/CCS-based SNARK constructions such as Marlin [CHM+20].

one with sublinear verification overhead, but assume a *single* verifier that has access to all K statements. This is incompatible with K mutually-unaware clients, each verifying only its own proof.

Collaborative zk-SNARKs [SVV16; WZC+18; OB22; CLMZ23; GGJ+23; ABVT26; GGJ+25; LZW+25; HMW+25] target a different axis (distributing the proving task across servers holding witness shares for privacy) and do not amortize cryptographic work across distinct proofs. The authors of [HMW+25] observe that the witness-independent part need not be proved inside the MPC and can be handled separately without harming privacy, but they do not propose a way to amortize this part across multiple proofs.

Folding with local verification was introduced by Ràfols and Zacharakis [RZ23], who present a compiler that transforms a folding scheme into a locally verifiable protocol by arranging the computation as a tree of folding rounds. A verifier that checks a statement at a leaf completely verifies the correctness of the folding steps in the nodes along the folding tree path. This naturally covers folding schemes such as Nova [KST22], where the correctness of a folding step is verified through linear operations on commitments. For more general relations, however, fully verifying each folding step succinctly may itself require cryptographic data that are expensive to produce on the prover side (such as additional commitments and polynomial commitment openings), or concretely expensive verification operations (such as bilinear group operations). Although such folding schemes can still be used with the compiler, this concretely expensive proving and verification process must be repeated at every node, largely eliminating the cryptographic savings of delegation and amortization.

Correlated randomness under local verification. For these more complex relations, a more natural approach is to defer part of the verification and amortize it across the K executions, rather than fully verifying every folding step independently. Such an approach fundamentally relies on correlated verifier randomness; for example, on multiple proofs sharing the same random evaluation point. Two obstacles, however, remain. First, the existing framework for local verification does not cover folding schemes in which part of the verification is intentionally deferred. Second, standard Fiat–Shamir compilation generates verifier challenges independently across proofs, preventing the required correlation. Achieving such shared randomness while preserving local verification is therefore the main conceptual challenge addressed in this work. More broadly, this problem may be of independent interest in distributed proving settings where verifiers remain mutually unaware.

1.1 Our Contributions

We give a framework for amortizing proof components across mutually unaware verifiers. It enables multiple non-interactive proofs to share verifier randomness while preserving local verification, thereby unlocking amortization techniques normally restricted to the single-verifier setting. The framework applies to protocols whose verification reduces to polynomial-constraint checks at a random evaluation point, a structure common to SNARKs derived from polynomial IOPs in the R1CS/CCS fam-

ily [CHM+20; CFF+21; RZ21; STW23]. As a main application, we instantiate the framework for the witness-independent polynomial-evaluation component of R1CS/CCS-based SNARKs, reducing the prover's cryptographic cost from $\mathcal{O}(K \cdot s)$ to $\mathcal{O}(K \log K + s)$ group operations, with only $\mathcal{O}(\log K)$ additional work per verifier and no inter-verifier communication. Our analysis is carried out in the reduction-of-knowledge framework of Kothapalli and Parno [KP23]. Concretely, our contributions are as follows:

(1) A multi-verifier Fiat–Shamir transform. In Sec. 5 we analyze a variant of the Fiat–Shamir transform [FS87] in which K parallel non-interactive proofs are forced to share the same first verifier challenge, derived as a random-oracle query to a Merkle commitment of all K statements. The compiled protocol involves no communication between verifiers, and we analyze knowledge soundness from the perspective of a single verifier against a malicious prover.

The shared first challenge is precisely what enables amortization under local verification: when verifiers agree on a single random evaluation point, polynomial-constraint sub-checks across the K executions can be folded by linearity of the commitment scheme, so that a single polynomial-commitment opening (plus $\mathcal{O}(\log K)$ work per verifier) suffices. Variants of this construction appeared implicitly in [YLF+22; ZXH+22] without a general security analysis; we give a rigorous treatment in the reduction-of-knowledge framework. The transformation is itself a generic tool, applicable to any multi-verifier protocol with shared first-round randomness.

(2) From polynomial constraints to amortized local folding. The shared verifier randomness produced by our multi-verifier Fiat–Shamir transform enables polynomial-constraint checks arising inside recursive folding schemes to be amortized across executions. In Sec. 7 we present a generic compiler that derives an amortized protocol—formalized as a *local folding scheme*—from any public-coin reduction of knowledge $\mathcal{R}^2 \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}$, i.e., a folding scheme for $\mathcal{R}$ whose internal verification reduces to polynomial-constraint sub-checks ($\mathcal{R}_{\mathsf{PCC}}$).

To avoid paying for these sub-checks at every recursive folding step, in Sec. 6 we introduce a new primitive, *promise local folding schemes*, which allows polynomial-constraint checks arising during the recursive K-to-1 folding to be deferred and amortized later. This extends the folding scheme with selective verification of [RZ23] to support deferred polynomial-constraint checks while maintaining a modular reduction-of-knowledge analysis under recursive composition.

(3) Amortized delegation of bivariate polynomial evaluations. In Sec. 8 we instantiate the framework for the bottleneck identified above. For K clients each holding an evaluation claim $y_i = P(\alpha_i, \beta_i)$ for a fixed public bivariate polynomial P of sparsity s (e.g., encoding a common R1CS/CCS instance shape), our construction delegates these evaluations with total cryptographic cost $\mathcal{O}(K \log K + s)$ group operations, compared with $\mathcal{O}(K \cdot s)$ for the naive approach (see Table 1).

To achieve this, we develop techniques for implicitly committing to and homomorphically combining the sparse verifier-challenge polynomials arising in the protocol, thereby avoiding the $\mathcal{O}(s)$-sized MSM that would otherwise dominate the prover cost.

(4) Composition with private SNARK delegation. An R1CS/CCS-based SNARK prover naturally decomposes into a witness-independent, sparsity-dependent component—which our framework amortizes—and a witness-dependent component whose structure is essentially a uniform sumcheck. This decomposition enables clean composition with existing protocols for private SNARK proof delegation [SVV16; WZC+18; OB22; CLMZ23; GGJ+23; ABVT26; GGJ+25; LZW+25; HMW+25]: the witness-independent component can be factored out of the MPC layer entirely and amortized by an untrusted server, while the remaining witness-dependent sumcheck is handled by existing distributed protocols.

To our knowledge, this is the first amortization of SNARK prover work across distinct clients that does not require the amortizing party to access witnesses. We note that clients who do not wish to wait for a full batch can always fall back to a standard non-amortized proof; the batch size K is a tunable latency-cost tradeoff controlled by the service provider.

1.2 Related Work

Reductions of Knowledge. Kothapalli and Parno introduced reductions of knowledge [KP23] as a generalization of folding [KST22] and (atomic/split) accumulation schemes [BGH19; BCMS20; BCL+21]. Their framework allows one to express complex constructions in a simple and elegant manner. The work of Bünz et al. [BMNW25b] extends this notion to the Interactive Oracle Proofs setting [BCS16] as Interactive Oracle Reductions. We make extensive use of the framework of [KP23] in our work, which enables us to reason about and describe complex constructions cleanly and succinctly.

Recursive Proof Composition. The work of [BGH19; BCMS20] initiated a line of research [BCL+21; KST22; KS24; BC23; EG23; BC25a; BC25b; BMNW25a; BMNW25b] that constructs proof-carrying data [CT10] and incrementally verifiable computation [Val08] by postponing proving parts of the computation and verifiably "combining" the postponed parts. We use the same insight to improve amortized efficiency in the multi-verifier setting: (1) we combine statements from different verifiers instead of proving them, and (2) we postpone proving that this was done correctly, to further amortize costs. These constructions, however, are designed for a *single* verifier setting: typically an IVC chain or an accumulator instance where the verifier participates in the folding sequence and sees each folded statement together with the running accumulator state. Our multi-verifier setting departs from this assumption—no single party sees all K statements, and the local verification requirement means each client must accept or reject from its own statement and its own proof alone.

Proof Aggregation and SIMD. A separate line of work targets *proof aggregation*, where K SNARK proofs are compressed into a single proof verifiable in sublinear time. Examples include IPP [BMM+21], SnarkPack [GMN22], and [NPR26] targeting Groth16 [Gro16], aPlonK [ABST23] and GAPP [GPPS26] for PlonK-style systems. These constructions assume a single verifier that sees all K statements and proofs, and are therefore inapplicable to our multi-verifier setting. A similar limitation applies to *SIMD* constructions [Tha13; WJB+17; WTs+18; TKPS22; ZLW+21] which build schemes for data-parallel versions of the GKR protocol [GKR08] by leveraging the decomposition properties of the multivariate sumcheck [LFKN92].

The Multi-Verifier Setting. This setting has been considered mainly for specific problems, most notably vector and polynomial commitment openings [SCP+22; CNR+22; ZXH+22; TCZ+20; YLF+22]. A more general treatment was given in [RZ23], where the authors introduce a compiler capturing many-to-one reductions with selective verification derived from folding schemes [KST22]. Our work generalizes their definitions and introduces new techniques that lead to more efficient constructions, leveraging the fact that the amortization correctness proof can itself be postponed and further amortized. The works [YLF+22; ZXH+22] use the same Merkle-tree-based Fiat-Shamir technique that we do, but the former does not provide a security analysis and the latter provides an ad-hoc proof. In particular, the latter introduces a Fiat–Shamir-style transform for one-to-many GKR/Virgo-style arguments in which verifier challenges are correlated using Merkle authentication paths. Their analysis views the Merkle reconstruction procedure as additional "dummy" rounds and proves soundness via a protocol-specific state-restoration argument. Our work adopts the same high-level viewpoint but differs in both scope and abstraction. Rather than analyzing a specific protocol directly, we formulate the mechanism generically in the reduction-of-knowledge framework of [KP23]. The key observation is that the dummy Merkle rounds themselves define an augmented reduction of knowledge, allowing the resulting protocol to be

viewed as a standard Fiat–Shamir transform applied to this augmented protocol. Soundness then follows from the security of the standard Fiat–Shamir transform together with the composition properties of reductions of knowledge. This yields a modular and conceptually cleaner analysis that applies generically to public-coin protocols and composes naturally with our local-folding framework.

SNARK proof distribution and delegation. A line of work considers parallelizing SNARK proof computation across servers [WZC+18; LXZ+24; LZL+25]. A related setting is collaborative zk-SNARKs [OB22], where different parties hold different witness pieces and jointly create a single SNARK proof. Both settings have received attention recently [SVV16; LXZ+24; CLMZ23; GGJ+23; LZW+25; GGJ+25; HMW+25]. These works are orthogonal to ours, in the sense that the witness-independent part can be stripped from their constructions, leading to simplifications and efficiency improvements. With the exception of [HMW+25], such optimizations are not considered; [HMW+25] considers separating the witness-independent component but does not amortize it across distinct proofs. Hence, our cross-proof amortization complements what they do.

Scope: PlonK-style and lookup-based systems. Our framework exploits a structural property specific to R1CS- and CCS-based arithmetizations: the prover's work factors cleanly into a witness-dependent component and a witness-independent component whose cost is dominated by the bivariate polynomial evaluation of the public constraint matrices. PlonK-style [GWC19] and lookup-based [GW20] systems do not exhibit this decomposition: their constraint checks interleave public preprocessed polynomials with witness polynomials at a common evaluation point, and the prover's substantial cryptographic costs—wiring and quotient polynomial commitments, the permutation grand product, and lookup-table arguments—all depend on the witness. Extending our techniques to these settings is an interesting open problem requiring different approaches.

2 Techniques Overview

Our goal is to amortize prover work across K independently verifiable proofs. In the multi-verifier setting considered in this work, each verifier: (i) verifies only its own statement and proof, (ii) does not communicate with the other verifiers, and (iii) may not even know that the other executions exist. Thus the challenge is not merely to aggregate many proofs, but to amortize prover work while preserving local verification.

A natural starting point is locally verifiable folding [RZ23]. There, the prover repeatedly folds pairs of instances into one, while each verifier checks only a logarithmic-size path in the folding tree, similar to checking a Merkle authentication path. In familiar folding schemes, such as Nova-style folding [KST22], each folding step preserves essentially the same relation: the verifier checks that the parent instance is the prescribed linear combination of the two child instances. For example, if the children contain witness commitments $\mathsf{Comm}(\mathbf{z}_0; r_0)$ and $\mathsf{Comm}(\mathbf{z}_1; r_1)$, then the verifier checks consistency of a folded commitment of the form

$$\mathsf{Comm}(\mathbf{z}^*; r^*) = \mathsf{Comm}(\mathbf{z}_0; r_0) + \gamma \cdot \mathsf{Comm}(\mathbf{z}_1; r_1),$$

together with analogous checks for the remaining folded quantities. The key point is that these checks are linear in the committed data and can therefore be performed locally using the homomorphic structure of the commitment scheme.

Our setting differs from this. In many folding constructions arising from IOP-based proof systems, the folded relation no longer preserves only such linear consistency conditions. Instead, folding generates polynomial consistency checks between committed polynomials. These checks are generally nonlinear,

so they cannot be verified merely by taking linear combinations of commitments. They must eventually be reduced to polynomial openings or pairing-based checks at verifier-chosen random points.

This creates a choice. One could perform these polynomial checks immediately at every node of the folding tree. This approach is prohibitively expensive: the verifier would pay for polynomial-opening checks along its entire path, and the prover would lose much of the intended amortization.

Alternatively, one can defer the polynomial checks produced during folding and prove them together only at the end. However, this changes the nature of the folding process itself. In Nova-style folding, once the verifier checks the local linear relations between the child instances and the folded instance, knowledge of a witness for the folded instance immediately implies knowledge of witnesses for the two children.

In our setting, this is no longer true. If the polynomial consistency checks are deferred, then at the moment the folded instance is created, the verifier has not yet checked that the reduction step was performed correctly. Instead, correctness of the reduction holds only conditioned on additional polynomial constraints that remain unverified until the final proof is produced. Thus each node along the verifier's local path contributes not only a new folded instance, but also new deferred polynomial obligations certifying that the corresponding folding step was valid.

However, deferring these checks is useful only if they can themselves be amortized efficiently. This requires many polynomial openings to occur at the same evaluation point, which standard Fiat–Shamir does not provide in the multi-verifier setting: each verifier derives its randomness independently from its own local transcript. Thus the central observation of this work is that amortization under local verification requires correlated verifier randomness.

To enable this strategy, we introduce two main building blocks. First, we introduce a *multi-verifier Fiat–Shamir transform* that correlates verifier challenges across independently verifiable proofs while preserving local verification. This allows the deferred polynomial checks arising throughout the folding tree to be reduced to same-point openings and amortized together. Second, we introduce *promise local folding schemes*, which formalize folding processes in which correctness of each reduction step depends on additional polynomial obligations that are verified only later. We then instantiate these techniques for the witness-independent polynomial relations arising in R1CS/CCS-based SNARKs.

We begin in Sec. 2.1 with reductions of knowledge [KP23], and in Sec. 2.2 introduce a multi-verifier Fiat–Shamir transform that restores the shared randomness needed for amortization under local verification. In Sec. 2.3 and Sec. 2.4, we define plain and promise local folding schemes and construct them from reductions of knowledge. Finally, in Sec. 2.5 we apply these techniques to the witness-independent polynomial relations arising in R1CS/CCS-based SNARKs.

2.1 Reductions of Knowledge

To formalize our results, we make extensive use of *reductions of knowledge* [KP23]. A reduction of knowledge $\Pi : \mathcal{R} \to \mathcal{R}'$ is an interactive protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ that reduces knowledge of w s.t. $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$ to knowledge of w' s.t. $(\mathsf{x}', \mathsf{w}') \in \mathcal{R}'$, where x', w' are the respective outputs of $\mathcal{V}$ and $\mathcal{P}$ after their interaction. As shown in [KP23], reductions of knowledge:

1. compose sequentially, in the sense that when $\mathcal{P}$ and $\mathcal{V}$ engage in an execution $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$ and then an execution of $\Pi_2 : \mathcal{R}_2 \to \mathcal{R}_3$ the combined protocol, denoted $\Pi_2 \circ \Pi_1$, is a reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_3$,
2. compose in parallel, in the sense that when $\mathcal{P}$ and $\mathcal{V}$ engage in two parallel executions of $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$ and $\Pi_2 : \mathcal{R}_3 \to \mathcal{R}_4$ the combined protocol, denoted $\Pi_1 \times \Pi_2$, is a reduction of

knowledge from $\mathcal{R}_1 \times \mathcal{R}_3$ to $\mathcal{R}_2 \times \mathcal{R}_4$, where

$$\mathcal{R} \times \mathcal{R}' = \{((\mathsf{x}, \mathsf{w}), (\mathsf{x}', \mathsf{w}')) \,|(\mathsf{x}, \mathsf{w}) \in \mathcal{R} \text{ and } (\mathsf{x}', \mathsf{w}') \in \mathcal{R}'\}$$

If the reductions are public coin they can be made non-interactive in the random oracle model; we denote the resulting protocol as $\mathsf{NIRoK} = \mathsf{FS}(\Pi)$. We use the transformation where a different random oracle is used for each round of Π. As demonstrated in [KP23], many constructions can be expressed in this framework with a significantly simplified analysis.

We extend reductions of knowledge by allowing the prover $\mathcal{P}$ to also have an auxiliary input aux. In our constructions this auxiliary input corresponds to transcripts the prover has by interacting with other verifiers, aiming to amortize the proving cost. Knowledge soundness is defined w.r.t. adversarially sampled auxiliary input.

To help in expressing complex protocols as a series of compositions of reductions of knowledge we introduce (1) the identity reduction of knowledge $\mathsf{ID} : \mathcal{R} \to \mathcal{R}$ where $\mathcal{P}$ and $\mathcal{V}$ simply output their corresponding inputs without engaging in interaction and (2) the auxiliary reduction of knowledge $\mathsf{AUX}_{\mathcal{R},\mathcal{R}'} : \mathcal{R} \to \mathcal{R} \times \mathcal{R}'$ in which $\mathcal{P}(\mathsf{w})$ sends a statement x' of $\mathcal{R}'$ to $\mathcal{V}(\mathsf{x})$ (which, along with the corresponding witness w', is part of its auxiliary input) and the verifier outputs the pair $(\mathsf{x}, \mathsf{x}')$ and similarly the prover outputs $(\mathsf{w}, \mathsf{w}')$. Both are trivially reductions of knowledge.

2.2 A Multi-Verifier Fiat-Shamir Transform

Let $K = 2^\kappa$ be a constant[2] and consider $\mathcal{P}$ engaging in parallel executions of a reduction of knowledge $\Pi : \mathcal{R} \to \mathcal{R}'$ with verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_K$. We propose an alternative to the standard Fiat-Shamir transform which derives a non-interactive protocol from Π where the first message among all verifiers is the same. The core idea is to derive the first message as $\rho(\mathsf{h})$ where ρ is a random oracle and h is the root of a Merkle tree of all verifiers' inputs $\mathsf{x}_1, \ldots, \mathsf{x}_K$. Each verifier $\mathcal{V}_i$ can recompute the first message via $\log K$ invocations of the random oracle, assuming $\mathcal{P}$ shares the corresponding sibling nodes.

Intuitively, this preserves soundness because from the perspective of the i-th verifier the first message computed as $\rho(\mathsf{h})$ is defined after the statement x_i, since the latter is in the hash chain resulting in h. Directly analyzing this correlated Fiat–Shamir transformation, however, would require reasoning about the interaction between multiple transcripts and shared random-oracle queries.

Instead, we reduce the analysis to a standard Fiat–Shamir argument. The key observation is that reconstructing the Merkle authentication path can itself be viewed as additional public-coin rounds preceding the original protocol. This allows us to reinterpret the correlated-randomness transform as an ordinary Fiat–Shamir transform applied to an augmented reduction of knowledge.

Consider the point of view of a single verifier $\mathcal{V}_i$ and assume the first message of Π is sent by $\mathcal{V}_i$. Our goal is to prove that from the perspective of $\mathcal{V}_i$ the resulting protocol is knowledge sound. We first define a transformation $\mathsf{MT}(\Pi, \kappa)$ that returns a public coin reduction of knowledge $\widetilde{\Pi} : \mathcal{R} \to \mathcal{R}'$. We define $\widetilde{\Pi}$ where $\mathcal{P}(\mathsf{w}, \mathsf{aux} = \gamma_1, \ldots, \gamma_\kappa)$ and $\mathcal{V}(\mathsf{x})$ interact as follows:

1. $\mathcal{P}$ and $\mathcal{V}$ engage in $\kappa + 1$ rounds where in the first round the verifier sends a random γ'_0 and in the j-th round $\mathcal{P}$ sends γ_j and $\mathcal{V}$ replies with γ'_j.
2. After the $\kappa + 1$-th round, $\mathcal{P}$ and $\mathcal{V}$ engage in an execution of Π.

[2]Throughout this work we consider a prover interacting with K verifiers, where K is treated as constant with respect to the security parameter λ. In practice, however, K is a tunable batching parameter controlling how many proofs are amortized together, and our efficiency bounds explicitly describe the dependence on K. Our intended regime is that the size of the computations being proven (e.g., the circuit sparsity s) is much larger than K.

The resulting protocol is trivially knowledge sound regardless of the auxiliary input used. Indeed, the messages exchanged during the first $\kappa + 1$ rounds are ignored during the protocol in the sense that no verification check is imposed on them. Their sole purpose is to encode the reconstruction of a Merkle path into the transcript structure itself.

Next, consider the Fiat-Shamir protocol derived as $\mathsf{FS}(\widetilde{\Pi})$. In this protocol, the first message sent by $\mathcal{V}$ is computed as $\gamma'_0 \leftarrow \rho_0(\mathsf{x})$ and the j-th message as $\rho_j(\gamma'_{j-1}, \gamma_j)$. Viewing the prover's message γ_j as the j-th sibling node of a Merkle tree and $\gamma'_j = \rho_j(\gamma'_{j-1}, \gamma_j)$ as the next node, the first $\kappa + 1$ rounds of the transformed protocol correspond to reconstructing the leftmost path of the Merkle tree with leaf x_i. Further, the resulting protocol is knowledge sound (with some additional error due to the $\kappa + 1$ extra rounds) since it is a (standard) Fiat-Shamir transformation of a reduction of knowledge.

We next derive our modified FS transformation $\mathsf{FS}_{\mathsf{MT}}(\Pi, \kappa)$. The transformation works as follows:

- The non-interactive prover, on input $\mathsf{x}_1, \ldots, \mathsf{x}_K$, first computes a Merkle tree of all the statements. Let h be the root of the Merkle tree. Then it runs K executions of $\mathsf{FS}(\Pi)$, in all of which it uses as the first message $\rho_{\kappa+1}(\mathsf{h})$. In the j-th proof, it also adds the sibling nodes of the j-th Merkle path leading to h.
- The non-interactive verifier, on input i, x_i and proof $\pi_i = (\gamma_1, \ldots, \gamma_\kappa, \pi_\Pi)$ uses the random oracles to compute h by computing $\gamma'_0 = \rho_0(\mathsf{x}_i)$ and γ'_j as either $\rho_j(\gamma'_{j-1}, \gamma_j)$ or $\rho_j(\gamma_j, \gamma'_{j-1})$ depending on whether the sibling is a left or a right one. It then verifies π_Π normally, except that it uses as the first message corresponding to the first round of Π the value $\rho_{\kappa+1}(\mathsf{h})$.

Theorem 2.2.1 (Thm. 5.2 informal)**.** *Let $K = 2^\kappa$ be a constant, $i \in \{1, \ldots, K\}$ and $\Pi : \mathcal{R} \to \mathcal{R}'$ be a reduction of knowledge. Consider the algorithms Prove_i, Verify_i derived by applying Transformation $\mathsf{FS}_{\mathsf{MT}}(\Pi, \kappa)$ and using the i-th proof. Then $(\mathsf{Prove}_i, \mathsf{Verify}_i)$ is a non-interactive reduction of knowledge $\mathsf{NIRoK}_i : \mathcal{R} \to \mathcal{R}'$ in the random oracle model.*

To argue about knowledge soundness, we show that for all $i \in \{1, \ldots, K\}$ and all auxiliary inputs, the verification of $\mathsf{FS}_{\mathsf{MT}}(\Pi)$ is equivalent to the verification of $\mathsf{FS}(\Pi)$. Indeed, the only difference between the two is that in rounds $1, \ldots, \kappa$ the inputs to $\rho_i(\cdot, \cdot)$ are in some cases swapped, depending on the value of i. Setting $\rho'_i(l, r) = \rho_i(r, l)$ if the inputs are swapped and $\rho'_i(l, r) = \rho_i(l, r)$ otherwise, the verifier of $\mathsf{FS}_{\mathsf{MT}}(\Pi)$ using random oracles $\boldsymbol{\rho}$ is identical to the verifier of $\mathsf{FS}(\Pi)$ using random oracles $\boldsymbol{\rho}'$. We conclude by noting that $\boldsymbol{\rho}, \boldsymbol{\rho}'$ are distributed identically.

2.3 Local Folding Schemes

In many protocols derived from Polynomial Interactive Oracle Proofs, verification reduces to checking polynomial identities of the form $Q(X, P_1(X), \ldots, P_\ell(X)) = 0$, where the polynomials $P_i(X)$ are committed using a polynomial commitment scheme. Verifying such identities eventually requires polynomial-opening checks at verifier-chosen random points.

Our construction relies on the observation that these polynomial checks need not be performed immediately. Instead, the prover can defer them and only "promise" that the corresponding polynomial relations hold. The deferred checks generated across many verifiers can then be amortized together later. This motivates a folding abstraction in which the correctness of a folding step may depend on additional deferred verification obligations.

In a plain *local folding scheme* a prover runs $\mathsf{LProve}(\{\mathsf{x}_i, \mathsf{w}_i\}_{i=1}^K)$ and outputs: (1) a folded statement-witness pair (x, w), which belongs to $\mathcal{R}$ in the honest case, and (2) K local proofs π_i. The scheme is knowledge sound in the following sense: if the prover knows w such that $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$ and the corresponding algorithm $\mathsf{LVerify}(\mathsf{x}_i, \mathsf{x}, \pi_i)$ verifies, then the prover must know a witness for x_i. Crucially, π_i should

be sublinear in K, capturing that verifier i does not even read the other statements used by LProve. In the plain case, the definition is essentially identical to Folding Schemes with Selective Verification [RZ23], except that verification and knowledge soundness are formulated only from the perspective of a single verifier.

We then relax the definition and introduce *Promise Local Folding Schemes* to capture the idea of deferring part of the verification by allowing LProve to additionally output K promises. These correspond to statements $(\mathsf{x}'_i, \mathsf{w}'_i) \in \mathcal{R}'$ for some relation $\mathcal{R}'$. The verifier additionally receives the promise statement x'_i, and knowledge soundness now requires that if: (1) the prover knows w such that $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$, (2) $\mathsf{LVerify}(\mathsf{x}_i, \mathsf{x}, \mathsf{x}'_i, \pi_i)$ verifies, and (3) the prover knows w'_i such that $(\mathsf{x}'_i, \mathsf{w}'_i) \in \mathcal{R}'$, then the prover must know a witness w_i such that $(\mathsf{x}_i, \mathsf{w}_i) \in \mathcal{R}$. We call the resulting primitive a $\mathcal{R}'$-Promise Local Folding Scheme.

2.4 Compiling Reductions of Knowledge to Local Folding Schemes

Relations over committed polynomials. Define the polynomial committed constraint relation $\mathcal{R}_{\mathsf{PCC}}$ and the polynomial commitment opening language as

$$\mathcal{R}_{\mathsf{PCC}} = \{(\mathbf{c}, Q), \mathbf{P}(X) \mid \mathsf{c}_j \text{ is a commitment to } P_j(X) \text{ and } Q(X, \mathbf{P}(X)) = 0\}$$

$$\mathcal{R}_{\mathsf{PCO}} = \{(\mathsf{c}, x, y), P(X) \mid \mathsf{c} \text{ is a commitment to } P(X) \text{ and } P(x) = y\}$$

where we denote $\boldsymbol{P}(X) = (P_1(X), \ldots, P_\ell(X))$. When using homomorphic polynomial commitments, there exists a well known reduction of knowledge $\Pi_{\mathsf{PC}} : \mathcal{R}_{\mathsf{PCC}} \to \mathcal{R}_{\mathsf{PCO}}$. The reduction works as follows:

1. $\mathcal{V}$ sends a random point $x \leftarrow \mathbb{F}$
2. $\mathcal{P}$ replies with $\mathbf{y} = \mathbf{P}(x)$
3. $\mathcal{V}$ sends a random point $\alpha \leftarrow \mathbb{F}$
4. $\mathcal{P}$ and $\mathcal{V}$ output $\sum \alpha^i P_i(X)$ and $(\sum \alpha^i \mathsf{c}_i, x, \sum \alpha^i y_i)$ respectively.
5. $\mathcal{V}$ checks that $Q(x, \mathbf{y}) = 0$

Essentially, the prover and verifier engage in a randomized test over the constraints at a random point x. Then the problem reduces to proving that the given polynomials open to the claimed evaluations at x. For the latter, the homomorphic properties of the polynomial commitment scheme are leveraged to reduce to a single polynomial commitment opening.

$\mathcal{R}^{\kappa}_{\mathsf{PCC}}$-Promise Local Folding scheme from $\mathcal{R} \times \mathcal{R} \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}$. Let $\mathsf{NIRoK} : \mathcal{R} \times \mathcal{R} \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}$ be a non-interactive reduction of knowledge. Intuitively, NIRoK reduces two $\mathcal{R}$-statements to one while outputting the deferred polynomial commitment checks as a $\mathcal{R}_{\mathsf{PCC}}$-statement. We describe a compiler $\mathsf{L}_{\mathsf{PCC}}(\mathsf{NIRoK}, \kappa)$ to derive an $\mathcal{R}^{\kappa}_{\mathsf{PCC}}$-Promise Local Folding for $\mathcal{R}$ that amortizes the cost of proving 2^κ statements. It closely follows the one from [RZ23]. It labels the leaves of a binary tree with the 2^κ statements and traverses the tree bottom-up, applying NIRoK at each internal node.

- The prover proceeds in κ iterations:

  1st iteration: compute for all odd $i \in \{1, 3, \ldots, 2^\kappa - 1\}$

  $$(\mathsf{x}^*, \mathsf{x}_{\mathsf{PCC}}), (\mathsf{w}^*, \mathsf{w}_{\mathsf{PCC}}), \pi \leftarrow \mathsf{NIRoK.Reduce}((\mathsf{x}_i, \mathsf{x}_{i+1}), (\mathsf{w}_i, \mathsf{w}_{i+1}))$$

  and add $(\mathsf{x}_{i+1}, \mathsf{x}^*, \mathsf{x}_{\mathsf{PCC}}, \pi)$ to the i-th proof and $(\mathsf{x}_i, \mathsf{x}^*, \mathsf{x}_{\mathsf{PCC}}, \pi)$ to the $i+1$-th proof.

ℓ-th iteration: in the ℓ-th iteration, let $\mathsf{x}_1, \ldots \mathsf{x}_{2^{\kappa-\ell+1}}$ be the $\mathcal{R}$ statements produced in the previous iteration. Compute for all odd $j \in \left\{1, 3, \ldots, 2^{\kappa-\ell+1} - 1\right\}$

$$(\mathsf{x}^*, \mathsf{x}_{\mathsf{PCC}}), (\mathsf{w}^*, \mathsf{w}_{\mathsf{PCC}}), \pi \leftarrow \mathsf{NIRoK.Reduce}((\mathsf{x}_j, \mathsf{x}_{j+1}), (\mathsf{w}_j, \mathsf{w}_{j+1}))$$

and add $(\mathsf{x}_{j+1}, \mathsf{x}^*, \mathsf{x}_{\mathsf{PCC}}, \pi)$ to proofs that contained x_j and $(\mathsf{x}_j, \mathsf{x}^*, \mathsf{x}_{\mathsf{PCC}}, \pi)$ to proofs that contained x_{j+1}.

- The i-th verifier receives a proof containing κ tuples $(\mathsf{x}_i, \mathsf{x}^*, \mathsf{x}_{\mathsf{PCC},i}, \pi)$. It traverses the "path" verifying all the proofs along the way. Concretely:

1st iteration: Let x_{i+b} be the verifier's input statement. Also, let $\mathsf{x}_{i+1-b}, \mathsf{x}^*, \mathsf{x}_{\mathsf{PCC}}, \pi_i$ be the first tuple. Assert

$$\mathsf{NIRoK.Verify}((\mathsf{x}_i, \mathsf{x}_{i+1}), (\mathsf{x}^*, \mathsf{x}_{\mathsf{PCC}}), \pi) = 1.$$

ℓ-th iteration: Let x_{j+b} be the verifier's reduced statement x^* from the previous iteration. Also, let $\mathsf{x}_{j+1-b}, \mathsf{x}^*, \mathsf{x}_{\mathsf{PCC}}, \pi$ be the ℓ-th tuple of the proof. Assert

$$\mathsf{NIRoK.Verify}((\mathsf{x}_j, \mathsf{x}_{j+1}), (\mathsf{x}^*, \mathsf{x}_{\mathsf{PCC}}), \pi) = 1.$$

The promise of the i-th verifier is a statement $\mathsf{x}_{\mathsf{PCC}} \in \mathcal{R}_{\mathsf{PCC}}^{\kappa}$ which consists of the $\mathcal{R}_{\mathsf{PCC}}$ statements sent by the prover as part of the proof. Note that the "sibling" statement is always added to the proof so that each verifier can fold it with its own statement to derive the (common) folded statement.

Theorem 2.4.1 (Thm. 7.1.1 informal)**.** *Let $K = 2^{\kappa}$ be a constant and $\mathsf{NIRoK} : \mathcal{R} \times \mathcal{R} \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}$ be a non-interactive reduction of knowledge. Then construction $\mathsf{L}_{\mathsf{PCC}}(\mathsf{NIRoK}, \kappa)$ is a $\mathcal{R}_{\mathsf{PCC}}^{\kappa}$-promise Local Folding scheme for $\mathcal{R}$.*

We prove the above theorem using the reduction of knowledge framework. We argue as follows. Fix some index i and consider the local verification of the i-th statement. We claim that the verification procedure behaves as a reduction of knowledge. Indeed, one can view a single round of the verification as follows (we describe using the interactive case, the non-interactive is similar):

- $\mathcal{P}$ first gives x_{1-b} to $\mathcal{V}$ (holding some x_b)
- $\mathcal{P}$ and $\mathcal{V}$ reduce $(\mathsf{x}_0, \mathsf{x}_1)$ to $(\mathsf{x}^*, \mathsf{x}_{\mathsf{PCC}})$ using the reduction of knowledge.

Therefore, a single iteration behaves like a composition $\Pi \circ \mathsf{AUX}$. But then the verifier as a whole behaves as the sequential composition of the latter reduction with itself κ times; indeed the output of each iteration is the input of the next. Along the way the verifier also collects the $\mathcal{R}_{\mathsf{PCC}}$ statements and "pushes" them to the next iteration until it outputs all of them at the end as its promise statement. A slightly more complicated composition captures the above as well. Since the verifier as a whole behaves like a reduction of knowledge, knowledge soundness of the local folding scheme follows.

Plain Local Folding. Our ultimate goal is to construct plain local folding schemes from reductions of knowledge. The compiler $\mathsf{L}_{\mathsf{PCC}}$ is only an intermediate step. To obtain a plain local folding scheme, it remains to prove the $\mathcal{R}^{\kappa}_{\mathsf{PCC}}$ statements accumulated by each verifier.

To achieve amortization, we also need a local folding scheme for the promise statements from relation $\mathcal{R}_{\mathsf{PCC}}$. Doing this naively, however, would inflate the proving time, as the resulting polynomial commitment openings would be at different evaluation points. Instead, we first reduce the K $\mathcal{R}^{\kappa}_{\mathsf{PCC}}$ statements to polynomial commitment openings at a common point. Recall from Sec. 2.2 that $\mathsf{FS}_{\mathsf{MT}}(\Pi_{\mathsf{PC}}, \kappa)$ forces all executions of Π_{PC} to use the same first verifier message, which in Π_{PC} is the evaluation point x.

Let $\mathcal{R}_{\mathsf{PCO}(x)}$ denote the relation capturing polynomial commitment openings at a fixed evaluation point. This is the same as $\mathcal{R}_{\mathsf{PCO}}$ but with a "hardcoded" x, namely, $\mathcal{R}_{\mathsf{PCO}(x)} = \{(\mathsf{c}, y), p(X) \mid ((\mathsf{c}, x, y), p(X)) \in \mathcal{R}_{\mathsf{PCO}}\}$. Applying $\mathsf{FS}_{\mathsf{MT}}(\Pi_{\mathsf{PC}}, \kappa)$ therefore results in K statements $\mathsf{x}_{\mathsf{PCO}(x),i}$, one for each verifier. Each such statement combines the κ deferred polynomial-constraint checks accumulated along that verifier's path in the folding tree into a single polynomial commitment opening at the same evaluation point.

The prover applies $\mathsf{L}_{\mathsf{PCC}}(\mathsf{FS}(\Pi_{\mathsf{PCO}(x)}), \kappa)$ where $\Pi_{\mathsf{PCO}(x)} : \mathcal{R}^2_{\mathsf{PCO}(x)} \to \mathcal{R}_{\mathsf{PCO}(x)}$. This results in a single statement $\mathsf{x}_{\mathsf{PCO}}$ which the prover proves using the polynomial commitment and shares to all verifiers. We fully describe the compiler $\mathsf{L}(\mathsf{NIRoK})$ next.

- The prover on input $(\mathsf{x}_i, \mathsf{w}_i)$ does the following:
  1. Use $\mathsf{L}_{\mathsf{PCC}}(\mathsf{NIRoK}, \kappa)$ to reduce all statements to x^* and $\mathsf{x}_{\mathsf{PCC},i}$. Let $\pi_{1,i}$ be the resulting proofs.
  2. Use $\mathsf{FS}_{\mathsf{MT}}(\Pi_{\mathsf{PC}})$ to reduce each $\mathsf{x}_{\mathsf{PCC},i}$ to a statement $\mathsf{x}_{\mathsf{PCO}(x),i}$ where x is common for all i. Let $\pi_{2,i}$ be the corresponding proof for the i-th statement.
  3. Use $\mathsf{L}_{\mathsf{PCC}}(\mathsf{FS}(\Pi_{\mathsf{PCO}(x)}), \kappa)$ to reduce all $\mathsf{x}_{\mathsf{PCO}(x),i}$ to a single statement $\mathsf{x}_{\mathsf{PCO}(x)}$. Let $\pi_{3,i}$ be the resulting proofs.
  4. Prove the statement $\mathcal{R}_{\mathsf{PCO}(x)}$ using the polynomial commitment. Let π_{PC} be the corresponding proof.

  It sets as the i-th proof $\pi_i := ((\mathsf{x}_{\mathsf{PCC},i}, \pi_{1,i}), (\mathsf{x}_{\mathsf{PCO}(x),i}, \pi_{2,i}), (\mathsf{x}_{\mathsf{PCO}(x)}, \pi_{3,i}), \pi_{\mathsf{PC}})$.

- The verifier on input (x_i, π_i) does the following:
  1. Use $\mathsf{L}_{\mathsf{PCC}}(\mathsf{NIRoK}, \kappa)$ to verify the reduction of x_i to $(\mathsf{x}, \mathsf{x}_{\mathsf{PCC},i})$ using $\pi_{1,i}$.
  2. Use $\mathsf{FS}_{\mathsf{MT}}(\Pi_{\mathsf{PC}})$ to verify the reduction of $\mathsf{x}_{\mathsf{PCC},i}$ to $(\mathsf{x}_{\mathsf{PCO},i})$ using $\pi_{2,i}$. It then defines the relation $\mathcal{R}_{\mathsf{PCO}(x)}$ given the challenge x and converts $(\mathsf{x}_{\mathsf{PCO},i})$ to a statement $(\mathsf{x}_{\mathsf{PCO}(x),i})$ for the latter.
  3. Use $\mathsf{L}_{\mathsf{PCC}}(\mathsf{FS}(\Pi_{\mathsf{PCO}(x)}), \kappa)$ to verify the reduction of $\mathsf{x}_{\mathsf{PCO}(x),i}$ to $(\mathsf{x}_{\mathsf{PCO}(x)})$ using $\pi_{3,i}$.
  4. Use PC to verify the proof π_{PC} of the final statement $\mathsf{x}_{\mathsf{PCO}}$.

Theorem 2.4.2 (Thm. 7.2.1 informal)**.** *Let $K = 2^{\kappa}$ be a constant and $\mathsf{NIRoK} : \mathcal{R} \times \mathcal{R} \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}$ be a non-interactive reduction of knowledge. Then construction $\mathsf{L}(\mathsf{NIRoK}, \kappa)$ is a (plain) Local Folding scheme for $\mathcal{R}$.*

We argue similarly as in the case of $\mathsf{L}_{\mathsf{PCC}}(\mathsf{NIRoK}, \kappa)$. In particular, we consider the perspective of the i-th verifier. The verification of $\mathsf{L}_{\mathsf{PCC}}$ can be viewed as a reduction of knowledge as demonstrated in Thm. 2.4.1. Therefore, the verification of L decomposes as a sequential composition of reductions

of knowledge: the local folding verification together with the multi-verifier Fiat–Shamir step yields $\mathcal{R} \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCO}(x)}$, where x is the evaluation point fixed by the shared first verifier message; the resulting $\mathcal{R}_{\mathsf{PCO}(x)}$ statement is then folded once more, and finally proven by a polynomial commitment opening. Since $\pi_{3,i}$ and π_{PC} verify, the verifier's $\mathcal{R}_{\mathsf{PCO}(x)}$ statement is true which implies that the "promise" must be true, since the x in $\mathcal{R}_{\mathsf{PCO}(x)}$ is defined by the $\mathcal{R}_{\mathsf{PCO}}$ statement. Having verified the promise, the verification as a whole is a reduction of knowledge $\mathcal{R} \to \mathcal{R}$ and therefore the construction derived by $\mathsf{L}(\mathsf{NIRoK}, \kappa)$ is knowledge sound.

2.5 Application: Amortized Delegation of Bivariate Polynomial Evaluations

From R1CS to bivariate polynomial evaluation. For completeness, we give here a brief overview of Marlin-like [CHM+20] SNARK constructions. We note that Spartan-like [Set20] schemes also admit a reduction to such relations, albeit following a different route. For a more detailed overview of the reductions, we refer the reader to [PRZ26]. For ease of exposition, we omit the public inputs and batching techniques, and further assume square R1CS matrices. In what follows, we denote by $\boldsymbol{\lambda}(X)$ the Lagrange basis polynomials over the subgroup $\mathbb{H}$ of n-th roots of unity.

In R1CS, the prover must convince the verifier that it knows a vector $\mathbf{z} \in \mathbb{F}^n$ such that $\mathbf{Az} \circ \mathbf{Bz} = \mathbf{Cz}$, where $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{n \times n}$ are public matrices encoding the desired computation, where each has at most s non-zero entries. The protocol begins with $\mathcal{P}$ sending commitments to polynomials $\mathbf{z}^\top \boldsymbol{\lambda}(X)$ and $\mathbf{z}_M^\top \boldsymbol{\lambda}(X)$, where $\mathbf{z}_M = \mathbf{Mz}$, for $M \in \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$, to which $\mathcal{V}$ responds with a random $\alpha \in \mathbb{F}$. Now, $\mathcal{P}$ needs to prove that $\boldsymbol{\lambda}(\alpha)^\top \mathbf{z}_M = \boldsymbol{\lambda}(\alpha)^\top \mathbf{Mz}$, which can be viewed as an inner-product equality. Hence, $\mathcal{P}$ can convince $\mathcal{V}$ by showing that the univariate sumcheck [BCR+19] of the polynomial

$$g(X) \coloneqq \boldsymbol{\lambda}(\alpha)^\top \boldsymbol{\lambda}(X) \cdot \mathbf{z}_M^\top \boldsymbol{\lambda}(X) - (\boldsymbol{\lambda}(\alpha)^\top \mathbf{M})^\top \boldsymbol{\lambda}(X) \cdot \mathbf{z}^\top \boldsymbol{\lambda}(X)$$

is 0. Since the univariate sumcheck reduces this in turn to evaluation at a random point $\beta \in \mathbb{F}$ sampled by $\mathcal{V}$, the task reduces to checking (1) committed univariate polynomial evaluation claims, and (2) correctness of values $y_M = \boldsymbol{\lambda}(\alpha)^\top \mathbf{M} \boldsymbol{\lambda}(\beta)$, for $\mathbf{M} \in \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$. If $\mathcal{P}$ does not prove (2), then one would have only a *proof-succinct* zkNARK where $\mathcal{V}$ directly checks the claims itself. Crucially, proving (2) succinctly can be delegated to an external server without harming privacy, since it only concerns publicly available information. This work focuses on amortizing the work of a server who is tasked with producing K proofs of (2) that show that $y_M = \boldsymbol{\lambda}(\alpha)^\top \mathbf{M} \boldsymbol{\lambda}(\beta)$, for $\mathbf{M} \in \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$. We illustrate how this can be composed with collaborative zkSNARK protocols in Fig. 1.

Amortizing Bivariate Evaluations. A sparse matrix $\mathbf{M}$ can be identified with a bivariate polynomial $P(X, Y)$ of individual degree at most $n - 1$ in each variable via evaluations over the grid $\mathbb{H} \times \mathbb{H}$, in which case $P(\alpha, \beta) = \boldsymbol{\lambda}(\alpha)^\top \mathbf{M} \boldsymbol{\lambda}(\beta)$. Moreover, P remains sparse in the Lagrange basis. Therefore, we consider the problem of amortizing proofs for the relation $\mathcal{R}_P$, where

$$\mathcal{R}_P = \{(\alpha, \beta, y) \mid P(\alpha, \beta) = y\}$$

for some fixed sparse polynomial P determined by the circuit description. Since $\mathcal{R}_P$ is efficiently decidable, the verifier merely aims to outsource the computation of $P(\alpha, \beta)$, whose cost is roughly $\|P\|$ field operations, where $\|P\|$ denotes the sparsity of $P(X, Y)$. To enable repeated reductions to the same relation, we introduce the relaxed relation:

$$\mathcal{R}_{\mathbf{M}} = \left\{ \begin{array}{c} (\mathsf{c}, \mathsf{d}, y) \\ (\mathbf{u}, \mathbf{v}) \end{array} \;\middle|\; \begin{array}{c} \mathsf{c}, \mathsf{d} \text{ are commitments to} \\ \mathbf{u}^\top \boldsymbol{\lambda}(X),\ \mathbf{v}^\top \boldsymbol{\lambda}(X) \\ \text{and } \mathbf{u}^\top \mathbf{M} \mathbf{v} = y \end{array} \right\}.$$

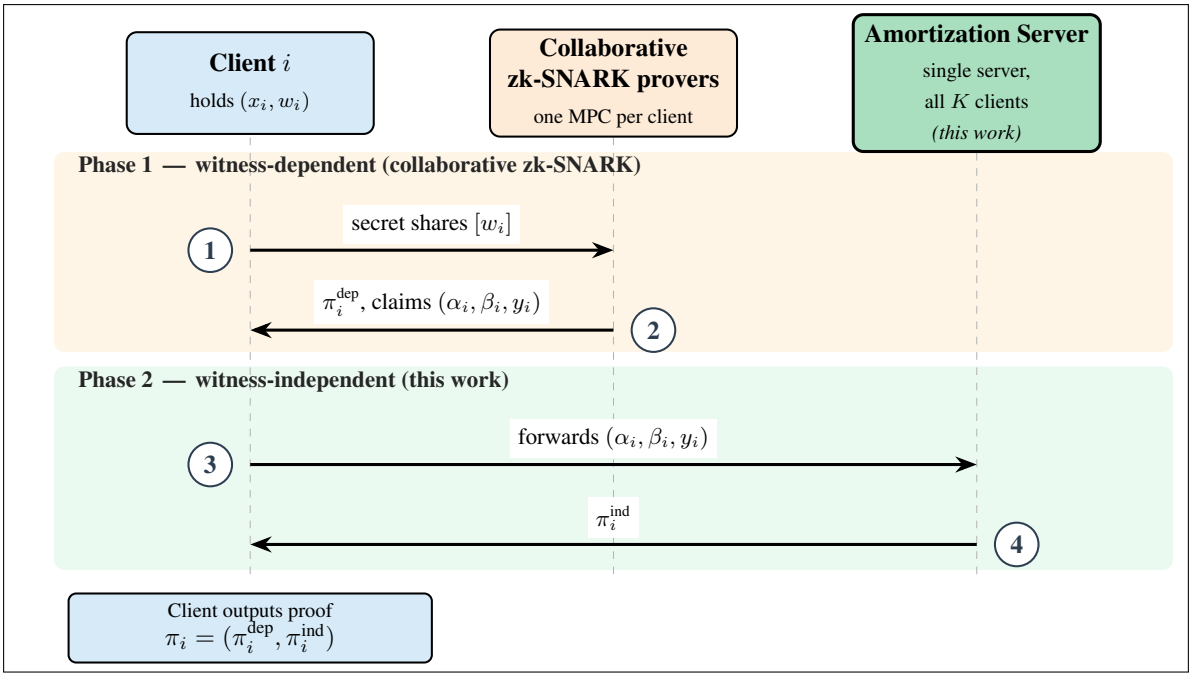

Figure 1: Workflow of our framework composed with collaborative zkSNARK protocols, from the perspective of a single client i. **Phase 1:** the client secret-shares its witness to a dedicated collaborative zk-SNARK MPC instance (1), which returns the witness-dependent proof component together with the witness-independent claims (α_i, β_i, y_i) derived during proving (2). **Phase 2:** the client forwards the claims to a single amortization server shared across all K clients (3), which returns the witness-independent proof component (4). The amortization server is the only component whose work is shared across clients.

In what follows, consider the concrete case of using KZG [KZG10] as the polynomial commitment[3]. Setting $\mathbf{u} = \boldsymbol{\lambda}(\alpha)$ and $\mathbf{v} = \boldsymbol{\lambda}(\beta)$ recovers an instance of $\mathcal{R}_P$. Our goal is to construct a reduction of knowledge $\Pi_{\mathbf{M}} : \mathcal{R}_{\mathbf{M}}^2 \to \mathcal{R}_{\mathbf{M}}$, after which we can apply the compiler L to derive a local folding scheme for the relation.

We begin from the standard folding approach. Given two statements, define

$$y^* = \mathbf{u}^{*\top}\mathbf{M}\mathbf{v}^* = (\mathbf{u}_0 + \gamma\mathbf{u}_1)^\top\mathbf{M}(\mathbf{v}_0 + \gamma\mathbf{v}_1),$$

where γ is a verifier challenge. Before receiving γ, the prover sends the "cross term" $z = \mathbf{u}_0^\top\mathbf{M}\mathbf{v}_1 + \mathbf{u}_1^\top\mathbf{M}\mathbf{v}_0$. The folded evaluation is then $y^* = y_0 + \gamma z + \gamma^2 y_1$. The verifier computes the folded commitments as $\mathsf{c}^* = \mathsf{c}_0 + \gamma\mathsf{c}_1$, and $\mathsf{d}^* = \mathsf{d}_0 + \gamma\mathsf{d}_1$, while the prover derives the folded witness polynomials $(\mathbf{u}_0 + \gamma\mathbf{u}_1)^\top\boldsymbol{\lambda}(X)$ and $(\mathbf{v}_0 + \gamma\mathbf{v}_1)^\top\boldsymbol{\lambda}(X)$. This already yields a reduction of knowledge to which we can apply the compiler L. However, the server still incurs unnecessary cost: it must commit to $\mathbf{u}_i = \boldsymbol{\lambda}(\alpha_i)$ and $\mathbf{v}_i = \boldsymbol{\lambda}(\beta_i)$ for each client, requiring two size-n MSMs, where n is the dimension of $\mathbf{M}$, incurring a total prover cost that remains $\mathcal{O}(nK)$.

Leveraging succinct representations of verifier polynomials. The vectors $\mathbf{u} = \boldsymbol{\lambda}(\alpha)$ and $\mathbf{v} = \boldsymbol{\lambda}(\beta)$ admit succinct descriptions: each is fully determined by a single field element. However, they are not sparse in the usual sense, and committing to them still requires large MSMs. Instead, we exploit the fact

[3]The construction itself does not depend on KZG; any homomorphic polynomial commitment suffices. When instantiated with KZG, we obtain the efficiency bounds of Table 1.

that (see, e.g., [CFF+21])

$$\boldsymbol{\lambda}(\alpha)^\top \boldsymbol{\lambda}(X) = \frac{(X^n-1)\alpha - (\alpha^n-1)X}{n(X-\alpha)}$$

and let the server *implicitly commit* to these vectors via commitments $\mathsf{c}_N, \mathsf{c}_T$ to

$$N(X) = (X^n-1)\alpha - (\alpha^n-1)X, \qquad T(X) = X - \alpha.$$

The pair binds the server to $\boldsymbol{\lambda}(\alpha)^\top \boldsymbol{\lambda}(X)$ while requiring only $\mathcal{O}(1)$ group operations to construct. We refine $\mathcal{R}_{\mathbf{M}}$ to work with *sparse, implicit* polynomials as follows:

$$\mathcal{R}_{\mathbf{M}} = \left\{ \begin{pmatrix} y, \mathsf{c}_N, \mathsf{c}_T, \\ \mathsf{d}_N, \mathsf{d}_T, \end{pmatrix}, \begin{pmatrix} N_u(X), T_u(X) \\ N_v(X), T_v(X) \\ \mathbf{u}, \mathbf{v} \end{pmatrix} \;\middle|\; \begin{array}{c} \mathsf{c}_N, \mathsf{c}_T \text{ are commitments to } N_u(X), T_u(X), \\ \mathsf{d}_N, \mathsf{d}_T \text{ are commitments to } N_v(X), T_v(X), \\ N_u(X)/nT_u(X) = \mathbf{u}^\top \boldsymbol{\lambda}(X) \\ N_v(X)/nT_v(X) = \mathbf{v}^\top \boldsymbol{\lambda}(X) \\ \mathbf{u}^\top \mathbf{M} \mathbf{v} = y \end{array} \right\}$$

Attempting to use the above reduction of knowledge we encounter a new problem: how to compute the commitment to $(\boldsymbol{\lambda}(\alpha_0) + \gamma \boldsymbol{\lambda}(\alpha_1))^\top \boldsymbol{\lambda}(X)$? Some algebraic manipulation gives:

$$(\boldsymbol{\lambda}(\alpha_0) + \gamma \boldsymbol{\lambda}(\alpha_1))^\top \boldsymbol{\lambda}(X) = \frac{N_0(X)}{nT_0(X)} + \gamma \frac{N_1(X)}{nT_1(X)} = \frac{N_0(X)T_1(X) + \gamma N_1(X) T_0(X)}{nT_0(X)T_1(X)}$$

We ask the prover to send commitments to $N_u(X), T_u(X), N_v(X), T_v(X)$ where

$$N_u(X) = N_{u,0}(X)T_{u,1}(X) + \gamma N_{u,1}(X)T_{u,0}(X), \quad T_u(X) = T_{u,0}(X)T_{u,1}(X) \tag{1}$$

and similarly for $N_v(X), T_v(X)$. Note that these equations, together with the commitments, define polynomial constraints, and therefore an $\mathcal{R}_{\mathsf{PCC}}$ statement. The following invariant holds if we handle y^* as in the previous case: the rational functions $N_u(X)/nT_u(X), N_v(X)/nT_v(X)$ are polynomials $\mathbf{u}^\top \boldsymbol{\lambda}(X), \mathbf{v}^\top \boldsymbol{\lambda}(X)$ of degree less than n, such that $\mathbf{u}^\top \mathbf{M} \mathbf{v} = y^*$. We conclude that the resulting construction is a reduction of knowledge $\Pi_{\mathbf{M}} : \mathcal{R}_{\mathbf{M}}^2 \to \mathcal{R}_{\mathbf{M}} \times \mathcal{R}_{\mathsf{PCC}}$ where the PCC constraints are those of Eq. 1 and the corresponding constraints for $N_v(X), T_v(X)$. Applying the compiler L to $\Pi_{\mathbf{M}}$, we derive a (plain) Local Folding scheme for $\mathcal{R}_{\mathbf{M}}$ and therefore, for relation $\mathcal{R}_P$.

Efficiency. We are interested in the case of delegating evaluations of $P(\alpha, \beta)$. Therefore, we want to measure the efficiency in the case of applying the local folding scheme to inputs derived from sparse polynomials $\boldsymbol{\lambda}(\alpha)$, $\boldsymbol{\lambda}(\beta)$. To do so we need to bound how the degrees and sparsities of N_u, T_u, N_v, T_v grow during the recursive calls to $\Pi_{\mathbf{M}}$. For the polynomials T_u, T_v, it is easy to see that their degree and sparsity doubles in each iteration. For the case of N_u, N_v, we note that during the ℓ-th invocation they can be written as $p(X)X^n + q(X)$ where the degrees of $p(X), q(X)$ is roughly 2^ℓ and therefore the sparsity is roughly $2^{\ell+1}$. Also note that the final polynomial opening performed in the final step of Λ is of degree roughly $n + K$. The initial commitments of the i-th statement, corresponding to the sparse commitments to $\boldsymbol{\lambda}(\alpha_i), \boldsymbol{\lambda}(\beta_i)$, can be re-computed and checked by the verifier alone using constant group operations. Thus, to verify the statement $y_i = \boldsymbol{\lambda}(\alpha_i)^\top \mathbf{M} \boldsymbol{\lambda}(\beta_i)$, it is enough for the verifier to reconstruct the statement and verify the aggregated and the i-th local proof.

We next present an informal theorem stating the efficiency of the construction

Theorem 2.5.1 (Cor. 8.2.1 informal)**.** *Let $\mathbf{M} \in \mathbb{F}^{n \times n}$, $K < n$ and consider an execution of $\Lambda = \mathsf{L}(\mathsf{FS}(\Pi_{\mathbf{M}}), \kappa)$ using the KZG commitment scheme on input statements derived from statements (α, β, y) of $\mathcal{R}_P$. Then*

- *the asymptotic proving time of Λ is $(K \cdot s + n \cdot \log n)\ \mathbb{F} + (n + K \cdot \log K)\, \mathbb{G}$*

- *the asymptotic verifying time of* Λ *is dominated by* $1 \cdot \mathsf{P} + \log K \cdot (\mathbb{F} + \mathbb{G} + \rho)$,
- *the proof size is* $\lambda \cdot \log K$.

where $\mathbb{G}$, $\mathbb{F}$ *denote group and field operations,* P *pairing operations, and* ρ *random oracle queries.*

Note, in particular, that the cryptographic cost of proving K statements grows with $n + K \cdot \log K$ instead of $K \cdot n$.

Verifying the final $\mathcal{R}_{\mathbf{M}}$ instance. After the completion of Λ, each $\mathcal{V}_i$ is left with checking the i-th local proof as well as the common final folded $\mathcal{R}_{\mathbf{M}}$ statement. The latter can be reduced to a fresh instance in $\mathcal{R}_P$ as follows.

$\mathcal{P}$ provides commitments c, d to polynomials $\mathbf{u}^\top \boldsymbol{\lambda}(X) = \frac{N_u(X)}{nT_u(X)}$, $\mathbf{v}^\top \boldsymbol{\lambda}(X) = \frac{N_v(X)}{nT_v(X)}$, respectively, and proves their well formedness against the commitments to the final numerators and denominators c_N, c_T, d_N, and d_T. Now, each $\mathcal{V}$ is left with checking an instance of $\mathcal{R}_{\mathbf{M}}$ with explicit commitment to the left and right vectors. To this end, we refer the reader to [PRZ26], where the authors provide two reductions of knowledge with different efficiency profiles from said relation to $\mathcal{R}_P$ (their Π_{GBF} reductions of Section 4). After this step, each $\mathcal{V}_i$ is left with verifying an instance in $\mathcal{R}_P$, which can be done by running any argument of knowledge for this relation (e.g. general purpose bivariate polynomial evaluation arguments such as the ones in [BMM+21; GPPS26], or through sparse matrix evaluation arguments as in [CHM+20]).

3 Preliminaries

Sets and Algorithms. We write $[n] = \{1, \ldots, n\}$. We often index collections over the hypercube $\{0,1\}^\kappa$: for a set $S = \{s_1, \ldots, s_{2^\kappa}\}$, we equivalently write $S = \{s_{\mathbf{b}}\}_{\mathbf{b} \in \{0,1\}^\kappa}$ where $s_{\mathbf{b}} = s_i$ for $i = 1 + \sum_j b_j 2^{j-1}$. We use the shorthand $s_{b\mathbf{b}'}$, $s_{\mathbf{b}'b}$, and $s_{\mathbf{b}_1\mathbf{b}_2}$ for the corresponding concatenated bitstrings. We denote the empty string with ϵ. We write $x \leftarrow S$ for sampling a uniform element from S, and $y \leftarrow \mathcal{A}^{\boldsymbol{\mathcal{O}}}(x)$ for executing $\mathcal{A}$ on input x with fresh randomness and oracle access to $\boldsymbol{\mathcal{O}} = (\mathcal{O}_1, \ldots, \mathcal{O}_\ell)$.

Security Parameter and Random Oracles. We denote the security parameter by λ and assume all inputs are of length polynomial in λ. Let $\mathcal{U}(\lambda)$ denote the set of functions $\rho : \{0,1\}^* \to \{0,1\}^\lambda$. A random oracle is sampled uniformly from $\mathcal{U}(\lambda)$.

Algebraic Structures. Throughout this work $\mathbb{F}$ is a finite field. We denote vectors with $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{F}^n$ and matrices with $\mathbf{M} = (m_{ij})_{i,j=1}^n$. The sparsity of a matrix $\|\mathbf{M}\|$ is the number of non-zero elements in $\mathbf{M}$. The subspace of polynomials of degree less than d in $\mathbb{F}[X]$ is denoted as $\mathbb{F}^{<d}[X]$. Similarly to matrices, the sparsity of a polynomial $\|p\|$ is the number of non-zero coefficients it has w.r.t. a fixed basis. Let $\mathbb{H} = \{h_1, \ldots, h_n\}$ be a multiplicative subgroup of $\mathbb{F}$. We denote $\boldsymbol{\lambda}_{\mathbb{H}}(X) = (\lambda_{h_1}(X), \ldots, \lambda_{h_n}(X))$ the Lagrange polynomials basis associated to $\mathbb{H}$. We omit the subscripts and simply write $\boldsymbol{\lambda}(X) = (\lambda_1(X), \ldots, \lambda_n(X))$ when the context allows. We use the following fact (see, e.g., [CFF+21]):

$$\boldsymbol{\lambda}(\alpha)^\top \boldsymbol{\lambda}(X) = \frac{(X^n - 1)\alpha - (\alpha^n - 1)X}{n(X - \alpha)}$$

We identify bivariate polynomials $P(X, Y) \in \mathbb{F}[X, Y]$ with their evaluation matrices $\mathbf{M} = (P(h_i, h_j))_{i,j \in [n]^2}$ over the domain $\mathbb{H}$. If P is sparse, then so is $\mathbf{M}$, and $P(\alpha, \beta) = \boldsymbol{\lambda}(\alpha)^\top \mathbf{M} \boldsymbol{\lambda}(\beta)$, where $\boldsymbol{\lambda}(\alpha)$ denotes the vector of Lagrange basis evaluations at α. Conversely, every matrix $\mathbf{M}$ over $\mathbb{H} \times \mathbb{H}$ defines a corresponding bivariate polynomial of individual degree at most $n - 1$.

3.1 Polynomial Commitments

A polynomial commitment scheme consists of PPT algorithms $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ where

- $(\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{PC.Setup}(1^\lambda, D)$: on input the security parameter 1^λ and a degree bound $D = \mathsf{poly}(\lambda)$ outputs a commitment key ck and a verification key vk,
- $\mathsf{c} \leftarrow \mathsf{PC.Commit}(\mathsf{ck}, p(X), d)$: on input a commitment key ck, a polynomial $p(X) \in \mathbb{F}^{<d}$ and a degree $d \leq D$ outputs a commitment c,
- $\pi \leftarrow \mathsf{PC.Open}(\mathsf{ck}, \mathsf{c}, x, y, p(X), d)$: on input a commitment key ck, a commitment c, two field elements $x, y \in \mathbb{F}$, a polynomial $p(X) \in \mathbb{F}^{<d}$ and a degree $d \leq D$ outputs a proof π,
- $0/1 \leftarrow \mathsf{PC.Verify}(\mathsf{vk}, \mathsf{c}, x, y, \pi, d)$ on input a verification key vk, a commitment c, two field elements $x, y \in \mathbb{F}$, a proof π and a degree bound $d \leq D$ outputs a bit.

Completeness. For all $\lambda \in \mathbb{N}$, $D = \mathsf{poly}(\lambda)$, $d \leq D$, and $p(X) \in \mathbb{F}^{<d}[X]$ and $x \in \mathbb{F}$:

$$\Pr\left[\begin{array}{l} 1 \neq \mathsf{PC.Verify}(\mathsf{vk}, \mathsf{c}, x, y, \pi, d) \end{array} \middle| \begin{array}{r} (\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{PC.Setup}(1^\lambda, D) \\ \mathsf{c} \leftarrow \mathsf{PC.Commit}(\mathsf{ck}, p(X), d) \\ \pi \leftarrow \mathsf{PC.Open}(\mathsf{ck}, \mathsf{c}, x, p(x), p(X), d) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Binding. For all $\lambda \in \mathbb{N}$, $D = \mathsf{poly}(\lambda)$ and all PPT adversaries $\mathcal{A}$:

$$\Pr\left[\begin{array}{c} \mathsf{c} = \mathsf{PC.Commit}(\mathsf{ck}, p_1(X), d_1) \\ \mathsf{c} = \mathsf{PC.Commit}(\mathsf{ck}, p_2(X), d_2) \\ p_1(X) \neq p_2(X) \end{array} \middle| \begin{array}{c} (\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{PC.Setup}(1^\lambda, D) \\ \\ (\mathsf{c}, p_1, d_1, p_2, d_2) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Extractability. For all $\lambda \in \mathbb{N}$, $D = \mathsf{poly}(\lambda)$ and all PPT adversaries $\mathcal{A}$ there exists an expected polynomial time extractor $\mathcal{E}$ such that:

$$\Pr\left[\begin{array}{c} \mathsf{PC.Verify}(\mathsf{vk}, \mathsf{c}, x, y, \pi, d) = 1 \wedge \\ \\ (p(x) \neq y \ \vee \ \deg(p) \geq d \ \vee \ \mathsf{c} \neq \mathsf{PC.Commit}(\mathsf{ck}, p(X), d)) \end{array} \middle| \begin{array}{r} (\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{PC.Setup}(1^\lambda, D) \\ (\mathsf{c}, x, y, \pi, d) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{vk}) \\ p(X) \leftarrow \mathcal{E}^{\mathcal{A}(\mathsf{ck}, \mathsf{vk})} \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Additionally, a polynomial commitment scheme is homomorphic if there exists an efficient algorithm $\mathsf{Combine}$ that on input the commitment key ck, two commitments $\mathsf{c}_1, \mathsf{c}_2$ and two field elements $\alpha, \beta \in \mathbb{F}$ outputs a fresh commitment c^* s.t. for all $\lambda \in \mathbb{N}$, $D = \mathsf{poly}(\lambda)$, $d_1, d_2 \leq D$ and all $\alpha, \beta \in \mathbb{F}$:

$$\Pr\left[\begin{array}{l} \mathsf{c}^* \neq \mathsf{PC.Commit}(\mathsf{ck}, p^*(X), \max\{d_1, d_2\}) \end{array} \middle| \begin{array}{r} (\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{PC.Setup}(1^\lambda, D) \\ \mathsf{c}_1 \leftarrow \mathsf{PC.Commit}(\mathsf{ck}, p_1(X), d_1) \\ \mathsf{c}_2 \leftarrow \mathsf{PC.Commit}(\mathsf{ck}, p_2(X), d_2) \\ \mathsf{c}^* \leftarrow \mathsf{Combine}(\mathsf{ck}, \mathsf{c}_1, \alpha, \mathsf{c}_2, \beta) \\ p^*(X) = \alpha p_1(X) + \beta p_2(X) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

We also denote $\mathsf{Combine}(\mathsf{ck}, \mathsf{c}_1, \alpha, \mathsf{c}_2, \beta)$ with $\alpha \mathsf{c}_1 + \beta \mathsf{c}_2$.

3.2 Merkle Trees

In this section, we recall Merkle trees. Let $K = 2^\kappa \in \mathbb{N}$ and $\boldsymbol{\rho} = (\rho_0, \ldots, \rho_\kappa)$ where $\rho_i \leftarrow \mathcal{U}(\lambda)$. A Merkle tree in the random oracle model consists of three algorithms

- $\mathsf{h}, \{\pi_{\mathbf{b}}\}_{\mathbf{b} \in \{0,1\}^\kappa} \leftarrow \mathsf{MT.Prove}^{\boldsymbol{\rho}(\cdot)}(\kappa, \{\mathsf{x}_{\mathbf{b}}\}_{\mathbf{b} \in \{0,1\}^\kappa})$

- $\mathsf{h}' \leftarrow \mathsf{MT.Path}^{\rho(\cdot)}(\kappa, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \{\mathsf{h}_i\}_{i\in\{1,\ldots,\kappa\}})$
- $0/1 \leftarrow \mathsf{MT.Verify}^{\rho(\cdot)}(\kappa, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \mathsf{h}, \pi_{\mathbf{b}})$

described in Fig. 2.

$\mathsf{MT.Prove}^{\rho(\cdot)}(\kappa, \{\mathsf{x}_{\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa})$

if $\kappa = 0$ output $(\mathsf{h}, \pi) = (\rho_0(\mathsf{x}), \epsilon)$,
otherwise:

$\boldsymbol{\rho} := (\rho_0, \ldots, \rho_{\kappa-1})$
$\mathsf{h}_0, \{\pi_{0\mathbf{b}'}\} \leftarrow \mathsf{MT.Prove}^{\rho(\cdot)}(\kappa - 1, \{\mathsf{x}_{0\mathbf{b}'}\}_{\mathbf{b}'\in\{0,1\}^{\kappa-1}})$
$\mathsf{h}_1, \{\pi_{1\mathbf{b}'}\} \leftarrow \mathsf{MT.Prove}^{\rho(\cdot)}(\kappa - 1, \{\mathsf{x}_{1\mathbf{b}'}\}_{\mathbf{b}'\in\{0,1\}^{\kappa-1}})$
$\mathsf{h} \leftarrow \rho_\kappa(\mathsf{h}_0, \mathsf{h}_1)$
For $(b, \mathbf{b}') \in \{0,1\}^\kappa$:
  $\pi_{b\mathbf{b}'} := (\pi_{\mathbf{b}'}, \mathsf{h}_{1-b})$
Output $\mathsf{h}, \{\pi_{\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa}$

$\mathsf{MT.Path}^{\rho(\cdot)}(\kappa, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \{\mathsf{h}_i\}_{i\in\{1,\ldots,\kappa\}})$

Parse $\mathbf{b} := (b_1, \ldots, b_\kappa)$.
$\mathsf{h}_{b_1} \leftarrow \rho_0(\mathsf{x}_{\mathbf{b}})$
For $i \in \{1, \ldots, \kappa\}$:
  $\mathsf{h}_{1-b_i} \leftarrow \mathsf{h}_i$
  $\mathsf{h}_{b_i} \leftarrow \rho_i(\mathsf{h}_0, \mathsf{h}_1)$
Output h_b

$\mathsf{MT.Verify}^{\rho(\cdot)}(\kappa, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \mathsf{h}, \pi_{\mathbf{b}})$

Parse $\pi_{\mathbf{b}} := \{\mathsf{h}_i\}_{i\in\{1,\ldots,\kappa\}}$
Output $\mathsf{h} \stackrel{?}{=} \mathsf{MT.Path}^{\rho(\cdot)}(\kappa, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \{\mathsf{h}_i\}_{i\in\{1,\ldots,\kappa\}})$

Figure 2: Merkle Tree Construction in the random oracle model.

4 Reductions of Knowledge

In this section we recall *reductions of knowledge*. Our definition closely follows the one by Kothapalli and Parno [KP23] but allows the prover also having access to some auxiliary input. Soundness should hold for auxiliary inputs that are chosen adversarialy.

4.1 (Public Coin) Reduction of Knowledge

Let $\mathcal{R}, \mathcal{R}'$ be ternary relation consisting of public parameters, statement and witness tuples. A reduction of knowledge $\Pi : \mathcal{R} \to \mathcal{R}'$ is a tuple $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ where

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}')$ is a PPT algorithm that on input the security parameter λ and parameters $\mathsf{pp}' = \{0,1\}^{\mathsf{poly}(\lambda)}$ outputs parameters pp.

- $(\mathsf{w}', \mathsf{x}') \leftarrow \langle \mathcal{P}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux}), \mathcal{V}(\mathsf{pp}, \mathsf{x}) \rangle$ is an interactive protocol between PPT (interactive) algorithms $\mathcal{P}, \mathcal{V}$, denoted as the prover and verifier respectively, where $\mathcal{P}$ takes as input the parameters pp, a statement x, a witness w and auxiliary input aux and after the interaction outputs a fresh witness w', and $\mathcal{V}$ takes as input the parameters pp and a statement x and after the interaction outputs statement x'.

Completeness. For all $\lambda \in \mathbb{N}$, $\mathsf{pp}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$ and PPT algorithms $\mathcal{A}$

$$\Pr\left[\begin{array}{c|c} (\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R} & \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \wedge & (\mathsf{x}, \mathsf{w}, \mathsf{aux}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{pp}, \mathsf{x}', \mathsf{w}') \notin \mathcal{R}' & (\mathsf{w}', \mathsf{x}') \leftarrow \langle \mathcal{P}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux}), \mathcal{V}(\mathsf{pp}, \mathsf{x}) \rangle \end{array}\right] = 0$$

We write $\Pi : \mathcal{R} \to \mathcal{R}'$ to denote that Π is a reduction of knowledge from relation $\mathcal{R}$ to relation $\mathcal{R}'$.

Knowledge Soundness. For all $\lambda \in \mathbb{N}$, $\mathsf{pp}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$ and all expected polynomial time adversaries $\mathcal{P}^*, \mathcal{A}$ there exists a polynomial time extractor $\mathcal{E}$ such that

$$\Pr\left[(\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ (\mathsf{x}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \mathsf{w} \leftarrow \mathcal{E}(\mathsf{pp}, \mathsf{x}, \mathsf{st}) \end{array}\right] \geq$$

$$\Pr\left[(\mathsf{pp}, \mathsf{x}', \mathsf{w}') \in \mathcal{R}' \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ (\mathsf{x}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{w}', \mathsf{x}') \leftarrow \langle \mathcal{P}^*(\mathsf{pp}, \mathsf{x}, \mathsf{st}), \mathcal{V}(\mathsf{pp}, \mathsf{x}) \rangle \end{array}\right] - \mathsf{negl}(\lambda)$$

Public Reducibility. For all $\lambda \in \mathbb{N}$, $\mathsf{pp}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$, there exists a polynomially computable function ϕ such that for all polynomial time adversaries $\mathcal{A}, \mathcal{P}^*$

$$\Pr\left[\phi(\mathsf{pp}, \mathsf{x}, \mathsf{tr}) \neq \mathsf{x}' \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ (\mathsf{x}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{w}', \mathsf{x}') \leftarrow \langle \mathcal{P}^*(\mathsf{pp}, \mathsf{x}, \mathsf{st}), \mathcal{V}(\mathsf{pp}, \mathsf{x}) \rangle \\ \mathsf{tr} \leftarrow \mathsf{tr}\left(\langle \mathcal{P}^*(\mathsf{pp}, \mathsf{x}, \mathsf{st}), \mathcal{V}(\mathsf{pp}, \mathsf{x}) \rangle\right) \end{array}\right] = 0$$

where $\mathsf{tr}(\cdot)$ denotes the transcript of the interaction of $\mathcal{P}^*, \mathcal{V}$ with their respective inputs.

Public Coin. If furthermore the protocol $\langle \mathcal{P}, \mathcal{V} \rangle$ is public coin, we say that the reduction of knowledge is also public coin.

Note that the auxiliary input is not explicit in the knowledge soundness property. Any such input can be part of the value st output by $\mathcal{A}$. Therefore, the corresponding results presented in [KP23] immediately hold for this stronger notion.

We also recall the results for abstracting knowledge soundness for a large family of algebraic public coin protocols. These results are taken verbatim from [KP23] which in turn is an adaptation of the results of [BCS21].

Definition 4.1.1 (Tree of accepting transcripts)**.** Consider an m-round public-coin interactive protocol $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ that satisfies the interface described in Def. 4.1. A $(n_1, \ldots, n_m)$-tree of accepting transcripts for statement x is a tree of depth m where each node at layer i has n_i outgoing edges such that (1) each node in layer $i \in [m]$ is labeled with a prover message for round i; (2) each outgoing edge from layer $i \in [m]$ is labeled with a different choice of verifier randomness for round i; (3) each leaf is labeled with an accepting statement-witness pair output by the prover and verifier corresponding to the interaction along the path.

Lemma 4.1.1. *Consider an m-round public-coin interactive protocol $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ that satisfies the interface described in Def. 4.1 and completeness. Then $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ is a reduction of knowledge if there exists a PPT extractor χ that, for all instances x_1 outputs a satisfying witness w_1 with probability $1 - \mathsf{negl}(\lambda)$, given an $(n_1, \ldots, n_m)$-tree of accepting transcripts for x_1 where the verifiers randomness is sampled from space Q such that $|Q| = O(2^\lambda)$, and $\prod_i n_i = \mathsf{poly}(\lambda)$.*

Remark 4.1.1. In the definition, we allow the Gen algorithm to also take external parameters pp' as input. This slight modification compared to [KP23] allows us to consider "trivial" reductions of knowledge, as for example the identity reduction, where the parties simply output their corresponding inputs. In this case, the corresponding Gen algorithm can simply output pp' as its own parameters.

4.2 Non-interactive Reduction of Knowledge in the RO model

Let $\mathcal{R}, \mathcal{R}'$ be ternary relations. A non-interactive reduction of knowledge $\mathsf{NIRoK} : \mathcal{R} \to \mathcal{R}'$ for the pair $\mathcal{R}, \mathcal{R}'$ is a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{Reduce}, \mathsf{Verify})$ where

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}')$: on input the security parameter λ and parameters $\mathsf{pp}' = \{0,1\}^{\mathsf{poly}(\lambda)}$ outputs parameters pp.
- $\pi, \mathsf{x}', \mathsf{w}' \leftarrow \mathsf{Reduce}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux})$: on input the parameters pp, a pair (x, w), auxiliary input aux and access to random oracles $\rho_1, \ldots, \rho_\ell$, outputs a proof π and a new pair $(\mathsf{x}', \mathsf{w}')$.
- $0/1 \leftarrow \mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{x}', \pi)$: on input the parameters pp, statements x, x', a proof π and access to random oracles $\rho_1, \ldots, \rho_\ell$ outputs a bit.

Completeness. For all $\lambda \in \mathbb{N}$, $\mathsf{pp}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$ and PPT algorithms $\mathcal{A}$:

$$\Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R} & \rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \wedge & \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ (b \neq 1 \ \vee (\mathsf{pp}, \mathsf{x}', \mathsf{w}') \notin \mathcal{R}) & \mathsf{x}, \mathsf{w}, \mathsf{aux} \leftarrow \mathcal{A}^{\rho(\cdot)}(\mathsf{pp}) \\ & \pi, \mathsf{x}', \mathsf{w}' \leftarrow \mathsf{Reduce}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux}) \\ & b \leftarrow \mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{x}', \pi) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Knowledge Soundness. For all $\lambda \in \mathbb{N}$, $\mathsf{pp}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$ and expected polynomial time adversaries $\mathcal{A}_1, \mathcal{A}_2$, there exists an expected polynomial time extractor $\mathcal{E}$ such that:

$$\Pr\left[(\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R} \;\middle|\; \begin{array}{r} \rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}_1^{\rho(\cdot)}(\mathsf{pp}) \\ \mathsf{w} \leftarrow \mathcal{E}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{st}) \end{array}\right] \geq$$

$$\Pr\left[\begin{array}{c} (\mathsf{pp}, \mathsf{x}', \mathsf{w}') \in \mathcal{R}' \wedge \\ \mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{x}', \pi) = 1 \end{array} \;\middle|\; \begin{array}{r} \rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}_1^{\rho(\cdot)}(\mathsf{pp}) \\ \pi, \mathsf{x}', \mathsf{w}' \leftarrow \mathcal{A}_2^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st}) \end{array}\right] - \mathsf{negl}(\lambda)$$

If we have a public coin reduction of knowledge Π for which the FS transform holds (which is true when Π satisfies the conditions of Lemma 4.1.1), the Reduce and Verify functions correspond to the transformed protocol.

4.3 Fiat-Shamir transform for Reductions of Knowledge

We next recall the Fiat-Shamir Transform construction [FS87] in the context of reductions of knowledge. The transform FS takes as input a public coin reduction of knowledge $\Pi : \mathcal{R} \to \mathcal{R}'$ and returns a non-interactive reduction of knowledge in the random oracle model $\mathsf{NIRoK} = \mathsf{FS}(\Pi)$ by simulating the messages of Π's verifier $\mathcal{V}$ using these random oracles. In this work, we consider the transform where in each round a different random oracle is used to sample the verifier's challenge. One can also consider the use of a single random oracle (with a small soundness error overhead). We refer the reader to [CY24] for further details. The transform is presented in Fig. 3.

$\mathsf{Gen}(1^\lambda, \mathsf{pp}')$

output $\mathsf{pp} \leftarrow \Pi.\mathsf{Gen}(1^\lambda, \mathsf{pp}')$

$\mathsf{Reduce}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux})$

Simulate an execution of $(\mathsf{x}', \mathsf{w}') \leftarrow \langle \mathcal{P}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux}), \mathcal{V}(\mathsf{pp}, \mathsf{x}) \rangle$
where the verifier's messages $\gamma'_1, \ldots, \gamma'_\ell$ are computed as

$$\gamma'_i \leftarrow \begin{cases} \rho_i(\mathsf{pp}, \mathsf{x}), & \text{if } i = 0 \\ \rho_i(\gamma'_{i-1}, \gamma_i), & \text{otherwise} \end{cases}$$

where γ_i is the message sent by $\mathcal{P}$ on round i
Define $\pi := (\gamma_1, \ldots, \gamma_\ell)$ and output $(\mathsf{x}', \mathsf{w}', \pi)$

$\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{x}', \pi)$

Parse $\pi := (\gamma_1, \ldots, \gamma_\ell)$ and set $\mathsf{tr} := (\mathsf{pp}, \mathsf{x}, \gamma_1, \gamma'_1, \ldots, \gamma_\ell, \gamma'_\ell)$
where the verifier's messages $\gamma'_1, \ldots, \gamma'_\ell$ are computed as

$$\gamma'_i \leftarrow \begin{cases} \rho_i(\mathsf{pp}, \mathsf{x}), & \text{if } i = 0 \\ \rho_i(\gamma'_{i-1}, \gamma_i), & \text{otherwise} \end{cases}$$

Accept iff $\mathcal{V}(\mathsf{tr})$ outputs x'.

Figure 3: The Fiat-Shamir transform for reduction of knowledge. The transform takes an ℓ-round, public coin reduction of knowledge $\Pi : \mathcal{R} \to \mathcal{R}'$ and returns a non-interactive reduction of knowledge $\mathsf{FS}(\Pi) = \mathsf{NIRoK} : \mathcal{R} \to \mathcal{R}'$.

Theorem 4.3.1. *Let $\Pi : \mathcal{R} \to \mathcal{R}'$ be an ℓ-round, public coin interactive reduction of knowledge that satisfies the hypothesis of Lemma 4.1.1. Then the construction $\mathsf{NIRoK} = \mathsf{FS}(\Pi) : \mathcal{R} \to \mathcal{R}'$ of Fig 3 is a non-interactive reduction of knowledge in the random oracle model.*

Remark 4.3.1. Care must be taken when sequentially composing reductions of knowledge since (in the general case) the extractor time can grow exponentially due to its recursive structure. In this work, all sequential compositions presented are done by composing protocols a constant number of times which guarantees a constant blowup in extractor's time. Therefore, we don't consider the general case limitation and always assume that the sequentially composed protocols are knowledge sound.

4.4 Sequential and Parallel Composition of Reductions of Knowledge

We next recall the results of [KP23; Kot24] regarding composition of reductions of knowledge. We first present the definition for a relation pair.

Definition 4.4.1. Let $\mathcal{R}_1, \mathcal{R}_2$ be ternary relations. We define

$$\mathcal{R}_1 \times \mathcal{R}_2 = \{\mathsf{pp}, (\mathsf{x}_1, \mathsf{w}_1), (\mathsf{x}_2, \mathsf{w}_2) \mid (\mathsf{pp}, \mathsf{x}_1, \mathsf{w}_1) \in \mathcal{R}_1 \ \wedge \ (\mathsf{pp}, \mathsf{x}_2, \mathsf{w}_2) \in \mathcal{R}_2\}$$

Furthermore, for $n \in \mathbb{N}$ we define $\mathcal{R}^n = \underbrace{\mathcal{R} \times \ldots \times \mathcal{R}}_{n \text{ times}}$.

We next recall the definitions for sequential and parallel composition and present the theorem stating that both compositions result in reductions of knowledge.

Definition 4.4.2. Let $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$ be ternary relations.

1. Let $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2 = (\mathsf{Gen}, \mathcal{P}_1, \mathcal{V}_1)$ and $\Pi_2 : \mathcal{R}_2 \to \mathcal{R}_3 = (\mathsf{Gen}, \mathcal{P}_2, \mathcal{V}_2)$ be reductions of knowledge. We define the sequential composition of Π_1, Π_2 denoted as $\Pi_s = \Pi_2 \circ \Pi_1$ as the tuple $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ where $\langle \mathcal{P}(\mathsf{pp}, \mathsf{x}_1, \mathsf{w}_1, \mathsf{aux} = (\mathsf{aux}_1, \mathsf{aux}_2)), \mathcal{V}(\mathsf{pp}, \mathsf{x}_1) \rangle$ is the following protocol:
   - $\mathcal{P}, \mathcal{V}$ interactively compute $(\mathsf{w}_2, \mathsf{x}_2) \leftarrow \langle \mathcal{P}_1(\mathsf{pp}, \mathsf{x}_1, \mathsf{w}_1, \mathsf{aux}_1), \mathcal{V}_1(\mathsf{pp}, \mathsf{x}_1) \rangle$
   - $\mathcal{P}, \mathcal{V}$ interactively compute $(\mathsf{w}_3, \mathsf{x}_3) \leftarrow \langle \mathcal{P}_2(\mathsf{pp}, \mathsf{x}_2, \mathsf{w}_2, \mathsf{aux}_2), \mathcal{V}_2(\mathsf{pp}, \mathsf{x}_2) \rangle$
   - $\mathcal{P}$ outputs w_3 and $\mathcal{V}$ outputs x_3.
2. Let $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2 = (\mathsf{Gen}, \mathcal{P}_1, \mathcal{V}_1)$ and $\Pi_2 : \mathcal{R}_3 \to \mathcal{R}_4 = (\mathsf{Gen}, \mathcal{P}_2, \mathcal{V}_2)$ be reductions of knowledge. We define the parallel composition of Π_1, Π_2 denoted as $\Pi_p = \Pi_1 \times \Pi_2$ the tuple $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ where $\langle \mathcal{P}(\mathsf{pp}, (\mathsf{x}_1, \mathsf{x}_3), (\mathsf{w}_1, \mathsf{w}_3), \mathsf{aux} = (\mathsf{aux}_1, \mathsf{aux}_3)), \mathcal{V}(\mathsf{pp}, (\mathsf{x}_1, \mathsf{x}_3)) \rangle$ is the following protocol:
   - $\mathcal{P}, \mathcal{V}$ interactively compute $(\mathsf{w}_2, \mathsf{x}_2) \leftarrow \langle \mathcal{P}_1(\mathsf{pp}, \mathsf{x}_1, \mathsf{w}_1, \mathsf{aux}_1), \mathcal{V}_1(\mathsf{pp}, \mathsf{x}_1) \rangle$
   - $\mathcal{P}, \mathcal{V}$ interactively compute $(\mathsf{w}_4, \mathsf{x}_4) \leftarrow \langle \mathcal{P}_2(\mathsf{pp}, \mathsf{x}_3, \mathsf{w}_3, \mathsf{aux}_3), \mathcal{V}_2(\mathsf{pp}, \mathsf{x}_3) \rangle$
   - $\mathcal{P}$ outputs $(\mathsf{w}_2, \mathsf{w}_4)$ and $\mathcal{V}$ outputs $(\mathsf{x}_2, \mathsf{x}_4)$.

Definition 4.4.3. Let $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$ be ternary relations.

1. Let $\mathsf{NIRoK}_1 : \mathcal{R}_1 \to \mathcal{R}_2 = (\mathsf{Gen}, \mathsf{Reduce}_1, \mathsf{Verify}_1)$ and $\mathsf{NIRoK}_2 : \mathcal{R}_2 \to \mathcal{R}_3 = (\mathsf{Gen}, \mathsf{Reduce}_2, \mathsf{Verify}_2)$ be non-interactive reductions of knowledge. We define the sequential composition of $\mathsf{NIRoK}_1, \mathsf{NIRoK}_2$ as $\mathsf{NIRoK}_s = \mathsf{NIRoK}_2 \circ \mathsf{NIRoK}_1$ as the tuple $(\mathsf{Gen}, \mathsf{Reduce}, \mathsf{Verify})$ described in Fig. 4
2. Let $\mathsf{NIRoK}_1 : \mathcal{R}_1 \to \mathcal{R}_2 = (\mathsf{Gen}, \mathsf{Reduce}_1, \mathsf{Verify}_1)$ and $\mathsf{NIRoK}_2 : \mathcal{R}_3 \to \mathcal{R}_4 = (\mathsf{Gen}, \mathsf{Reduce}_2, \mathsf{Verify}_2)$ be reductions of knowledge. We define the parallel composition of $\mathsf{NIRoK}_1, \mathsf{NIRoK}_2$ denoted as $\mathsf{NIRoK}_p = \mathsf{NIRoK}_1 \times \mathsf{NIRoK}_2$ the tuple $(\mathsf{Gen}, \mathsf{Reduce}, \mathsf{Verify})$ described in Fig. 5.

The following theorem is proven in [KP23] for the interactive case. We also present an adaptation of their proof for the non-interactive case in Sec. A.1 of the supplementary material.

Theorem 4.4.1. *Let $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$ be ternary relations.*

1. *Let $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$ (resp. NIRoK_1) and $\Pi_2 : \mathcal{R}_2 \to \mathcal{R}_3$ (resp. NIRoK_2) be reductions of knowledge (resp. non-interactive reductions of knowledge). Then, the sequential composition $\Pi = \Pi_2 \circ \Pi_1$ (resp. $\mathsf{NIRoK} = \mathsf{NIRoK}_2 \circ \mathsf{NIRoK}_1$) is a reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_3$.*
2. *Let $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$ (resp. $\mathsf{NIRoK}_1 : \mathcal{R}_1 \to \mathcal{R}_2$) and $\Pi_2 : \mathcal{R}_3 \to \mathcal{R}_4$ (resp. $\mathsf{NIRoK}_2 : \mathcal{R}_3 \to \mathcal{R}_4$) be reductions of knowledge (resp. non-interactive reductions of knowledge). Then, the parallel composition $\Pi = \Pi_2 \times \Pi_1$ (resp. $\mathsf{NIRoK} = \mathsf{NIRoK}_2 \times \mathsf{NIRoK}_1$) is a reduction of knowledge from $\mathcal{R}_1 \times \mathcal{R}_3$ to $\mathcal{R}_2 \times \mathcal{R}_4$.*

$\mathsf{NIRoK}_s.\mathsf{Reduce}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{w}_1, (\mathsf{aux}_1, \mathsf{aux}_2))$

$\pi_1, \mathsf{x}_2, \mathsf{w}_2 \leftarrow \mathsf{NIRoK}_1.\mathsf{Reduce}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{w}_1, \mathsf{aux}_1)$
$\pi_2, \mathsf{x}_3, \mathsf{w}_3 \leftarrow \mathsf{NIRoK}_2.\mathsf{Reduce}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_2, \mathsf{w}_2, \mathsf{aux}_2)$
$\pi \leftarrow ((x_2, \pi_1), \pi_2)$

$\mathsf{NIRoK}_s.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_3, \pi)$

Parse $\pi := ((x_2, \pi_1), \pi_2)$
$b_1 \leftarrow \mathsf{NIRoK}_1.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_2, \pi_1)$
$b_2 \leftarrow \mathsf{NIRoK}_2.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_2, \mathsf{x}_3, \pi_2)$
Output $b_1 \wedge b_2$

Figure 4: Sequential composition of non-interactive reductions of knowledge.

$\mathsf{NIRoK}_p.\mathsf{Reduce}^{\rho(\cdot)}(\mathsf{pp}, (\mathsf{x}_1, \mathsf{x}_3), (\mathsf{w}_1, \mathsf{w}_3), (\mathsf{aux}_1, \mathsf{aux}_3))$

$\pi_1, \mathsf{x}_2, \mathsf{w}_2 \leftarrow \mathsf{NIRoK}_1.\mathsf{Reduce}(\mathsf{pp}, \mathsf{x}_1, \mathsf{w}_1, \mathsf{aux}_1)$
$\pi_2, \mathsf{x}_4, \mathsf{w}_4 \leftarrow \mathsf{NIRoK}_2.\mathsf{Reduce}(\mathsf{pp}, \mathsf{x}_3, \mathsf{w}_3, \mathsf{aux}_3)$
$\pi \leftarrow (\pi_1, \pi_2)$

$\mathsf{NIRoK}_p.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, (\mathsf{x}_1, \mathsf{x}_3), (\mathsf{x}_2, \mathsf{x}_4), \pi)$

Parse $\pi := (\pi_1, \pi_2)$
$b_1 \leftarrow \mathsf{NIRoK}_1.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_2, \pi_1)$
$b_2 \leftarrow \mathsf{NIRoK}_2.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_3, \mathsf{x}_4, \pi_2)$
Output $b_1 \wedge b_2$

Figure 5: Parallel composition of non-interactive reductions of knowledge.

Finally, we present two "trivial", yet useful reductions of knowledge that will allow us to express complex protocols in the language of reductions of knowledge. The first one reduces a statement to itself and the other two reduce a statement in $\mathcal{R}$ to a statement in either $\mathcal{R} \times \mathcal{R}'$ or $\mathcal{R}' \times \mathcal{R}$ for some $\mathcal{R}'$. All act in some sense as an identity reduction since the outputs always contain the initial statement.

Definition 4.4.4. Let $\mathcal{R}, \mathcal{R}'$ be ternary relations.

- The protocol $\mathsf{ID}_{\mathcal{R}} : \mathcal{R} \to \mathcal{R}$ is the identity reduction of knowledge where $\mathcal{P}$ and $\mathcal{V}$ with corresponding inputs $(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux})$ and $(\mathsf{pp}, \mathsf{x})$ trivially interact, that is they exchange no messages, and simply output w and x respectively.
- The protocol $\mathsf{AUX}_{0,\mathcal{R},\mathcal{R}'} : \mathcal{R} \to \mathcal{R} \times \mathcal{R}'$ is the auxiliary reduction of knowledge where $\mathcal{P}$ and $\mathcal{V}$ with corresponding inputs $(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux} = (\mathsf{x}', \mathsf{w}'))$ and $(\mathsf{pp}, \mathsf{x})$ interact as follows: $\mathcal{P}$ sends x' to $\mathcal{V}$ and outputs $(\mathsf{w}, \mathsf{w}')$ and $\mathcal{V}$ outputs $(\mathsf{x}, \mathsf{x}')$.
- The protocol $\mathsf{AUX}_{1,\mathcal{R},\mathcal{R}'} : \mathcal{R}' \to \mathcal{R} \times \mathcal{R}'$ is the auxiliary reduction of knowledge where $\mathcal{P}$ and $\mathcal{V}$ with corresponding inputs $(\mathsf{pp}, \mathsf{x}', \mathsf{w}', \mathsf{aux} = (\mathsf{x}, \mathsf{w}))$ and $(\mathsf{pp}, \mathsf{x}')$ interact as follows: $\mathcal{P}$ sends x to $\mathcal{V}$ and outputs $(\mathsf{w}, \mathsf{w}')$ and $\mathcal{V}$ outputs $(\mathsf{x}, \mathsf{x}')$.

Both ID and AUX_b are trivially reductions of knowledge. We omit the subscripts when they are clear from context.

5 A Transformation for Amortized Multi-Verifier Reductions of Knowledge

Let $\Pi : \mathcal{R} \to \mathcal{R}'$ be an ℓ-round public coin reduction of knowledge where the verifier sends the first message. We present in Fig. 6 the transformation MT that takes $\Pi : \mathcal{R} \to \mathcal{R}'$ and a constant $\kappa \in \mathbb{N}$ and returns a new reduction of knowledge $\mathsf{MT}(\Pi) = \widetilde{\Pi} : \mathcal{R} \to \mathcal{R}'$.

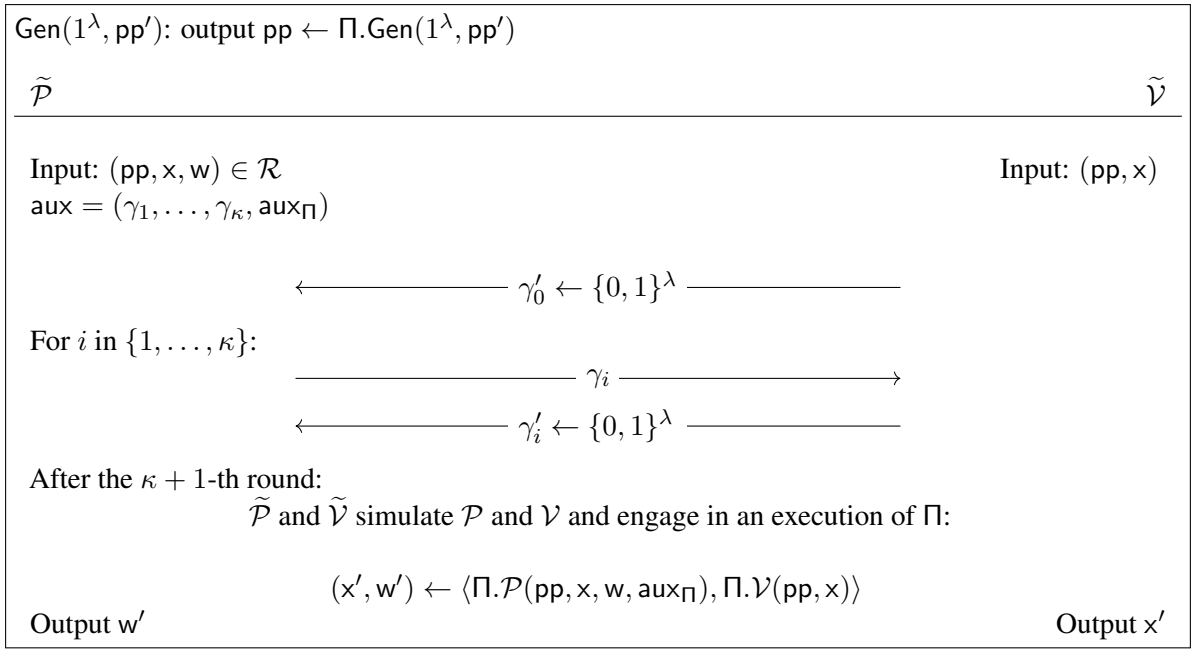

Figure 6: Transformation MT that takes $\kappa \in \mathbb{N}$ and $\Pi : \mathcal{R} \to \mathcal{R}'$ and returns a new reduction of knowledge $\mathsf{MT}(\Pi, \kappa) = \widetilde{\Pi} : \mathcal{R} \to \mathcal{R}'$ that allows amortizing multiple executions of Π.

Theorem 5.1. *Let $\kappa \in \mathbb{N}$ be a constant and $\Pi : \mathcal{R} \to \mathcal{R}'$ be a public coin reduction of knowledge where the verifier sends the first message. Then, the transformed protocol $\widetilde{\Pi} = \mathsf{MT}(\Pi, \kappa)$ is also a reduction of knowledge.*

Proof. The theorem follows directly from the properties of Π. In particular, the messages exchanged during the first κ rounds are independent of the inner execution of Π and no check is imposed on the γ_i, γ'_i. □

We next present another transformation, this time to derive a non-interactive reduction of knowledge from $\Pi : \mathcal{R} \to \mathcal{R}'$. Our transformation takes as input a value $\mathbf{b} \in \{0,1\}^\kappa$ and Π and returns a non-interactive reduction of knowledge $\mathsf{FS}_{\mathsf{MT}}(\Pi, \mathbf{b}) = \widetilde{\mathsf{NIRoK}}_{\mathbf{b}} : \mathcal{R} \to \mathcal{R}'$. Roughly, the transformation works by having the prover compute the Merkle root of 2^κ statements (all of which except the input statement are given as auxiliary inputs) and then running an FS-transformed Π. Note that by construction the first message of the Π verifier will be computed as a random oracle query to the Merkle tree root. We present the construction in Fig. 7.

Theorem 5.2. *Let $\kappa \in \mathbb{N}$ be a constant and $\Pi : \mathcal{R} \to \mathcal{R}'$ be a public coin reduction of knowledge where the verifier sends the first message. Then, for all $\mathbf{b} \in \{0,1\}^\kappa$, construction $\widetilde{\mathsf{NIRoK}}_{\mathbf{b}} = \mathsf{FS}_{\mathsf{MT}}(\Pi, \mathbf{b})$ of Fig 7 is a non-interactive reduction of knowledge from $\mathcal{R}$ to $\mathcal{R}'$. Furthermore,*

- *the time complexity of $\widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Verify}^{\rho(\cdot)}$ is the same as the time complexity of $\mathsf{NIRoK}.\mathsf{Verify}^{\rho(\cdot)}$ and an additional $\kappa + 1$ random oracle queries,*

- *the proof size* $|\widetilde{\pi}|$ *of* $\widetilde{\mathsf{NIRoK}}_{\mathbf{b}}$ *is* $|\pi| + \kappa\lambda$ *where* $|\pi|$ *is the proof size of* NIRoK,
- *there exists an amortized proving algorithm* AmortizedReduce *that takes as input* pp, $\{\mathsf{x}_{\mathbf{b}}, \mathsf{w}_{\mathbf{b}}, \mathsf{aux}_{\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa}$ *and outputs* $\{\pi_{\mathbf{b}}, \mathsf{x}'_{\mathbf{b}}, \mathsf{w}'_{\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa}$ *such that for all* $\mathbf{b} \in \{0,1\}^\kappa$*:*

$$\pi_{\mathbf{b}}, \mathsf{x}'_{\mathbf{b}}, \mathsf{w}'_{\mathbf{b}} = \widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Reduce}^{\boldsymbol{\rho}(\cdot)}_{\mathbf{b}}(\mathsf{pp}, \mathsf{x}_{\mathbf{b}}, \mathsf{w}_{\mathbf{b}}, \mathsf{aux}_{\mathbf{b}})$$

The running time of AmortizedReduce *is* $2^\kappa \cdot t_{\mathcal{P}}$ *and an additional* $\mathcal{O}(2^\kappa)$ *random oracle queries, where* $t_{\mathcal{P}}$ *is the time complexity of* $\mathsf{FS}(\Pi).\mathsf{Reduce}$.

$\mathsf{Gen}_{\mathbf{b}}(1^\lambda, \mathsf{pp}')$

output $\mathsf{pp} \leftarrow \Pi.\mathsf{Gen}(1^\lambda, \mathsf{pp}')$

$\mathsf{Reduce}^{\boldsymbol{\rho}(\cdot)}_{\mathbf{b}}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux})$

Parse $\mathsf{aux} := (\{\mathsf{x}_{\mathbf{b}'}\}_{\mathbf{b}'\in\{0,1\}^\kappa\setminus\mathbf{b}}, \mathsf{aux}_\Pi)$ and set $\mathsf{x}_{\mathbf{b}} = \mathsf{x}$
$\mathsf{h}, \{\pi_{\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa} \leftarrow \mathsf{MT}.\mathsf{Prove}^{\boldsymbol{\rho}(\cdot)}(\{\mathsf{pp}, \mathsf{x}_{\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa})$
Let $\boldsymbol{\rho}_\Pi := (\rho_{\kappa+1}, \ldots, \rho_{\kappa+\ell})$
Simulate an execution of $(\mathsf{x}', \mathsf{w}') \leftarrow \langle \mathcal{P}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux}_\Pi), \mathcal{V}(\mathsf{pp}, \mathsf{x})\rangle$
where the verifier messages $\gamma'_1, \ldots, \gamma'_\ell$ are computed as:

$$\gamma'_i \leftarrow \begin{cases} \rho_{\Pi,i}(\mathsf{h}), & \text{if } i = 1 \\ \rho_{\Pi,i}(\gamma'_{i-1}, \gamma_{\kappa+i}), & \text{otherwise} \end{cases}$$

Define $\pi := (\pi_{\mathbf{b}}, \gamma_1, \ldots, \gamma_{\kappa+\ell})$ and output $(\mathsf{x}', \mathsf{w}', \pi)$

$\mathsf{Verify}^{\boldsymbol{\rho}(\cdot)}_{\mathbf{b}}(\mathsf{pp}, \mathsf{x}, \mathsf{x}', \pi)$

Parse $\pi := (\pi_{\mathbf{b}}, \gamma_1, \ldots, \gamma_{\kappa+\ell})$ and $\mathbf{b} := (b_1, \ldots, b_\kappa)$
For $i \in \{1, \ldots, \kappa\}$ define:

$$\rho'_i(x) = \begin{cases} \rho_i(x_\lambda, \ldots, x_{2\lambda-1}, x_0, \ldots, x_{\lambda-1}), & \text{if } x \in \{0,1\}^{2\lambda} \text{ and } b_i = 1 \\ \rho_i(x), & \text{otherwise} \end{cases}$$

Define:

$$\boldsymbol{\rho}_{\mathbf{b}}(\cdot) = (\rho_{\mathbf{b},0}(\cdot), \ldots, \rho_{\mathbf{b},\kappa+\ell}(\cdot)) = \begin{cases} \rho'_i(\cdot), & \text{if } i \in \{1, \ldots, \kappa\} \\ \rho_i(\cdot), & \text{otherwise} \end{cases}$$

Output $b \leftarrow \widetilde{\mathsf{NIRoK}}^{\boldsymbol{\rho}_{\mathbf{b}}(\cdot)}.\mathsf{Verify}(\mathsf{pp}, \mathsf{x}, \mathsf{x}', \pi)$

Figure 7: Transformation $\mathsf{FS}_{\mathsf{MT}}$ that takes $\mathbf{b} \in \{0,1\}^\kappa$ and $\Pi : \mathcal{R} \to \mathcal{R}'$ and returns a non-interactive reduction of knowledge $\mathsf{FS}_{\mathsf{MT}}(\Pi, \mathbf{b}) = \widetilde{\mathsf{NIRoK}}_{\mathbf{b}} : \mathcal{R} \to \mathcal{R}'$. $\widetilde{\mathsf{NIRoK}}$ is the FS transformed protocol derived from $\widetilde{\Pi} = \mathsf{MT}(\Pi, \kappa)$.

Proof. Let $\Pi : \mathcal{R} \to \mathcal{R}'$ be a reduction of knowledge. Fix a constant $\kappa \in \mathbb{N}$ and $\mathbf{b} \in \{0,1\}^\kappa$. We start by showing that $\widetilde{\mathsf{NIRoK}}_{\mathbf{b}} = \mathsf{FS}_{\mathsf{MT}}(\Pi, \mathbf{b})$ is a non-interactive reduction of knowledge.

Completeness. Fix $\lambda \in \mathbb{N}$ and $\mathsf{pp}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$. Let $\mathsf{pp} \leftarrow \Pi.\mathsf{Gen}(1^\lambda, \mathsf{pp}')$, $(\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$ and let $\mathsf{aux} = (\{\mathsf{x}_{\mathbf{b}'}\}_{\mathbf{b}'\leftarrow\{0,1\}^\kappa\setminus\mathbf{b}}, \mathsf{aux}_\Pi)$ be the auxiliary input. We show that for

$$(\mathsf{x}', \mathsf{w}', \pi) \leftarrow \widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Reduce}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux})$$

it holds that $(\mathsf{pp}, \mathsf{x}', \mathsf{w}') \in \mathcal{R}$ and $\widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{x}', \pi) = 1$ except with negligible probability. First, define $\mathsf{aux}_{\mathbf{b}} = (\gamma_1, \ldots, \gamma_\kappa, \mathsf{aux}_\Pi)$ where $(\gamma_1, \ldots, \gamma_\kappa)$ is the $\mathbf{b}$-th Merkle proof computed during the execution of $\widetilde{\mathsf{NIRoK}}_{\mathbf{b}}$. Let $\boldsymbol{\rho}_{\mathbf{b}}$ be the random oracles defined as in $\widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Verify}^{\rho(\cdot)}_{\mathbf{b}}$. We claim that for these values the distributions

$$\left\{\widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Reduce}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux})\right\}, \left\{\widetilde{\mathsf{NIRoK}}.\mathsf{Reduce}^{\rho_{\mathbf{b}}(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux}_{\mathbf{b}})\right\}$$

are perfectly indistinguishable, where the distributions are defined over the randomness of the two algorithms. We show this by showing that the random oracle queries corresponding to the verifier's messages in the two executions are the same. Denote with $\gamma'_0, \ldots, \gamma'_{\kappa+\ell}$ the simulated verifier messages in $\widetilde{\mathsf{NIRoK}}$ and denote with $\gamma'_{\mathbf{b},0}, \ldots, \gamma'_{\mathbf{b},\kappa+\ell}$ the simulated verifier messages in $\widetilde{\mathsf{NIRoK}}_{\mathbf{b}}$. We prove the claim $\gamma'_i = \gamma'_{\mathbf{b},i}$ by induction on $i \in \{0, \ldots, \kappa + \ell\}$.

- $i = 0$**:** The statement holds since in both executions the prover computes the first verifier message as $\rho_0(\mathsf{pp}, \mathsf{x})$.
- $i > 0$**:** Assume the statement is true for some $j < i$, that is, for all $j < i$ it holds that $\gamma'_j = \gamma'_{\mathbf{b},j}$. We consider two cases. First, assume that $i \leq \kappa$. In both executions, the prover's i-message is γ_i which is the i-th element of the Merkle tree proof for the $\mathbf{b}$-th leaf. We have that

$$\gamma'_{\mathbf{b},i} = \rho_{\mathbf{b},i}(\gamma'_{\mathbf{b},i-1}, \gamma_i) = \rho_{\mathbf{b},i}(\gamma'_{i-1}, \gamma_i) = \begin{cases} \rho_i(\gamma_i, \gamma'_{i-1}), & \text{if } b_i = 1 \\ \rho_i(\gamma'_{i-1}, \gamma_i), & \text{otherwise} \end{cases} = \gamma'_i$$

  where the last equality holds since γ'_i is the i-th element of the $\mathbf{b}$-th Merkle tree path when computed with random oracles $\boldsymbol{\rho}$. For the case $i > \kappa$ the claim holds because in both cases an execution of

$$(\mathsf{x}', \mathsf{w}') \leftarrow \langle \Pi.\mathcal{P}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux}_\Pi), \Pi.\mathcal{V}(\mathsf{pp}, \mathsf{x})\rangle$$

  is simulated with the same random oracles and first messages $\gamma'_{\mathbf{b},\kappa} = \gamma'_\kappa$.

Since we have shown

$$\left\{\widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Reduce}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux})\right\} \equiv \left\{\widetilde{\mathsf{NIRoK}}.\mathsf{Reduce}^{\rho_{\mathbf{b}}(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, \mathsf{aux}_{\mathbf{b}})\right\}$$

and $\widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Verify}^{\rho}(\cdot)$ invokes $\widetilde{\mathsf{NIRoK}}.\mathsf{Verify}^{\rho_{\mathbf{b}}}(\cdot)$, completeness follows by completeness of $\widetilde{\mathsf{NIRoK}}$.

Knowledge Soundness. Fix $\lambda \in \mathbb{N}$, $\mathsf{pp}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$, constant $\kappa \in \mathbb{N}$ and $\mathbf{b} \in \{0,1\}^\kappa$. We show that for all expected polynomial time adversaries $\mathcal{A}_{\mathbf{b},1}, \mathcal{A}_{\mathbf{b},2}$, there exists an expected polynomial time extractor $\mathcal{E}_{\mathbf{b}}$ such that:

$$\Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R} & \boldsymbol{\rho} = \rho_0, \ldots, \rho_{\kappa+\ell} \leftarrow \mathcal{U}(\lambda) \\ & \mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ & \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}^{\boldsymbol{\rho}(\cdot)}_{\mathbf{b},1}(\mathsf{pp}) \\ & \mathsf{w} \leftarrow \mathcal{E}^{\boldsymbol{\rho}(\cdot)}_{\mathbf{b}}(\mathsf{pp}, \mathsf{st}) \end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{x}', \mathsf{w}') \in \mathcal{R}' \wedge & \boldsymbol{\rho} = \rho_0, \ldots, \rho_{\kappa+\ell} \leftarrow \mathcal{U}(\lambda) \\ \widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{x}', \pi) = 1 & \mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ & \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}^{\boldsymbol{\rho}(\cdot)}_{\mathbf{b},1}(\mathsf{pp}) \\ & \pi, \mathsf{x}', \mathsf{w}' \leftarrow \mathcal{A}^{\boldsymbol{\rho}(\cdot)}_{\mathbf{b},2}(\mathsf{pp}, \mathsf{st}) \end{array}\right] - \mathsf{negl}(\lambda)$$

We construct expected polynomial time adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ against knowledge soundness of $\widetilde{\mathsf{NIRoK}} = \mathsf{FS}(\mathsf{MT}(\Pi, \kappa))$ as follows: let $\mathbf{b} = (b_1, \ldots, b_\kappa)$ and define $\boldsymbol{\rho}_{\mathbf{b}} = (\rho'_{\mathbf{b},0}, \ldots, \rho'_{\mathbf{b},\kappa+\ell})$ where:

$$\rho'_{\mathbf{b},i}(x) = \begin{cases} \rho_i(x_\lambda, \ldots, x_{2\lambda-1}, x_0, \ldots, x_{\lambda-1}), & \text{if } x \in \{0,1\}^{2\lambda},\ 1 \le i \le \kappa \text{ and } b_i = 1 \\ \rho_i(x), & \text{otherwise} \end{cases}$$

Note that given random oracles $\boldsymbol{\rho}$ we can simulate queries to $\boldsymbol{\rho}_{\mathbf{b}}$ by swapping the inputs and that $\boldsymbol{\rho}, \boldsymbol{\rho}'$ are identically distributed. We define $\mathcal{A}_1, \mathcal{A}_2$ as follows:

- On input $\mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') = \Pi.\mathsf{Gen}(1^\lambda, \mathsf{pp})$, $\mathcal{A}_1^{\rho(\cdot)}(\mathsf{pp})$ outputs $\mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}_{\mathbf{b},1}^{\rho_{\mathbf{b}}(\cdot)}(\mathsf{pp})$ and
- On input $\mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') = \Pi.\mathsf{Gen}(1^\lambda, \mathsf{pp})$ and st output by $\mathcal{A}_1^{\rho(\cdot)}$, $\mathcal{A}_2^{\rho}(\mathsf{pp}, \mathsf{st})$ outputs $\pi, \mathsf{x}', \mathsf{w}' \leftarrow \mathcal{A}_{\mathbf{b},2}^{\rho_{\mathbf{b}}(\cdot)}(\mathsf{pp}, \mathsf{st})$.

We have that

$$\begin{aligned}
&\Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{x}', \mathsf{w}') \in \mathcal{R}' \wedge & \boldsymbol{\rho} = \rho_0, \ldots, \rho_{\kappa+\ell} \leftarrow \mathcal{U}(\lambda) \\ \widetilde{\mathsf{NIRoK}}.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{x}', \pi) = 1 & \mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ & \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}_1^{\rho(\cdot)}(\mathsf{pp}) \\ & \pi, \mathsf{x}', \mathsf{w}' \leftarrow \mathcal{A}_2^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st}) \end{array}\right] \\
&= \Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{x}', \mathsf{w}') \in \mathcal{R}' \wedge & \boldsymbol{\rho} = \rho_0, \ldots, \rho_{\kappa+\ell} \leftarrow \mathcal{U}(\lambda) \\ \widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Verify}^{\rho_{\mathbf{b}}(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{x}', \pi) = 1 & \mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ & \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}_{\mathbf{b},1}^{\rho_{\mathbf{b}}(\cdot)}(\mathsf{pp}) \\ & \pi, \mathsf{x}', \mathsf{w}' \leftarrow \mathcal{A}_{\mathbf{b},2}^{\rho_{\mathbf{b}}(\cdot)}(\mathsf{pp}, \mathsf{st}) \end{array}\right] \\
&= \Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{x}', \mathsf{w}') \in \mathcal{R}' \wedge & \boldsymbol{\rho}_{\mathbf{b}} = \rho_{\mathbf{b},0}, \ldots, \rho_{\mathbf{b}\kappa+\ell} \leftarrow \mathcal{U}(\lambda) \\ \widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Verify}^{\rho_{\mathbf{b}}(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{x}', \pi) = 1 & \mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ & \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}_{\mathbf{b},1}^{\rho_{\mathbf{b}}(\cdot)}(\mathsf{pp}) \\ & \pi, \mathsf{x}', \mathsf{w}' \leftarrow \mathcal{A}_{\mathbf{b},2}^{\rho_{\mathbf{b}}(\cdot)}(\mathsf{pp}, \mathsf{st}) \end{array}\right]
\end{aligned}$$

where

- the first equation holds by construction of $\widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Verify}^{\rho_{\mathbf{b}}}(\cdot)$: it accepts iff $\widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Verify}^{\rho}(\cdot)$ accepts, and
- the second equation holds because the random oracles $\boldsymbol{\rho}_{\mathbf{b}}$ are identically distributed to $\kappa + \ell + 1$ random oracles sampled by $\mathcal{U}(\lambda)$.

Finally, since $\widetilde{\mathsf{NIRoK}}$ is a reduction of knowledge, there exists an extractor $\mathcal{E}$ s.t.

$$\Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R} & \boldsymbol{\rho} = \rho_0, \ldots, \rho_{\kappa+\ell} \leftarrow \mathcal{U}(\lambda) \\ & \mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ & \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}_1^{\rho(\cdot)}(\mathsf{pp}) \\ & \mathsf{w} \leftarrow \mathcal{E}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st}) \end{array}\right] \ge$$

$$\Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{x}', \mathsf{w}') \in \mathcal{R}' \wedge & \boldsymbol{\rho} = \rho_0, \ldots, \rho_{\kappa+\ell} \leftarrow \mathcal{U}(\lambda) \\ \widetilde{\mathsf{NIRoK}}.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}, \mathsf{x}', \pi) = 1 & \mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ & \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}_1^{\rho(\cdot)}(\mathsf{pp}) \\ & \pi, \mathsf{x}', \mathsf{w}' \leftarrow \mathcal{A}_2^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st}) \end{array}\right] - \mathsf{negl}(\lambda)$$

We define the extractor $\mathcal{E}^{\rho}_{\mathbf{b}}$ that on input pp, st, where st is the output of $\mathcal{A}^{\rho}_{\mathbf{b},1}(\mathsf{pp})$, it outputs

$$\mathsf{w}' \leftarrow \mathcal{E}^{\rho_{\mathbf{b}}(\cdot)}(\mathsf{pp}, \mathsf{st})$$

We have that

$$\begin{aligned}
&\Pr\left[(\mathsf{pp},\mathsf{x},\mathsf{w}) \in \mathcal{R} \;\middle|\; \begin{array}{r} \boldsymbol{\rho} = \rho_0, \ldots, \rho_{\kappa+\ell} \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathbf{b},1}(\mathsf{pp}) \\ \mathsf{w} \leftarrow \mathcal{E}^{\rho(\cdot)}_{\mathbf{b}}(\mathsf{pp}, \mathsf{st}) \end{array} \right] \\
&\geq \Pr\left[(\mathsf{pp},\mathsf{x},\mathsf{w}) \in \mathcal{R} \;\middle|\; \begin{array}{r} \boldsymbol{\rho} = \rho_0, \ldots, \rho_{\kappa+\ell} \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}^{\boldsymbol{\rho}_{\mathbf{b}}(\cdot)}_{1}(\mathsf{pp}) \\ \mathsf{w} \leftarrow \mathcal{E}^{\boldsymbol{\rho}_{\mathbf{b}}(\cdot)}(\mathsf{pp}, \mathsf{st}) \end{array} \right] \\
&\geq \Pr\left[(\mathsf{pp},\mathsf{x},\mathsf{w}) \in \mathcal{R} \;\middle|\; \begin{array}{r} \boldsymbol{\rho}_{\mathbf{b}} = \rho_{\mathbf{b}0}, \ldots, \rho_{\mathbf{b},\kappa+\ell} \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}^{\boldsymbol{\rho}_{\mathbf{b}}(\cdot)}_{1}(\mathsf{pp}) \\ \mathsf{w} \leftarrow \mathcal{E}^{\boldsymbol{\rho}_{\mathbf{b}}(\cdot)}(\mathsf{pp}, \mathsf{st}) \end{array} \right] \\
&\geq \Pr\left[\begin{array}{c} (\mathsf{pp},\mathsf{x}',\mathsf{w}') \in \mathcal{R}' \wedge \\ \widetilde{\mathsf{NIRoK}}.\mathsf{Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp},\mathsf{x},\mathsf{x}',\pi) = 1 \end{array} \;\middle|\; \begin{array}{r} \boldsymbol{\rho}_{\mathbf{b}} = \rho_{\mathbf{b},0}, \ldots, \rho_{\mathbf{b},\kappa+\ell} \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}^{\boldsymbol{\rho}_{\mathbf{b}}(\cdot)}_{1}(\mathsf{pp}) \\ \pi, \mathsf{x}', \mathsf{w}' \leftarrow \mathcal{A}^{\boldsymbol{\rho}_{\mathbf{b}}(\cdot)}_{2}(\mathsf{pp}, \mathsf{st}) \end{array} \right] - \mathsf{negl}(\lambda) \\
&\geq \Pr\left[\begin{array}{c} (\mathsf{pp},\mathsf{x}',\mathsf{w}') \in \mathcal{R}' \wedge \\ \widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp},\mathsf{x},\mathsf{x}',\pi) = 1 \end{array} \;\middle|\; \begin{array}{r} \boldsymbol{\rho} = \rho_0, \ldots, \rho_{\kappa+\ell} \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}, \mathsf{st} \leftarrow \mathcal{A}^{\boldsymbol{\rho}(\cdot)}_{\mathbf{b},1}(\mathsf{pp}) \\ \pi, \mathsf{x}', \mathsf{w}' \leftarrow \mathcal{A}^{\boldsymbol{\rho}(\cdot)}_{\mathbf{b},2}(\mathsf{pp}, \mathsf{st}) \end{array} \right] - \mathsf{negl}(\lambda)
\end{aligned}$$

and therefore the transformation $\mathsf{FS}_{\mathsf{MT}}$ is knowledge sound for all $\mathbf{b} \in \{0,1\}^\kappa$.

Finally, the efficiency claims for proof size and verification time are immediate by construction; it is enough to note that the $\widetilde{\mathsf{NIRoK}}_{\mathbf{b}}.\mathsf{Verify}$ takes as proof π which consists of (1) a Merkle path proof of size κ and (2) a proof of $\mathsf{FS}(\Pi)$ and its verification consists of recomputing the Merkle path using the proof and invoking the verifier $\mathsf{FS}(\Pi)$. For the $\mathsf{AmortizedReduce}$ algorithm, it is enough to note that for all $\mathbf{b} \in \{0,1\}^\kappa$, the first part of the computation, i.e. computing the Merkle tree and all the proofs, is the same for all $\mathbf{b}$; therefore, it needs to be performed only once. The $\mathsf{AmortizedReduce}$ algorithm simply performs this computation and simulates the execution of Π for each of the 2^κ inputs. Since these computations are done via querying the random oracles $\mathcal{O}(2^\kappa)$ times, the total time of $\mathsf{AmortizedReduce}$ is $\mathcal{O}(2^\kappa) + 2^\kappa \cdot t_{\mathcal{P}}$. □

Remark 5.1. One can also reduce the rounds by (1) omitting the first message γ'_0 (translating to hashing the statement and then computing the Merkle tree of the hashes) and (2) directly use γ'_κ as the first message of $\Pi.\mathcal{V}$ instead of getting a fresh message γ'_κ and similarly adapting the corresponding transformations. We choose to have the additional rounds since this makes the presentation of our constructions cleaner.

6 Promise Local Folding Schemes

Definition 6.1. Let $K = 2^\kappa$ be a constant and $\mathcal{R}', \mathcal{R}$ be ternary relations. An $\mathcal{R}'$-Promise K-Local Folding Scheme Λ for $\mathcal{R}$ is a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{LProve}, \mathsf{LVerify})$ with access to random oracles $\rho_1, \ldots, \rho_\ell$ such that

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}')$: on input the security parameter λ and $\mathsf{pp}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$ outputs parameters pp.
- $(\{\pi_{\mathbf{b}}, \mathsf{x}'_{\mathbf{b}}, \mathsf{w}'_{\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa}, \mathsf{x}, \mathsf{w}) \leftarrow \Lambda.\mathsf{LProve}^{\rho(\cdot)}\left(\mathsf{pp}, \{\mathsf{x}_{\mathbf{b}}, \mathsf{w}_{\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa}\right)$: takes as input parameters pp and $K = 2^\kappa$ statement/witness pairs $\mathsf{x}_{\mathbf{b}}, \mathsf{w}_{\mathbf{b}}$ (that supposedly satisfy $(\mathsf{pp}, \mathsf{x}_{\mathbf{b}}, \mathsf{w}_{\mathbf{b}}) \in \mathcal{R}$) and outputs a fresh statement/witness pair (x, w) (that supposedly satisfies $(\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$) and K statement/witness pairs $\{\mathsf{x}'_{\mathbf{b}}, \mathsf{w}'_{\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa}$ (each supposedly satisfying $(\mathsf{pp}', \mathsf{x}'_{\mathbf{b}}, \mathsf{w}'_{\mathbf{b}}) \in \mathcal{R}'$) and corresponding proofs $\{\pi_{\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa}$.
- $0/1 \leftarrow \Lambda.\mathsf{LVerify}^{\rho(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \mathsf{x}, \mathsf{x}'_{\mathbf{b}}, \pi_{\mathbf{b}})$: takes as input parameters pp, statements $\mathsf{x}_{\mathbf{b}}$, x and $\mathsf{x}'_{\mathbf{b}}$ and a proof $\pi_{\mathbf{b}}$, and outputs 0 or 1

If the promise relation $\mathcal{R}'$ is the trivial relation that is always true, we refer to the scheme as a (plain) K-local folding scheme.

Completeness. For all $\mathbf{b} \in \{0,1\}^\kappa$, $\lambda \in \mathbb{N}, \mathsf{pp}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$ and PPT algorithms $\mathcal{A}$

$$\Pr\left[\begin{array}{c|r} \{q_{\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa} \subseteq \mathcal{R} & \rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \Downarrow & \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ b = 1 & \{\mathsf{x}_{\mathbf{b}}, \mathsf{w}_{\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa} \leftarrow \mathcal{A}^{\rho(\cdot)}(\mathsf{pp}) \\ \wedge(\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R} & \{\pi_{\mathbf{b}}, \mathsf{x}'_{\mathbf{b}}, \mathsf{w}'_{\mathbf{b}}\}_{\mathbf{b}}, \mathsf{x}, \mathsf{w} \leftarrow \Lambda.\mathsf{LProve}^{\rho(\cdot)}(\mathsf{pp}, \{\mathsf{x}_{\mathbf{b}}, \mathsf{w}_{\mathbf{b}}\}_{\mathbf{b}}) \\ \wedge(\mathsf{pp}', \mathsf{x}'_{\mathbf{b}}, \mathsf{w}'_{\mathbf{b}}) \in \mathcal{R}' & \{q_{\mathbf{b}} = (\mathsf{pp}, \mathsf{x}_{\mathbf{b}}, \mathsf{w}_{\mathbf{b}})\}_{\mathbf{b}\in\{0,1\}^\kappa} \\ & b \leftarrow \Lambda.\mathsf{LVerify}^{\rho(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \mathsf{x}, \mathsf{x}'_{\mathbf{b}}, \pi_{\mathbf{b}}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

Knowledge soundness. For all $\mathbf{b} \in \{0,1\}^\kappa$, $\lambda \in \mathbb{N}, \mathsf{pp}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$ and expected polynomial time adversaries $\mathcal{A}_1, \mathcal{A}_2$, there exists an expected polynomial time extractor $\mathcal{E}$ such that:

$$\Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{x}_{\mathbf{b}}, \mathsf{w}_{\mathbf{b}}) \in \mathcal{R} & \rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ & \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ & \mathsf{x}_{\mathbf{b}}, \mathsf{st} \leftarrow \mathcal{A}_1^{\rho(\cdot)}(\mathsf{pp}) \\ & \mathsf{w}_{\mathbf{b}} \leftarrow \mathcal{E}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_{\mathbf{b}}, \mathsf{st}) \end{array}\right] \geq$$

$$\Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R} \wedge & \rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ (\mathsf{pp}', \mathsf{x}'_{\mathbf{b}}, \mathsf{w}'_{\mathbf{b}}) \in \mathcal{R}' \wedge & \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \Lambda.\mathsf{LVerify}^{\rho(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \mathsf{x}, \mathsf{x}'_{\mathbf{b}}, \pi_{\mathbf{b}}) = 1 & \mathsf{x}_{\mathbf{b}}, \mathsf{st} \leftarrow \mathcal{A}_1^{\rho(\cdot)}(\mathsf{pp}) \\ & \pi_{\mathbf{b}}, \mathsf{x}, \mathsf{w}, \mathsf{x}'_{\mathbf{b}}, \mathsf{w}'_{\mathbf{b}} \leftarrow \mathcal{A}_2^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st}) \end{array}\right] - \mathsf{negl}(\lambda)$$

Efficiency. For all $\kappa \in \mathbb{N}$, $\mathbf{b} \in \{0,1\}^\kappa$, $\lambda \in \mathbb{N}$ it should hold that $|\pi_{\mathbf{b}}| + |\mathsf{x}'_{\mathbf{b}}| = o(2^\kappa \cdot |\mathsf{x}|)$, namely, the inputs of the LVerify algorithm should be asymptotically smaller than the total size of the aggregated statements.

7 Local Folding Schemes Compiler

We present a compiler to construct a K-local folding scheme from a reduction of knowledge $\Pi : \mathcal{R}^2 \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}$ where PCC is a language capturing constraints over committed polynomials[4]. The $\mathcal{R}_{\mathsf{PCC}}$ relation naturally appears in many protocols that are derived from polynomial interactive oracle proofs [CHM+20; CFF+21] and handling such statements can lead to significant efficiency gains.

Roughly, our approach is similar to the compiler of [RZ23], but instead of using a "full" reduction $\Pi : \mathcal{R}^2 \to \mathcal{R}$, we *postpone* the proving of statements that occur during the reduction itself, and we amortize their proof *after* we have reduced everything to a single claim. In other words, we use a local folding scheme to (partially) verify the local folding proofs themselves, similarly to recursive proof composition techniques.

We first formally define the relation $\mathcal{R}_{\mathsf{PCC}}$. Let PC be a polynomial commitment scheme. We assume that the public parameters include the commitment and verification keys ck, vk for a degree bound D. We define the *polynomial commitment constraint* relation as

$$\mathcal{R}_{\mathsf{PCC}} = \left\{ \begin{array}{c} \mathsf{pp} \\ (\mathbf{c}, \mathbf{d}, \mathbf{Q}) \\ \mathbf{p}(X) \end{array} \middle| \begin{array}{c} \{\mathsf{c}_i = \mathsf{PC.Commit}(\mathsf{ck}, p_i(X), d_i)\}_{i=1}^n, \\ \left\{p_i(X) \in \mathbb{F}^{<d_i}[X], d_i \leq D\right\}_{i=1}^n \\ \{Q_j \in \mathbb{F}[X, Y_1, \ldots, Y_n] \text{ and } Q_j(X, \mathbf{p}(X)) = \mathbf{0}\}_{j=1}^{\ell} \end{array} \right\}$$

Assuming common parameters pp, we can view many statements as a single statement by simply concatenating the commitments and the constraints. We also denote with $\mathcal{R}_{\mathsf{PCC\text{-}D}}$ the same relation but with no degree constraints, namely,

$$\mathcal{R}_{\mathsf{PCC\text{-}D}} = \left\{ \begin{array}{c} \mathsf{pp} \\ (\mathbf{c}, \mathbf{Q}) \\ \mathbf{p}(X) \end{array} \middle| \begin{array}{c} \{\mathsf{c}_i = \mathsf{PC.Commit}(\mathsf{ck}, p_i(X), D)\}_{i=1}^n, \\ \left\{p_i(X) \in \mathbb{F}^{<D}[X]\right\}_{i=1}^n \\ \{Q_j \in \mathbb{F}[X, Y_1, \ldots, Y_n] \text{ and } Q_j(X, \mathbf{p}(X)) = \mathbf{0}\}_{j=1}^{\ell} \end{array} \right\}$$

There exists a simple reduction of knowledge $\Pi_{\mathsf{D}} : \mathcal{R}_{\mathsf{PCC}} \to \mathcal{R}_{\mathsf{PCC\text{-}D}}$ which essentially consists of the prover sending "scaled" polynomials to enforce the degree constraints. We present the construction in Fig. 8.

We also define a relevant relation $\mathcal{R}_{\mathsf{PCO}}$ capturing the statement "c is a commitment to $p(X)$ of degree less than D and $p(x) = y$".

$$\mathcal{R}_{\mathsf{PCO}} = \left\{ \mathsf{pp}, (\mathsf{c}, x, y), p(X) \;\middle|\; \begin{array}{c} \mathsf{c} = \mathsf{PC.Commit}(\mathsf{ck}, p(X), D), \\ p(X) \in \mathbb{F}^{<D}[X] \\ p(x) = y \end{array} \right\}$$

and the relation $\mathcal{R}_{\mathsf{PCO}(x)} = \left\{ \mathsf{pp}, (\mathsf{c}, y), p(X) \;\middle|\; (\mathsf{c}, x, y) \in \mathcal{R}_{\mathsf{PCO}} \right\}$ which is the same as $\mathcal{R}_{\mathsf{PCO}}$ for a hard-coded point x. We next use two reductions of knowledge $\Pi_{\mathsf{PC}} : \mathcal{R}_{\mathsf{PCC\text{-}D}} \to \mathcal{R}_{\mathsf{PCO}(x)}$ and $\Pi_{\mathsf{PCO}} : \mathcal{R}^{\ell}_{\mathsf{PCO}(x)} \to \mathcal{R}_{\mathsf{PCO}(x)}$ assuming a homomorphic PC. Essentially, they use (1) randomized testing to reduce polynomial constraints to polynomial commitment openings at the same point, and (2) batched openings of polynomial commitments at the same point. These techniques are extensively used when designing arguments of knowledge and their proofs are well known. The protocols are presented in Fig. 9 and Fig. 10.

We will later use the fact that when applying the transform $\mathsf{FS}_{\mathsf{MT}}(\Pi_{\mathsf{PC}})$ we have a reduction of knowledge $\widetilde{\mathsf{NIRoK}}_{\mathsf{PC}} : \mathcal{R}_{\mathsf{PCC\text{-}D}} \to \mathcal{R}_{\mathsf{PCO}(x)}$ since by construction of the transformation, the first message of the verifier –which is the same in all executions– defines the value x of the reduced statements.

[4]We only consider the case of univariate polynomials but the constructions can be naturally adapted for the multivariate case.

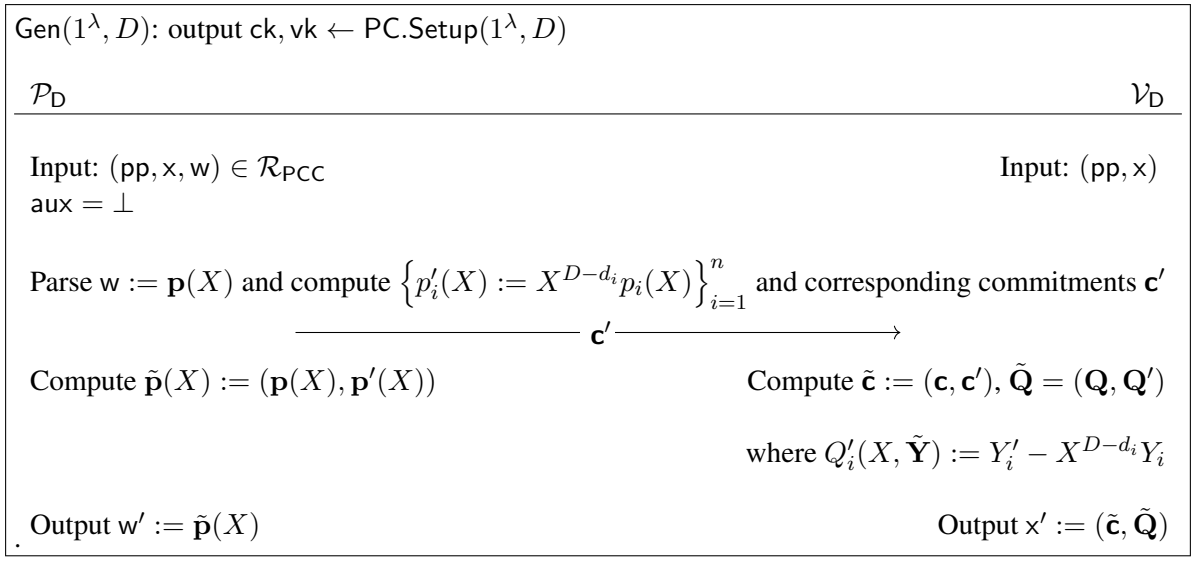

Figure 8: Reduction of Knowledge $\Pi_\mathsf{D} : \mathcal{R}_\mathsf{PCC} \to \mathcal{R}_{\mathsf{PCC\text{-}D}}$

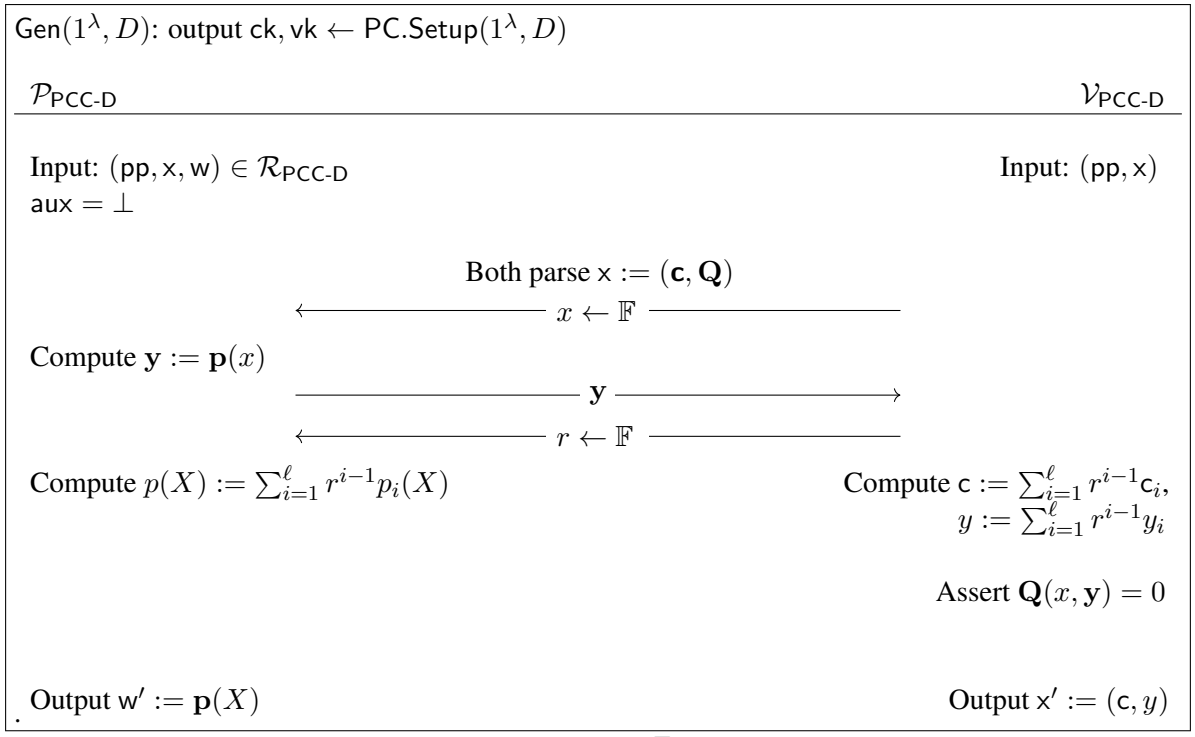

Figure 9: Reduction of Knowledge $\Pi_\mathsf{PC} : \mathcal{R}_{\mathsf{PCC\text{-}D}} \to \mathcal{R}_{\mathsf{PCO}(x)}$

7.1 $\mathcal{R}_\mathsf{PCC}$-Promise Local Folding Scheme Compiler

We next define the compiler to the $\mathcal{R}_\mathsf{PCC}$-Promise Local Folding Scheme. Our starting point is a non-interactive "partial" reduction of knowledge $\mathsf{NIRoK} : \mathcal{R}^2 \to \mathcal{R} \times \mathcal{R}_\mathsf{PCC}$. Our compiler applies NIRoK iteratively in a tree-like fashion similar to [RZ23]. We do not handle the $\mathcal{R}_\mathsf{PCC}$ statements at this point, we just collect them with the aim of proving them later. We define the construction inductively as a series of constructions $\Lambda'_1, \ldots, \Lambda'_\kappa$. The latter is presented in Fig. 11.

Theorem 7.1.1. *Let* NIRoK *be a non-interactive reduction of knowledge* $\mathcal{R}^2 \to \mathcal{R} \times \mathcal{R}_\mathsf{PCC}$ *in the random oracle model. Then for any constant* $K = 2^\kappa \in \mathbb{N}$*, the construction* $\Lambda'_\kappa = \mathsf{L}_\mathsf{PCC}(\mathsf{NIRoK}, \kappa)$ *is an* $\mathcal{R}^\kappa_\mathsf{PCC}$-

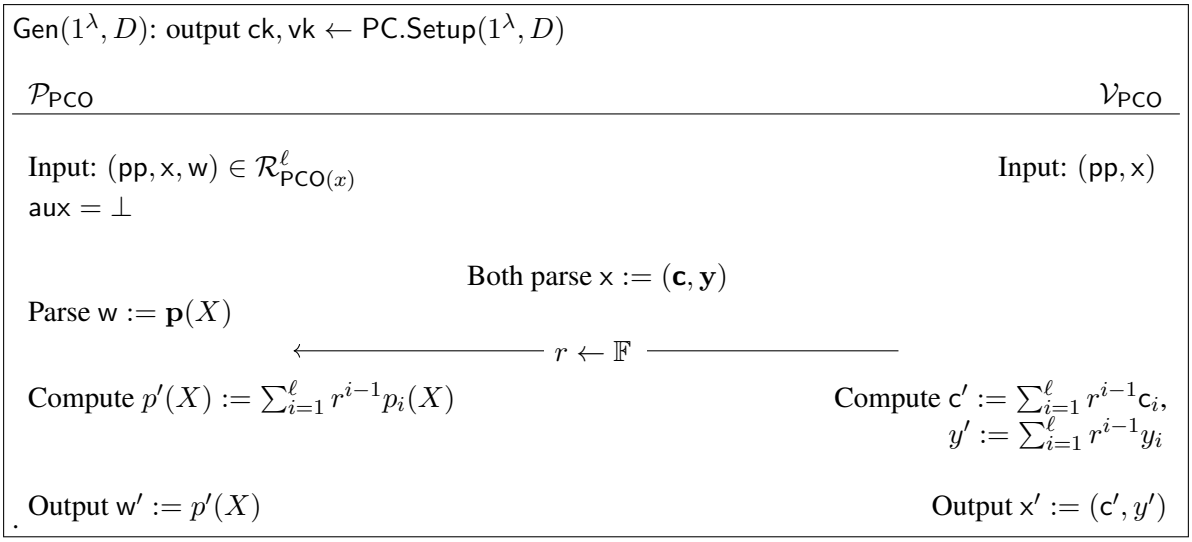

Figure 10: Reduction of Knowledge $\Pi_{\mathsf{PCO}(x)} : \mathcal{R}^{\ell}_{\mathsf{PCO}(x)} \to \mathcal{R}_{\mathsf{PCO}(x)}$

promise K-local folding scheme for $\mathcal{R}$ in the random oracle model. Furthermore, let $t_{\mathcal{P}}(\Pi), t_{\mathcal{V}}(\Pi), \pi(\Pi)$ the proving time, verifying time and proof size of $\mathsf{FS}(\Pi)$ and let $|\mathcal{R}|$ be the size of a statement of $\mathcal{R}$[5]. Then Λ'_κ has the following efficiency characteristics:

- *prover time is $\mathcal{O}(K \cdot t_{\mathcal{P}}(\mathsf{NIRoK}))$, verifier time is $\mathcal{O}(\kappa \cdot t_{\mathcal{V}}(\mathsf{NIRoK}))$, and proof size is $\mathcal{O}(\kappa \cdot (\pi(\mathsf{NIRoK}) + |\mathcal{R}|))$.*

Before proving the theorem that the above compiler is a local folding scheme, we prove a lemma that essentially states that the verification algorithm $\mathsf{LVerify}$ behaves as a specially crafted reduction of knowledge $\mathsf{NIRoK}_\kappa : \mathcal{R} \to \mathcal{R} \times \mathcal{R}^{\kappa}_{\mathsf{PCC}}$. Using this lemma, we can directly reduce knowledge soundness of the local folding scheme to knowledge soundness of the reduction of knowledge, which follows directly from the composition properties of NIRoK.

Lemma 7.1.1. *Let $\mathsf{NIRoK} : \mathcal{R}^2 \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}$ be a non-interactive reduction of knowledge, $\kappa \in \mathbb{N}$ a constant and $b_1, \ldots, b_\kappa \in \{0,1\}$. Also for all $0 \leq \ell \leq \kappa$ let $\Lambda'_i = \mathsf{L}_{\mathsf{PCC}}(\mathsf{NIRoK}, i)$. Consider the following series of non-interactive reductions of knowledge defined recursively:*

$$\mathsf{NIRoK}_1 = \mathsf{NIRoK} \circ \mathsf{AUX}_{b_1,\mathcal{R},\mathcal{R}} : \mathcal{R} \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}$$
$$\mathsf{NIRoK}_\ell = (\mathsf{NIRoK} \times \mathsf{ID}) \circ (\mathsf{AUX}_{b_\ell,\mathcal{R},\mathcal{R}} \times \mathsf{ID}) \circ \mathsf{NIRoK}_{\ell-1} : \mathcal{R} \to \mathcal{R} \times \mathcal{R}^{\ell}_{\mathsf{PCC}}$$

Then, for all $\ell \in \{1, \ldots, \kappa\}$, there exists an efficient transformation T_ℓ mapping proofs of $\Lambda'_\ell.\mathsf{Verify}$ to proofs of NIRoK_ℓ s.t. for all random oracles $\rho_1, \ldots, \rho_\ell \in \mathcal{U}(\lambda)$:

$$\Lambda'_\ell.\mathsf{LVerify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \pi_{\mathbf{b}}) = 1 \Rightarrow \mathsf{NIRoK}_\ell.\mathsf{Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{x}_{\mathbf{b}}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \mathsf{T}_\ell(\pi_{\mathbf{b}})) = 1$$

Proof. We show this by induction on ℓ.

Base Case. Assume $(\mathsf{pp}, b_1, \mathsf{x}_{b_1}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},b_1}, \pi_{b_1})$ satisfies $\Lambda'_1.\mathsf{LVerify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, b, \mathsf{x}_b, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},b}, \pi_b) = 1$. Our goal is to transform the proof π_{b_1} to an accepting proof of $\mathsf{NIRoK}_1.\mathsf{Verify}^{\boldsymbol{\rho}(\cdot)}$ with the same input. Note that the latter must be of the form:

$$\pi' = (((\mathsf{x}_0, \mathsf{x}_1), \pi_{\mathsf{AUX}}), \pi)$$

[5]We implicitly assume that the statements of $\mathcal{R}$ have a fixed, a priori known size.

$\Lambda'_1.\mathsf{Gen}(1^\lambda, \mathsf{pp}')$

Output $\mathsf{pp} \leftarrow \mathsf{NIRoK.Gen}(1^\lambda, \mathsf{pp}')$

$\Lambda'_1.\mathsf{LProve}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, (\mathsf{x}_0, \mathsf{w}_0), (\mathsf{x}_1, \mathsf{w}_1))$

$\pi, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC}}), (\mathsf{w}, \mathsf{w}_{\mathsf{PCC}}) \leftarrow \mathsf{NIRoK.Reduce}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, (\mathsf{x}_0, \mathsf{w}_0), (\mathsf{x}_1, \mathsf{w}_1))$
$\pi_0 := (\mathsf{x}_0, \mathsf{x}_1, \pi)$, $\pi_1 := (\mathsf{x}_0, \mathsf{x}_1, \pi)$,
$\mathsf{x}_{\mathsf{PCC},0} = \mathsf{x}_{\mathsf{PCC},1} := \mathsf{x}_{\mathsf{PCC}}$, $\mathsf{w}_{\mathsf{PCC},0} = \mathsf{w}_{\mathsf{PCC},1} := \mathsf{w}_{\mathsf{PCC}}$.
Output $(\{\pi_b, \mathsf{x}_{\mathsf{PCC},b}, \mathsf{w}_{\mathsf{PCC},b}\}_{b\in\{0,1\}}, \mathsf{x}, \mathsf{w})$

$\Lambda'_1.\mathsf{LVerify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, b, \mathsf{x}_b, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},b}, \pi_b)$

Parse $\pi_b := (\mathsf{x}_0^*, \mathsf{x}_1^*, \pi)$
Accept iff $x_b^* = x_b$ and $\mathsf{NIRoK.Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, (\mathsf{x}_0^*, \mathsf{x}_1^*), (\mathsf{x}, \mathsf{x}_{\mathsf{PCC},b}), \pi) = 1$

$\Lambda'_\kappa.\mathsf{Gen}(1^\lambda, \mathsf{pp}')$

Output $\mathsf{pp} \leftarrow \mathsf{NIRoK.Gen}(1^\lambda, \mathsf{pp}')$

$\Lambda'_\kappa.\mathsf{LProve}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, ((\mathsf{x}_{\mathbf{b}}, \mathsf{w}_{\mathbf{b}}))_{\mathbf{b}\in\{0,1\}^\kappa})$

Compute the local proofs for the first and second half of the statements:
For $\mathbf{b} \in \{0,1\}^{\kappa-1}$ set :
$\left\{\pi'_{0\mathbf{b}}, \mathsf{x}'_{\mathsf{PCC},0\mathbf{b}}, \mathsf{w}'_{\mathsf{PCC},0\mathbf{b}}\right\}_{\mathbf{b}}, \mathsf{x}_0, \mathsf{w}_0, \leftarrow \Lambda'_{\kappa-1}.\mathsf{LProve}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, (\mathbf{x}_{0\mathbf{b}}, \mathbf{w}_{0\mathbf{b}})_{\mathbf{b}})$
$\left\{\pi'_{1\mathbf{b}}, \mathsf{x}'_{\mathsf{PCC},1\mathbf{b}}, \mathsf{w}'_{\mathsf{PCC},1\mathbf{b}}\right\}_{\mathbf{b}}, \mathsf{x}_1, \mathsf{w}_1, \leftarrow \Lambda'_{\kappa-1}.\mathsf{LProve}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, (\mathbf{x}_{1\mathbf{b}}, \mathbf{w}_{1\mathbf{b}})_{\mathbf{b}})$

Reduce the fresh statements:
$\pi, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC}}), (\mathsf{w}, \mathsf{w}_{\mathsf{PCC}}) \leftarrow \mathsf{NIRoK.Reduce}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, (\mathsf{x}_0, \mathsf{w}_0), (\mathsf{x}_1, \mathsf{w}_1))$

Prepare the output:
For $\mathbf{b} \in \{0,1\}^{\kappa-1}$ set:
$\pi_{0\mathbf{b}} := (\pi'_{0\mathbf{b}}, \mathsf{x}_0, \mathsf{x}_1, \pi)$, $\pi_{1\mathbf{b}} := (\pi'_{1\mathbf{b}}, \mathsf{x}_0, \mathsf{x}_1, \pi)$,
$\mathsf{x}_{\mathsf{PCC},0\mathbf{b}} := (\mathsf{x}_{\mathsf{PCC}}, \mathsf{x}'_{\mathsf{PCC},0\mathbf{b}})$, $\mathsf{x}_{\mathsf{PCC},1\mathbf{b}} := (\mathsf{x}_{\mathsf{PCC}}, \mathsf{x}'_{\mathsf{PCC},1\mathbf{b}})$,
$\mathsf{w}_{\mathsf{PCC},0\mathbf{b}} := (\mathsf{w}_{\mathsf{PCC}}, \mathsf{w}'_{\mathsf{PCC},0\mathbf{b}})$, $\mathsf{w}_{\mathsf{PCC},1\mathbf{b}} := (\mathsf{w}_{\mathsf{PCC}}, \mathsf{w}'_{\mathsf{PCC},1\mathbf{b}})$,

Output $(\{\pi_{\mathbf{b}}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \mathsf{w}_{\mathsf{PCC},\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa}, \mathsf{x}, \mathsf{w})$.

$\Lambda'_\kappa.\mathsf{LVerify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \pi_{\mathbf{b}})$

Parse $\mathbf{b} := (b, \mathbf{b}')$, $\pi_{\mathbf{b}} := (\pi_{\mathbf{b}'}, \mathsf{x}_0, \mathsf{x}_1, \pi))$ and $\mathsf{x}_{\mathsf{PCC},\mathbf{b}} := (\mathsf{x}_{\mathsf{PCC},b}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}'})$.
Accept iff:
(i) $\Lambda'_{\kappa-1}.\mathsf{LVerify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathbf{b}', \mathsf{x}_{\mathbf{b}}, \mathsf{x}_b, \mathsf{x}_{\mathsf{PCC},\mathbf{b}'}, \pi_{\mathbf{b}'}) = 1$
(ii) $\mathsf{NIRoK.Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, (\mathsf{x}_0, \mathsf{x}_1), (\mathsf{x}, \mathsf{x}_{\mathsf{PCC},b}), \pi) = 1$

Figure 11: An inductively defined compiler $\mathsf{L}_{\mathsf{PCC}}$ to derive an $\mathcal{R}^\kappa_{\mathsf{PCC}}$-promise K-local folding scheme $\Lambda'_\kappa = \mathsf{L}_{\mathsf{PCC}}(\mathsf{NIRoK}, \kappa)$ from a non-interactive reduction of knowledge $\mathsf{NIRoK} : \mathcal{R}^2 \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}$ in the random oracle model.

since NIRoK_1 sequentially verifies two reductions of knowledge NIRoK and $\mathsf{AUX}_{b_1,\mathcal{R},\mathcal{R}}$. Since π_{b_1} verifies, by construction of Λ'_1 it must be the case that:

$$\pi_{b_1} = (\mathsf{x}_0^*, \mathsf{x}_1^*, \pi), \quad \mathsf{x}_{b_1} = \mathsf{x}_{b_1}^*, \quad \mathsf{NIRoK.Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, (\mathsf{x}_0^*, \mathsf{x}_1^*), (\mathsf{x}, \mathsf{x}_{\mathsf{PCC},b_1}), \pi) = 1$$

We set $\mathsf{T}_1(\pi_{b_1}) = (((\mathsf{x}_0^*, \mathsf{x}_1^*), \perp), \pi)$. We next show that the corresponding verification algorithm accepts $\mathsf{T}_1(\pi_{b_1})$. The algorithm accepts iff

1. $\mathsf{NIRoK}_{\mathsf{AUX}}(\mathsf{x}_{b_1}, (\mathsf{x}_0^*, \mathsf{x}_1^*), \perp) = 1$
2. $\mathsf{NIRoK}.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, (\mathsf{x}_0^*, \mathsf{x}_1^*), (\mathsf{x}, \mathsf{x}_{\mathsf{PCC},b_1}), \pi) = 1$

The former is true iff $\mathsf{x}_{b_1} = \mathsf{x}_{b_1}^*$ by construction of AUX. Since $\mathsf{LVerify}$ accepts, the latter condition is true and also the condition $\mathsf{NIRoK}.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, (\mathsf{x}_0^*, \mathsf{x}_1^*), (\mathsf{x}, \mathsf{x}_{\mathsf{PCC},b_1}), \pi) = 1$. Therefore

$$\mathsf{NIRoK}_1.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_{b_1}, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC},b_1}), \mathsf{T}_1(\pi_{b_1})) = 1$$

Inductive Case. Assume the statement is true for $i < \ell$. We will show that it is also true for ℓ. Consider some input $\mathsf{pp}, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \pi_{\mathbf{b}}$ where $\mathbf{b} = (\mathbf{b}', b_\ell) = (b_1, \ldots, b_\ell)$ and assume that

$$\Lambda'_\ell.\mathsf{LVerify}^{\rho(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \pi_{\mathbf{b}}) = 1$$

Our goal is to efficiently construct a proof $\mathsf{T}_\ell(\pi_{\mathbf{b}})$ such that

$$\mathsf{NIRoK}_\ell.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_{\mathbf{b}}, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}), \mathsf{T}_\ell(\pi_{\mathbf{b}})) = 1$$

Let $\mathsf{NIRoK}^{(1)} = (\mathsf{AUX}_{b_\ell,\mathcal{R},\mathcal{R}} \times \mathsf{ID})$ and $\mathsf{NIRoK}^{(2)} = (\mathsf{NIRoK} \times \mathsf{ID})$. Since $\mathsf{NIRoK}_\ell = \mathsf{NIRoK}^{(2)} \circ \mathsf{NIRoK}^{(1)} \circ \mathsf{NIRoK}_{\ell-1}$ an accepting proof will be of the form:

$$\pi' = (\pi^{(2)}, (\mathsf{x}^{(2)}, \pi^{(1)}), (\mathsf{x}^{(1)}, \pi^{(0)}))$$

and the verification algorithm accepts iff

$$\begin{aligned} \mathsf{NIRoK}_{\ell-1}(\mathsf{pp}, \mathsf{x}_{\mathbf{b}}, \mathsf{x}^{(1)}, \pi^{(0)}) &= 1 \\ \mathsf{NIRoK}^{(1)}(\mathsf{pp}, \mathsf{x}^{(1)}, \mathsf{x}^{(2)}, \pi^{(1)}) &= 1 \\ \mathsf{NIRoK}^{(2)}(\mathsf{pp}, \mathsf{x}^{(2)}, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}), \pi^{(2)}) &= 1 \end{aligned} \tag{2}$$

Now, since $\Lambda'_\ell.\mathsf{LVerify}^{\rho(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \pi_{\mathbf{b}}) = 1$ the following conditions are satisfied by construction:

1. $\pi_{\mathbf{b}} = (\pi_{\mathbf{b}'}, \mathsf{x}_0, \mathsf{x}_1, \pi)$,
2. $\mathsf{x}_{\mathsf{PCC},\mathbf{b}} = (\mathsf{x}_{\mathsf{PCC},b_\ell}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}'})$,
3. $\Lambda'_{\ell-1}.\mathsf{LVerify}^{\rho(\cdot)}(\mathsf{pp}, \mathbf{b}', \mathsf{x}_{\mathbf{b}}, \mathsf{x}_{b_\ell}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}'}, \pi_{\mathbf{b}'}) = 1$,
4. $\mathsf{NIRoK}.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_0, \mathsf{x}_1, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},b_\ell}, \pi) = 1$

We construct T_ℓ by setting the following values:

- $\pi_0 := \mathsf{T}_{\ell-1}(\pi_{\mathbf{b}'})$
- $\mathsf{x}^{(1)} = (\mathsf{x}_{b_\ell}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}'})$,
- $\pi^{(1)} = (\perp, \perp)$
- $\mathsf{x}^{(2)} = ((\mathsf{x}_0, \mathsf{x}_1), \mathsf{x}_{\mathsf{PCC},\mathbf{b}'})$,
- $\pi^{(2)} = (\pi, \perp)$

We next show that with these values the equations in (2) are satisfied. For the first equation, we have

$$\mathsf{NIRoK}_{\ell-1}.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_{\mathbf{b}}, (\mathsf{x}_{b_\ell}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}'}), \mathsf{T}_{\ell-1}(\pi_{\mathbf{b}'})) = 1$$

Since $\Lambda'_{\ell-1}.\mathsf{LVerify}^{\rho(\cdot)}(\mathsf{pp}, \mathbf{b}', \mathsf{x}_{\mathbf{b}}, \mathsf{x}_{b_\ell}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}'}, \pi_{\mathbf{b}'}) = 1$ and by our induction hypothesis, the equation indeed holds. We next show that

$$\mathsf{NIRoK}^{(1)}.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, (\mathsf{x}_{b_\ell}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}'}), ((\mathsf{x}_0, \mathsf{x}_1), \mathsf{x}_{\mathsf{PCC},\mathbf{b}'}), (\bot, \bot)) = 1$$

This is a parallel execution of the (trivial) reductions of knowledge ID and AUX. By construction, this is true for both values of b_ℓ. Finally, we show that

$$\mathsf{NIRoK}^{(2)}.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, ((\mathsf{x}_0, \mathsf{x}_1), \mathsf{x}_{\mathsf{PCC},\mathbf{b}'}), ((\mathsf{x}, \mathsf{x}_{\mathsf{PCC},b_\ell}), \mathsf{x}_{\mathsf{PCC},\mathbf{b}'}), (\pi, \bot)) = 1$$

where we let $\mathsf{x}_{\mathsf{PCC},\mathbf{b}} = (\mathsf{x}_{\mathsf{PCC},b_\ell}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}'})$. This is a parallel execution of NIRoK and the (trivial) reduction of knowledge ID. This equation is satisfied since Λ'_κ verifies and therefore:

$$\begin{aligned}\mathsf{NIRoK}.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, (\mathsf{x}_0, \mathsf{x}_1), (\mathsf{x}, \mathsf{x}_{\mathsf{PCC},b_\ell}), \pi) = 1\\ \mathsf{NIRoK}_{\mathsf{ID}}(\mathsf{pp}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}'}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}'}, \bot) = 1\end{aligned}$$

We conclude that $\mathsf{NIRoK}_\ell.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_{\mathbf{b}}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \mathsf{T}_\ell(\pi_{\mathbf{b}})) = 1$.

□

We next present the proof of Thm 7.1.1.

Proof. Completeness, and the claimed efficiency measures are straightforward by construction. Note also that the efficiency requirement of Def. 6.1 is satisfied; in particular, the size of the promise statement $\mathsf{x}_{\mathsf{PCC},\mathbf{b}}$ depends on κ but only linearly and the proof consists of κ statements/proofs, therefore the proof is sublinear in $2^\kappa\,|\mathsf{x}|$. For proving knowledge soundness we rely on Lemma 7.1.1. Fix $\mathbf{b} \in \{0,1\}^\kappa$, $\mathsf{pp}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$ and let $\mathcal{A}^{\rho(\cdot)}_{\mathbf{b},1}$,$\mathcal{A}^{\rho(\cdot)}_{\mathbf{b},2}$ be any pair of expected polynomial time algorithms with access to random oracles $\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda)$ that act as knowledge soundness adversaries against the local folding scheme. On inputs $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}')$ and $\mathsf{x}_{\mathbf{b}}, \mathsf{st} \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathbf{b},1}(\mathsf{pp})$, adversary $\mathcal{A}^{\rho(\cdot)}_{\mathbf{b},2}(\mathsf{pp}, \mathsf{st})$ outputs

$$\mathsf{x}, \mathsf{w}, \mathsf{x}_{\mathsf{PCC}}, \mathsf{w}_{\mathsf{PCC}}, \pi_{\mathbf{b}} \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathbf{b},2}(\mathsf{pp}, \mathsf{st})$$

Next, consider the non-interactive reduction of knowledge defined by

$$\begin{aligned}\mathsf{NIRoK}_1 &= \mathsf{NIRoK} \circ \mathsf{AUX}_{b_1,\mathcal{R},\mathcal{R}} : \mathcal{R} \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}\\ \mathsf{NIRoK}_\ell &= (\mathsf{NIRoK} \times \mathsf{ID}) \circ (\mathsf{AUX}_{b_\ell,\mathcal{R},\mathcal{R}} \times \mathsf{ID}) \circ \mathsf{NIRoK}_{\ell-1} : \mathcal{R} \to \mathcal{R} \times \mathcal{R}^\ell_{\mathsf{PCC}}\end{aligned}$$

We construct a pair of expected polynomial time adversaries $\mathcal{A}^{\rho(\cdot)}_{\mathsf{RoK},1}, \mathcal{A}^{\rho(\cdot)}_{\mathsf{RoK},2}$ against the reduction of knowledge NIRoK_κ that we describe next. On input $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}')$, $\mathcal{A}^{\rho(\cdot)}_{\mathsf{RoK},1}(\mathsf{pp})$ simply invokes $\mathcal{A}^{\rho(\cdot)}_{\mathbf{b},1}(\mathsf{pp})$ and outputs whatever it outputs. On input $\mathsf{x}_{\mathbf{b}}, \mathsf{st}$ adversary $\mathcal{A}^{\rho(\cdot)}_{\mathsf{RoK},2}(\mathsf{pp}, \mathsf{st})$ does the following:

1. it computes $\mathsf{x}, \mathsf{w}, \mathsf{x}_{\mathsf{PCC}}, \mathsf{w}_{\mathsf{PCC}}, \pi_{\mathbf{b}} \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathbf{b},2}(\mathsf{pp})$
2. it computes $\pi_{\mathsf{RoK}} = \mathsf{T}_\kappa(\pi_{\mathbf{b}})$
3. it outputs $(\mathsf{x}, \mathsf{x}_{\mathsf{PCC}}), (\mathsf{w}, \mathsf{w}_{\mathsf{PCC}}), \pi_{\mathsf{RoK}}$

By Lemma 7.1.1 it holds that

$$\Lambda'_\kappa.\mathsf{LVerify}^{\rho(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_\mathbf{b}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \pi_\mathbf{b}) = 1 \Rightarrow \mathsf{NIRoK}_\kappa.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_\mathbf{b}, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}), \mathsf{T}_\kappa(\pi_\mathbf{b})) = 1$$

and therefore

$$\Pr\left[\begin{array}{c}(\mathsf{pp}, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC}}), (\mathsf{w}, \mathsf{w}_{\mathsf{PCC}})) \in \mathcal{R} \times \mathcal{R}^\kappa_{\mathsf{PCC}} \wedge \\ \mathsf{NIRoK}_\kappa.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_\mathbf{b}, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC}}), \pi_{\mathsf{RoK}}) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_\mathbf{b}, \mathsf{st} \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathsf{RoK},1}(\mathsf{pp}) \\ \pi_{\mathsf{RoK}}, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC}}), (\mathsf{w}, \mathsf{w}_{\mathsf{PCC}}) \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathsf{RoK},2}(\mathsf{pp}, \mathsf{st})\end{array}\right] \geq$$

$$\Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R} \wedge \\ (\mathsf{pp}, \mathsf{x}_{\mathsf{PCC}}, \mathsf{w}_{\mathsf{PCC}}) \in \mathcal{R}^\kappa_{\mathsf{PCC}} \wedge \\ \mathsf{NIRoK}_\kappa.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_\mathbf{b}, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC}}), \mathsf{T}_\kappa(\pi_\mathbf{b})) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_\mathbf{b}, \mathsf{st} \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathbf{b},1}(\mathsf{pp}) \\ \pi_\mathbf{b}, \mathsf{x}, \mathsf{w}, \mathsf{x}_{\mathsf{PCC}}, \mathsf{w}_{\mathsf{PCC}} \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathbf{b},2}(\mathsf{pp}, \mathsf{st})\end{array}\right] \geq$$

$$\Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R} \wedge \\ (\mathsf{pp}, \mathsf{x}_{\mathsf{PCC}}, \mathsf{w}_{\mathsf{PCC}}) \in \mathcal{R}^\kappa_{\mathsf{PCC}} \wedge \\ \Lambda'_\kappa.\mathsf{LVerify}^{\rho(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_\mathbf{b}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \pi_\mathbf{b}) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_\mathbf{b}, \mathsf{st} \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathbf{b},1}(\mathsf{pp}) \\ \pi_\mathbf{b}, \mathsf{x}, \mathsf{w}, \mathsf{x}_{\mathsf{PCC}}, \mathsf{w}_{\mathsf{PCC}} \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathbf{b},2}(\mathsf{pp}, \mathsf{st})\end{array}\right]$$

By knowledge soundness of the NIRoK_κ, there exists an expected polynomial time extractor $\mathcal{E}$ s.t.

$$\Pr\left[\begin{array}{c}(\mathsf{pp}, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC}}), (\mathsf{w}, \mathsf{w}_{\mathsf{PCC}})) \in \mathcal{R} \times \mathcal{R}^\kappa_{\mathsf{PCC}} \wedge \\ \mathsf{NIRoK}_\kappa.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_\mathbf{b}, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC}}), \pi_{\mathsf{RoK}}) = 1\end{array} \middle| \begin{array}{r}\rho_1 \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_\mathbf{b}, \mathsf{st} \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathsf{RoK},1}(\mathsf{pp}) \\ \mathsf{w}_\mathbf{b} \leftarrow \mathcal{E}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right] + \mathsf{negl}(\lambda) \geq$$

$$\Pr\left[\begin{array}{c}(\mathsf{pp}, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC}}), (\mathsf{w}, \mathsf{w}_{\mathsf{PCC}})) \in \mathcal{R} \times \mathcal{R}^\kappa_{\mathsf{PCC}} \wedge \\ \mathsf{NIRoK}_\kappa.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_\mathbf{b}, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC}}), \pi_{\mathsf{RoK}}) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_\mathbf{b}, \mathsf{st} \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathsf{RoK},1}(\mathsf{pp}) \\ \pi_{\mathsf{RoK}}, (\mathsf{x}, \mathsf{x}_{\mathsf{PCC}}), (\mathsf{w}, \mathsf{w}_{\mathsf{PCC}}) \leftarrow \\ \mathcal{A}^{\rho(\cdot)}_{\mathsf{RoK},2}(\mathsf{pp}, \mathsf{st})\end{array}\right] \geq$$

$$\Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R} \wedge \\ (\mathsf{pp}, \mathsf{x}_{\mathsf{PCC}}, \mathsf{w}_{\mathsf{PCC}}) \in \mathcal{R}^\kappa_{\mathsf{PCC}} \wedge \\ \Lambda'_\kappa.\mathsf{LVerify}^{\rho(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_\mathbf{b}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \pi_\mathbf{b}) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{Gen}^{\rho(\cdot)}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_\mathbf{b}, \mathsf{st} \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathbf{b},1}(\mathsf{pp}) \\ \pi_\mathbf{b}, \mathsf{x}, \mathsf{w}, \mathsf{x}_{\mathsf{PCC}}, \mathsf{w}_{\mathsf{PCC}} \leftarrow \mathcal{A}^{\rho(\cdot)}_{\mathbf{b},2}(\mathsf{pp}, \mathsf{st})\end{array}\right]$$

and therefore $\mathcal{E}$ is a valid knowledge extractor against Λ'_κ.

□

Remark 7.1.1. Consider a reduction of knowledge $\mathsf{NIRoK} : \mathcal{R}^2 \to \mathcal{R}$. We can always view this as a reduction of knowledge $\mathsf{NIRoK} : \mathcal{R}^2 \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}$ with an empty $\mathcal{R}_{\mathsf{PCC}}$ statement. In this case, the compiler $\mathsf{L}_{\mathsf{PCC}}(\mathsf{NIRoK}, \kappa)$ results in a plain local folding scheme for $\mathcal{R}$.

7.2 From $\mathcal{R}_{\mathsf{PCC}}$-Promise to Plain Local Folding Schemes

We present a compiler from an $\mathcal{R}_{\mathsf{PCC}}$-Promise Local Folding Scheme to a plain (i.e. without promise) Local Folding Scheme. Combining this with the compiler of the previous section, one can derive a Local Folding Scheme for $\mathcal{R}$ from a non-interactive reduction of knowledge $\mathsf{NIRoK} : \mathcal{R}^2 \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}$. The compiler essentially does the following:

1. it uses the $\mathcal{R}_{\mathsf{PCC}}$-Promise Local Folding Scheme $\Lambda'_\kappa = \mathsf{L}_{\mathsf{PCC}}(\mathsf{NIRoK}, \kappa)$ to reduce all statements to a single claim of $\mathcal{R}$ *and* an $\mathcal{R}^\kappa_{\mathsf{PCC}}$ statement for each individual statement,
2. it removes the degree tests by reducing each $\mathcal{R}_{\mathsf{PCC}}$ to a $\mathcal{R}_{\mathsf{PCC\text{-}D}}$,
3. it reduces each $\mathcal{R}^\kappa_{\mathsf{PCC\text{-}D}}$ statement to an $\mathcal{R}_{\mathsf{PCO}}$ statement using $\widetilde{\mathsf{NIRoK}}_{\mathsf{PC}} = \mathsf{FS}_{\mathsf{MT}}(\Pi_{\mathsf{PC}})$,
4. after fixing x from the previous step, it uses a local folding scheme $\Lambda_{\mathsf{PCO},\kappa} = \mathsf{L}_{\mathsf{PCC}}(\mathsf{NIRoK}_{\mathsf{PCO}(x)}, \kappa)$ derived from $\mathsf{NIRoK}_{\mathsf{PCO}(x)} : \mathcal{R}^2_{\mathsf{PCO}(x)} \to \mathcal{R}_{\mathsf{PCO}(x)}$ to reduce all $\mathcal{R}_{\mathsf{PCO}(x)}$ statements to a single $\mathcal{R}_{\mathsf{PCO}(x)}$ statement,
5. it proves the latter statement, which simply consists of opening a polynomial commitment at x.

During the third step, we employ the algorithm AmortizedReduce as described in Thm. 5.2 to amortize the cost of computing all proofs. For the sake of notational simplicity, we present the construction with all algorithms having access to all involved random oracles, which we denote $\boldsymbol{\rho}$, instead of using a different set of random oracles for each of the components.

Theorem 7.2.1. *Let $K = 2^\kappa \in \mathbb{N}$ and let*

1. PC *be a homomorphic commitment scheme over some field $\mathbb{F}$ with $|\mathbb{F}| = \Theta(2^\lambda)$, of maximum supported degree $D = \mathcal{O}(\mathsf{poly}(\lambda))$, and with commitments of size $\mathcal{O}(\lambda)$,*
2. $\mathsf{NIRoK} : \mathcal{R}^2 \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}$ *be a reduction of knowledge that in each invocation outputs at most constant m polynomial commitments for some m and $\mathsf{NIRoK.Gen}$ invokes $\mathsf{PC.Setup}(1^\lambda, D)$ for some $D \in \mathbb{N}$.*

Then construction $\Lambda_\kappa = \mathsf{L}(\mathsf{NIRoK}, \kappa)$ is a (plain) K-local folding scheme for the relation $\mathcal{R}$ in the random oracle model. Furthermore, let $t_{\mathcal{P}}(\Pi^*), t_{\mathcal{V}}(\Pi^*), \pi(\Pi^*)$ the proving time, verifying time and proof size of $\mathsf{FS}(\Pi^*)$ and let $|\mathcal{R}^*|$ the size of a statement of $\mathcal{R}^*$[6] and let $t_{\mathcal{P}}(\mathsf{PC}), t_{\mathcal{V}}(\mathsf{PC}), \pi(\mathsf{PC})$ be the corresponding measures for the polynomial commitment PC. Then Λ_κ has the following efficiency characteristics:

- *prover time is $\mathcal{O}(t_{\mathcal{P}}(\mathsf{PC}) + K \cdot (t_{\mathcal{P}}(\mathsf{NIRoK}) + t_{\mathcal{P}}(\Pi_{\mathsf{D}}) + t_{\mathcal{P}}(\Pi_{\mathsf{PC}}) + t_{\mathcal{P}}(\Pi_{\mathsf{PCO}(x)})))$*
- *verifier time is $\mathcal{O}(t_{\mathcal{V}}(\mathsf{PC}) + \kappa \cdot (t_{\mathcal{V}}(\mathsf{NIRoK}) + t_{\mathcal{V}}(\Pi_{\mathsf{D}}) + t_{\mathcal{V}}(\Pi_{\mathsf{PC}}) + t_{\mathcal{V}}(\Pi_{\mathsf{PCO}})))$*
- *proof size is $\mathcal{O}(\pi(\mathsf{PC}) + \kappa \cdot (\lambda + |\mathcal{R}_\Pi| + |\mathcal{R}_{\mathsf{PCC}}| + |\mathcal{R}_{\mathsf{PCO}}| + \pi(\Pi)))$*

Proof. Completeness is straightforward. For proving knowledge soundness, we start by recalling that Lemma 7.1.1 tells us that there exist reductions of knowledge $\mathsf{NIRoK}^{(1)}, \mathsf{NIRoK}^{(2)}$ s.t.

$$\Lambda'_\kappa.\mathsf{LVerify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_{\mathbf{b}}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \pi_{\mathbf{b}}) = 1 \Rightarrow \mathsf{NIRoK}^{(1)}.\mathsf{Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{x}_{\mathbf{b}}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \mathsf{T}^{(1)}_\kappa(\pi_{\mathbf{b}})) = 1$$

$$\Lambda_{\mathsf{PCO}}.\mathsf{LVerify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_{\mathsf{PCO}(x),\mathbf{b}}, \mathsf{x}_{\mathsf{PCO}}, \epsilon, \pi_{\mathsf{PCO},\mathbf{b}}) = 1 \Rightarrow \mathsf{NIRoK}^{(2)}.\mathsf{Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{x}_{\mathsf{PCO}(x),\mathbf{b}}, \mathsf{x}_{\mathsf{PCO}}, \epsilon, \mathsf{T}^{(2)}_\kappa(\pi_{\mathsf{PCO},\mathbf{b}})) = 1$$

We start by viewing tests (1) – (3) of the construction as a reduction of knowledge $\mathsf{NIRoK}^* : \mathcal{R} \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCO}}$. Indeed, we can define

$$\mathsf{NIRoK}^* = (\mathsf{ID} \times (\mathsf{NIRoK}_{\mathsf{PC}} \circ \widetilde{\mathsf{NIRoK}}_{\mathsf{D}})) \circ \mathsf{NIRoK}^{(1)} : \mathcal{R} \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCO}}$$

The rest of the verification tests essentially constitute a proof of knowledge for the second statement, namely the one about $\mathcal{R}_{\mathsf{PCO}}$. Indeed, the verifier defines the relation $\mathcal{R}_{\mathbf{b},\mathsf{PCO}(x)}$ for the x included in

[6] We implicitly assume that the involved statements have a fixed, a priori known size.

$\Lambda_\kappa.\mathsf{Gen}(1^\lambda, \mathsf{pp}')$

Output $\mathsf{ck}, \mathsf{vk} \leftarrow \mathsf{PC.Setup}(1^\lambda, D)$

$\Lambda_\kappa.\mathsf{LProve}^{\rho(\cdot)}(\mathsf{pp}, \{\mathsf{x}_\mathbf{b}, \mathsf{w}_\mathbf{b}\}_{\mathbf{b}\in\{0,1\}^\kappa})$

1. Compute the local partial proofs via $\Lambda'_\kappa = \mathsf{L}_{\mathsf{PCC}}(\mathsf{NIRoK}, \kappa)$:
$(\{\pi'_\mathbf{b}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \mathsf{w}_{\mathsf{PCC},\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa}, \mathsf{x}, \mathsf{w}) \leftarrow \Lambda'_\kappa.\mathsf{LProve}^{\rho(\cdot)}(\mathsf{pp}, \{\mathsf{x}_\mathbf{b}, \mathsf{w}_\mathbf{b}\}_{\mathbf{b}\in\{0,1\}^\kappa})$.

2. reduce all $\mathcal{R}^\kappa_{\mathsf{PCC}}$ statements to $\mathcal{R}_{\text{PCC-D}}$ statements: for $\mathbf{b} \in \{0,1\}^\kappa$
$(\pi_{\mathsf{D},\mathbf{b}}, \mathsf{x}_{\text{PCC-D},\mathbf{b}}, \mathsf{w}_{\text{PCC-D},\mathbf{b}}) \leftarrow \mathsf{NIRoK}_\mathsf{D}.\mathsf{Reduce}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \mathsf{w}_{\mathsf{PCC},\mathbf{b}}, \epsilon)$

3. reduce all $\mathcal{R}_{\text{PCC-D}}$ statements to $\mathcal{R}_{\mathsf{PCO}}$ statements:
$(\{\pi_{\mathsf{PCC},\mathbf{b}}, \mathsf{x}_{\mathsf{PCO},\mathbf{b}}, \mathsf{w}_{\mathsf{PCO},\mathbf{b}}\}_{\mathbf{b}\in\{0,1\}^\kappa})$
$\leftarrow \widetilde{\mathsf{NIRoK}}_{\mathsf{PC}}.\mathsf{AmortizedReduce}^{\rho(\cdot)}(\mathsf{pp}, \{\mathsf{x}_{\text{PCC-D},\mathbf{b}}, \mathsf{w}_{\text{PCC-D},\mathbf{b}}, \epsilon\}_{\mathbf{b}\in\{0,1\}^\kappa})$

4. Compute the local proofs for $\mathcal{R}_{\mathsf{PCO}(x)}$ via $\Lambda_{\mathsf{PCO},\kappa} = \mathsf{L}_{\mathsf{PCC}}(\mathsf{FS}(\Pi_{\mathsf{PCO}(x)}), \kappa)$:
$(\{\pi_{\mathsf{PCO},\mathbf{b}},\}_{\mathbf{b}\in\{0,1\}^\kappa}, \mathsf{x}_{\mathsf{PCO}} := (\mathsf{c}, y), \mathsf{w}_{\mathsf{PCO}} := p(X))$
$\leftarrow \Lambda_{\mathsf{PCO},\kappa}.\mathsf{LProve}^{\rho(\cdot)}(\mathsf{pp}, \left\{\mathsf{x}_{\mathsf{PCO}(x),\mathbf{b}}, \mathsf{w}_{\mathsf{PCO},\mathbf{b}}\right\}_{\mathbf{b}\in\{0,1\}^\kappa})$
where $\mathsf{x}_{\mathsf{PCO}(x)}$ is derived from $\mathsf{x}_{\mathsf{PCO}}$ by removing x

5. Compute an opening proof for the statement $\mathsf{x}_{\mathsf{PCO}}$:
$\pi_{\mathsf{PC}} \leftarrow \mathsf{PC.Open}(\mathsf{ck}, \mathsf{c}, x, y, p(X), D)$

Compute the proofs:
For $\mathbf{b} \in \{0,1\}^\kappa$:
$\pi_\mathbf{b} := ((\pi'_\mathbf{b}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}), (\pi_{\mathsf{D},\mathbf{b}}, \mathsf{x}_{\text{PCC-D},\mathbf{b}}), (\pi_{\mathsf{PCC},\mathbf{b}}, \mathsf{x}_{\mathsf{PCO},\mathbf{b}}), (\pi_{\mathsf{PCO},\mathbf{b}}, \mathsf{x}_{\mathsf{PCO}}), \pi_{\mathsf{PC}})$

Output $\{\pi_\mathbf{b}\}_{\mathbf{b}\in\{0,1\}^\kappa}, \mathsf{x}, \mathsf{w}$

$\Lambda_\kappa.\mathsf{LVerify}^{\rho(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_\mathbf{b}, \mathsf{x}, \pi_\mathbf{b})$

Parse the proof:
$\pi_\mathbf{b} := ((\pi'_\mathbf{b}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}), (\pi_{\mathsf{D},\mathbf{b}}, \mathsf{x}_{\text{PCC-D},\mathbf{b}}), (\pi_{\mathsf{PCC},\mathbf{b}}, \mathsf{x}_{\mathsf{PCO},\mathbf{b}}), (\pi_{\mathsf{PCO},\mathbf{b}}, \mathsf{x}_{\mathsf{PCO}}), \pi_{\mathsf{PC}})$
Parse $\mathsf{x}_{\mathsf{PCO},\mathbf{b}} = (\mathsf{c}, x, y)$ and set $\mathsf{x}_{\mathsf{PCO}(x),\mathbf{b}} = (\mathsf{c}, y)$ Accept iff:
(1): $\Lambda'_\kappa.\mathsf{LVerify}^{\rho(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_\mathbf{b}, \mathsf{x}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \pi'_\mathbf{b}) = 1$
(2): $\mathsf{NIRoK}_\mathsf{D}.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_{\mathsf{PCC},\mathbf{b}}, \mathsf{x}_{\text{PCC-D},\mathbf{b}}, \pi_{\mathsf{D},\mathbf{b}}) = 1$
(3): $\widetilde{\mathsf{NIRoK}}_{\mathsf{PC}}.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_{\text{PCC-D},\mathbf{b}}, \mathsf{x}_{\mathsf{PCO},\mathbf{b}}, \pi_{\mathsf{PCC},\mathbf{b}}) = 1$
(4): $\Lambda_{\mathsf{PCO},\kappa}.\mathsf{LVerify}^{\rho(\cdot)}(\mathsf{pp}, \mathbf{b}, \mathsf{x}_{\mathsf{PCO}(x),\mathbf{b}}, \mathsf{x}_{\mathsf{PCO}}, \pi'_{\mathsf{PCO},\mathbf{b}}) = 1$
(5): $\mathsf{PC.Verify}(\mathsf{vk}, \mathsf{c}, x, y, \pi_{\mathsf{PC}}, D) = 1$

Figure 12: Compiler L that derives a (plain) K-local folding scheme $\Lambda_\kappa = \mathsf{L}(\mathsf{NIRoK}, \kappa)$ from a non-interactive Reduction of Knowledge $\mathsf{NIRoK} : \mathcal{R}^2 \to \mathcal{R} \times \mathcal{R}_{\mathsf{PCC}}$ in the random oracle model.

$\mathcal{R}_{\mathsf{PCO},\mathbf{b}}$ and verifies the second local proof, which is equivalent to verifying the reduction of knowledge $\mathsf{NIRoK}^{(2)} : \mathcal{R}_{\mathsf{PCO}(x)} \to \mathcal{R}_{\mathsf{PCO}(x)}$. Further, it verifies a proof for the reduced statement, namely, the

polynomial commitment opening proof in test (5).

By extractability of PC, a verifying proof implies we can extract a witness $\mathsf{w}_{\mathsf{PCO}}$ s.t. $(\mathsf{pp}, \mathsf{x}_{\mathsf{PCO}(x)}, \mathsf{w}_{\mathsf{PCO}}) \in \mathcal{R}_{\mathsf{PCO}(x)}$ and since the value x is used to define $\mathsf{x}_{\mathsf{PCO}(x)}$ we also get $(\mathsf{pp}, \mathsf{x}_{\mathsf{PCO}}, \mathsf{w}_{\mathsf{PCO}}) \in \mathcal{R}_{\mathsf{PCO}}$ except w.n.p. Therefore, the promise $\mathsf{x}_{\mathsf{PCO},\mathbf{b}}$ holds and we can view the verification as a whole as a reduction of knowledge $\mathcal{R} \to \mathcal{R}$. Arguing as in Thm. 7.1.1, given an accepting proof $\pi_{\mathbf{b}}$ we can extract a witness $\mathsf{w}_{\mathbf{b}}$ s.t. $(\mathsf{pp}, \mathsf{x}_{\mathbf{b}}, \mathsf{w}_{\mathbf{b}}) \in \mathcal{R}$.

For efficiency, we have:

- the Λ_κ.LProve algorithm (1) calls Λ'_κ.Prove which by Thm 7.1.1 has $\mathcal{O}(K \cdot t_{\mathcal{P}}(\Pi))$ time complexity, (2) calls K times $\mathsf{NIRoK}_{\mathsf{D}}$.Reduce which by definition has $\mathcal{O}(K \cdot t_{\mathcal{P}}(\Pi_{\mathsf{D}}))$ time complexity, (3) calls $\widetilde{\mathsf{NIRoK}}_{\mathsf{PC}}$.AmortizedReduce which by Thm. 5.2 has time complexity $\mathcal{O}(K \cdot t_{\mathcal{P}}(\Pi_{\mathsf{PC}}))$, (4) calls Λ_{PCO}.Prove which by Thm 7.1.1 has $\mathcal{O}(K \cdot t_{\mathcal{P}}(\Pi_{\mathsf{PCO}}))$ time complexity and (5) creates a PC opening proof which has time complexity $t_{\mathcal{P}}(\mathsf{PC})$.
- the Λ_κ.LVerify algorithm (1) calls Λ'_κ.Verify which by Thm 7.1.1 has $\mathcal{O}(\kappa \cdot t_{\mathcal{V}}(\Pi))$ time complexity, (2) calls $\mathsf{NIRoK}_{\mathsf{D}}$.Verify which by definition has $\mathcal{O}(t_{\mathcal{V}}(\Pi_{\mathsf{D}}))$ time complexity, (3) calls $\widetilde{\mathsf{NIRoK}}_{\mathsf{PC}}$.Verify which by Thm. 5.2 has time complexity $\mathcal{O}(\kappa \cdot t_{\mathcal{V}}(\Pi_{\mathsf{PC}}))$, (4) calls Λ_{PCO}.Verify which by Thm 7.1.1 has $\mathcal{O}(\kappa \cdot t_{\mathcal{V}}(\Pi_{\mathsf{PCO}}))$ time complexity and (5) verifies a PC opening proof which has time complexity $t_{\mathcal{V}}(\mathsf{PC})$.
- The proof consists of (1) the Λ'_κ proof which is of size $\kappa \cdot \pi(\Pi)$, (2) the statement $\mathsf{x}_{\mathsf{PCC},\mathbf{b}}$ which for constant m is of size $\mathcal{O}(\kappa)$ (3) the proof of $\widetilde{\mathsf{NIRoK}}_{\mathsf{D}}$ which is of size $\mathcal{O}(\kappa\lambda)$ for constant m since the prover only sends the commitments to the scaled polynomials, (4) the statement $\mathsf{x}_{\mathsf{PCC\text{-}D},\mathbf{b}}$ which for constant m is of size $\mathcal{O}(\kappa)$ (5) the proof of $\widetilde{\mathsf{NIRoK}}_{\mathsf{PC}}$ which by Thm. 5.2 is of size $\mathcal{O}(\kappa\lambda)$ for constant m (6) the PCO statements $\mathsf{x}_{\mathsf{PCO},\mathbf{b}}$, $\mathsf{x}_{\mathsf{PCO}}$ which are of size $|\mathcal{R}_{\mathsf{PCO}}|$ (7) the Λ_{PCO} proof which is of size $\mathcal{O}(\kappa\lambda)$ for constant m and (8) the proof of PC with size $\pi(\mathsf{PC})$.

□

Remark 7.2.1. Note that verifiers share common $\mathsf{x}_{\mathsf{PCC},\mathbf{b}}$ statements. In particular as we create the first local proof, all $\mathcal{R}_{\mathsf{PCC}}$ statements "converge" to the root statement and the closer we are to it, the more verifiers need to verify the statement there. With careful bookkeeping, this can be taken into account and further improve the (concrete) costs of the prover and verifier by accumulating each statement only once.

8 A Local Folding Scheme for Bivariate Polynomial Evaluation

We define the relation $\mathcal{R}_{\mathbf{M}}$ which captures that for a fixed matrix $\mathbf{M} \in \mathbb{F}^{n \times n}$ and $\mathbf{u}, \mathbf{v} \in \mathbb{F}^n$ it holds $y = \mathbf{u}^\top \mathbf{M} \mathbf{v}$; $\mathbf{u}, \mathbf{v}$ are committed as polynomials $u(X) = \mathbf{u}^\top \boldsymbol{\lambda}(X)$, $v(X) = \mathbf{v}^\top \boldsymbol{\lambda}(X)$ respectively. Crucially, these polynomials are committed in "sparse" form as pairs of polynomials $N_u(X), T_u(X)$

and $N_v(X), T_v(X)$ such that $u(X) = N_u(X)/nT_u(X)$ and $v(X) = N_v(X)/nT_v(X)$. We define:

$$\mathcal{R}_{\mathbf{M}} = \left\{ \begin{pmatrix} (\mathsf{ck}, \mathsf{vk}) \\ (\mathsf{c}_N, \mathsf{c}_T, \mathsf{d}_N, \mathsf{d}_T, \mathbf{d}, y), \\ \begin{pmatrix} N_u(X), T_u(X), \mathbf{u}, \\ N_v(X), T_v(X), \mathbf{v} \end{pmatrix} \end{pmatrix} \middle| \begin{array}{c} \mathbf{d} = (d_{N,u}, d_{T,u}, d_{N,v}, d_{T,v}) \wedge \\ \mathsf{c}_N = \mathsf{PC.Commit}(\mathsf{ck}, N_u(X), d_{N,u}) \wedge \\ \mathsf{d}_N = \mathsf{PC.Commit}(\mathsf{ck}, N_v(X), d_{N,v}) \wedge \\ \mathsf{c}_T = \mathsf{PC.Commit}(\mathsf{ck}, T_u(X), d_{T,u}) \wedge \\ \mathsf{d}_T = \mathsf{PC.Commit}(\mathsf{ck}, T_v(X), d_{T,v}) \wedge \\ N_u(X)/nT_u(X) = \mathbf{u}^\top \boldsymbol{\lambda}(X) \wedge \\ N_v(X)/nT_v(X) = \mathbf{v}^\top \boldsymbol{\lambda}(X) \wedge \\ y = \mathbf{u}^\top \mathbf{M} \mathbf{v} \end{array} \right\}$$

Note that the matrix $\mathbf{M}$ is "hardcoded" in the relation itself. Note also that for $\mathbf{u} = \boldsymbol{\lambda}(\alpha)$, $\mathbf{v} = \boldsymbol{\lambda}(\beta)$ we have $y = \boldsymbol{\lambda}(\alpha)^\top \mathbf{M} \boldsymbol{\lambda}(\beta)$ which is equivalent to the evaluation of the bivariate polynomial $M(\alpha, \beta)$ represented as the matrix $\mathbf{M}$.

8.1 A Reduction of Knowledge for (Partially) Folding $\mathcal{R}_{\mathbf{M}}$ Statements

We present a public-coin reduction of knowledge $\Pi_{\mathbf{M}}$ for $\mathcal{R}^2_{\mathbf{M}} \to \mathcal{R}_{\mathbf{M}} \times \mathcal{R}_{\mathsf{PCC}}$. The protocol is described in Fig. 13.

Theorem 8.1.1. *Let* PC *be a homomorphic polynomial commitment scheme and* $D \in \mathbb{N}$ *be sufficiently large. Then, the construction* $\Pi_{\mathbf{M}}$ *of Fig. 13 is a 2-round public coin reduction of knowledge* $\mathcal{R}^2_{\mathbf{M}} \to \mathcal{R}_{\mathbf{M}} \times \mathcal{R}_{\mathsf{PCC}}$*. Furthermore, let*

- $s_{\mathbf{M}}$ *be the sparsity of the matrix* $\mathbf{M}$*,*
- s_p *be the maximum sparsity of* $(N_{u,b}(X), T_{u,b}(X), N_{v,b}(X), T_{v,b}(X))$*,*
- $t_{\mathsf{Commit}}(s)$ *be the time to commit to an* s*-sparse polynomial using* PC *and*
- $|\mathsf{PC}|$ *be the size of a commitment produced by* PC*.*

Then construction $\Pi_{\mathbf{M}}$ *has the following efficiency characteristics:*

- *time complexity of the prover is* $\mathcal{O}((s_{\mathbf{M}} + s_p \log s_p)\ \mathbb{F} + t_{\mathsf{Commit}}(2 \cdot s_p))$*, time complexity of the verifier is* $\mathcal{O}(1\ \mathbb{F})$*, proof size is* $\mathcal{O}(1\ |\mathsf{PC}| + 1 \cdot \mathbb{F})$*, and size of* $\mathsf{x}_{\mathsf{PCC}}$ *is* $\mathcal{O}(1\ |\mathsf{PC}|)$*.*

Proof.
Public reducibility is straightforward. We next focus on the rest of the properties.

Completeness. Assume that $(\mathsf{pp}, \mathsf{x}_b, \mathsf{w}_b) \in \mathcal{R}_{\mathbf{M}}$ for $b \in \{0, 1\}$. If the openings of the polynomial commitments correspond to $N_{u,b}(X)/nT_{u,b}(X)$, $N_{v,b}(X)/nT_{v,b}(X)$ then the same is true for the new polynomials $N^*_u(X)/nT^*_u(X)$. In this case:

$$N_{u,b}(X)/nT_{u,b}(X) = \mathbf{u}_b^\top \boldsymbol{\lambda}(X), \qquad N_{v,b}(X)/nT_{v,b}(X) = \mathbf{v}_b^\top \boldsymbol{\lambda}(X), \qquad y_b = \mathbf{u}_b^\top \mathbf{M} \mathbf{v}_b$$

Since the $(\mathsf{pp}, \mathsf{x}_{\mathsf{PCC}}, \mathsf{w}_{\mathsf{PCC}}) \in \mathcal{R}_{\mathsf{PCC}}$ we know that the new commitments can be opened to the four polynomials that the prover computes.

$$\frac{N^*_u(X)}{nT^*_u(X)} = \frac{N_{u,0}(X)}{nT_{u,0}(X)} + \gamma \frac{N_{u,1}(X)}{nT_{u,1}(X)} = \mathbf{u}_0^\top \boldsymbol{\lambda}(X) + \gamma \mathbf{u}_1^\top \boldsymbol{\lambda}(X) = \mathbf{u}^{*\top} \boldsymbol{\lambda}(X)$$
$$\frac{N^*_v(X)}{nT^*_v(X)} = \frac{N_{v,0}(X)}{nT_{v,0}(X)} + \gamma \frac{N_{v,1}(X)}{nT_{v,1}(X)} = \mathbf{v}_0^\top \boldsymbol{\lambda}(X) + \gamma \mathbf{v}_1^\top \boldsymbol{\lambda}(X) = \mathbf{v}^{*\top} \boldsymbol{\lambda}(X)$$

Finally, doing the calculations for y^* we indeed get

$$y^* = (\mathbf{u}_0 + \gamma \mathbf{u}_1)^\top \mathbf{M} (\mathbf{v}_0 + \gamma \mathbf{v}_1) = \mathbf{u}_0^\top \mathbf{M} \mathbf{v}_0 + \gamma (\mathbf{u}_0^\top \mathbf{M} \mathbf{v}_1 + \mathbf{u}_1^\top \mathbf{M} \mathbf{v}_0) + \gamma^2 \mathbf{u}_1^\top \mathbf{M} \mathbf{v}_1 = y_0 + \gamma z + \gamma^2 y_1$$

$\mathsf{Gen}(1^\lambda, D)$: output $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D)$

$\mathcal{P}$ | $\mathcal{V}$

Input: $(\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}_{\mathbf{M}}^2$, $\mathsf{aux} = \bot$ | Input: $(\mathsf{pp}, \mathsf{x})$

Both parse $\mathsf{x}_b := (\mathsf{c}_{N,b}, \mathsf{c}_{T,b}, \mathsf{d}_{N,b}, \mathsf{d}_{T,b}, \mathbf{d}_b, y_b)$

Parse $\mathsf{w}_b := (N_{u,b}(X), T_{u,b}(X), \mathbf{u}_b, N_{v,b}(X), T_{v,b}(X), \mathbf{v}_b)$ and compute $z := \mathbf{u}_0^\top \mathbf{M} \mathbf{v}_1 + \mathbf{u}_1^\top \mathbf{M} \mathbf{v}_0$

$\longrightarrow z \longrightarrow$

$\longleftarrow \gamma \leftarrow \mathbb{F} \longleftarrow$

Compute $\mathbf{u}^* = \mathbf{u}_0 + \gamma \mathbf{u}_1$, $\mathbf{v}^* = \mathbf{v}_0 + \gamma \mathbf{v}_1$ and

$$N_u^*(X) = N_{u,0}(X)T_{u,1}(X) + \gamma N_{u,1}(X)T_{u,0}(X), \qquad T_u^*(X) = T_{u,0}(X)T_{u,1}(X),$$

$$N_v^*(X) = N_{v,0}(X)T_{v,1}(X) + \gamma N_{v,1}(X)T_{v,0}(X), \qquad T_v^*(X) = T_{v,0}(X)T_{v,1}(X)$$

and the corresponding commitments $\mathsf{c}_N^*, \mathsf{c}_T^*, \mathsf{d}_N^*, \mathsf{d}_T^*$.

$\longrightarrow \mathsf{c}_N^*, \mathsf{c}_T^*, \mathsf{d}_N^*, \mathsf{d}_T^* \longrightarrow$

Both define the constraints $\mathbf{Q}$ and the degrees $\mathbf{d}^*$:

$$Q_1(\cdot) = N_u^*(X) - N_{u,0}(X)T_{u,1}(X) - \gamma N_{u,1}(X)T_{u,0}(X)$$
$$Q_2(\cdot) = N_v^*(X) - N_{v,0}(X)T_{v,1}(X) - \gamma N_{v,1}(X)T_{v,0}(X)$$
$$Q_3(\cdot) = T_u^*(X) - T_{u,0}(X)T_{u,1}(X), \quad Q_4(\cdot) = T_v^*(X) - T_{v,0}(X)T_{v,1}(X)$$

$$d_{N^*,u} = \max\left\{d_{N,u,b} + d_{T,u,1-b}\right\}_b, \quad d_{N^*,v} = \max\left\{d_{N,v,b} + d_{T,v,1-b}\right\}_b$$
$$d_{T^*,u} = d_{T,u,0} + d_{T,u,1}, \quad d_{T^*,v} = d_{T,v,0} + d_{T,v,1}$$
$$\mathbf{d}^* = (d_{N^*,u}, d_{T^*,u}, d_{N^*,v}, d_{T^*,v})$$

Both compute $y^* = y_0 + \gamma z + \gamma^2 y_1$ and define $\mathsf{x}^* = (\mathsf{c}_N^*, \mathsf{c}_T^*, \mathsf{d}_N^*, \mathsf{d}_T^*, \mathbf{d}^*, y^*)$

$$\mathsf{x}_{\mathsf{PCC}} = (\mathsf{c}_N^*, \mathsf{c}_T^*, \mathsf{d}_N^*, \mathsf{d}_T^*, \mathsf{c}_{N,0}, \mathsf{c}_{T,0}, \mathsf{d}_{N,0}, \mathsf{d}_{T,0}, \mathsf{c}_{N,1}, \mathsf{c}_{T,1}, \mathsf{d}_{N,1}, \mathsf{d}_{T,1}, (\mathbf{d}^*, \mathbf{d}_0, \mathbf{d}_1), \mathbf{Q})$$

Define $\mathsf{w}^* := (N_u^*(X), T_u^*(X), N_v^*(X), T_v^*(X), \mathbf{u}^*, \mathbf{v}^*)$,

$$\mathsf{w}_{\mathsf{PCC}} := \begin{pmatrix} N_u^*(X), T_u^*(X), N_v^*(X), T_v^*(X), \\ N_{u,0}(X), T_{u,0}(X), N_{v,0}(X), T_{v,0}(X), \\ N_{u,1}(X), T_{u,1}(X), N_{v,1}(X), T_{v,1}(X) \end{pmatrix}$$

Output $\mathsf{w}^*, \mathsf{w}_{\mathsf{PCC}}$ | Output $\mathsf{x}^*, \mathsf{x}_{\mathsf{PCC}}$

Figure 13: Reduction of knowledge $\Pi_{\mathbf{M}} : \mathcal{R}_{\mathbf{M}}^2 \to \mathcal{R}_{\mathbf{M}} \times \mathcal{R}_{\mathsf{PCC}}$.

Knowledge soundness. We show that there exists an extractor $\mathcal{X}$ that outputs a witness for $\mathcal{R}_{\mathbf{M}}$ given a 3-tree of accepting transcripts. Knowledge soundness then follows by Lemma 4.1.1. Consider such a tree produced with challenges $\gamma_1, \gamma_2, \gamma_3$. Each leaf is labeled by $\mathsf{x}_j^* = (\mathsf{c}_{N,j}^*, \mathsf{c}_{T,j}^*, \mathsf{d}_{N,j}^*, \mathsf{d}_{T,j}^*, \mathbf{d}_j^*, y_j^*)$ and $\mathsf{w}_j^* = (N_{u,j}^*(X), T_{u,j}^*(X), \mathbf{u}_j^*, N_{v,j}^*(X), T_{v,j}^*(X), \mathbf{v}_j^*)$ and the corresponding statements/witnesses $\mathsf{x}_{\mathsf{PCC},j}$, $\mathsf{w}_{\mathsf{PCC},j}$. Each leaf contains valid openings for the polynomials $N_{u,b}(X)$, $T_{u,b}(X)$, $N_{v,b}(X)$, $T_{v,b}(X)$ as part of the $\mathcal{R}_{\mathsf{PCC}}$. The extractor $\mathcal{X}$ outputs these witnesses from the first leaf γ_1 and computes $\mathbf{u}_b$ and $\mathbf{v}_b$ as the coefficient vectors, in the Lagrange basis, of $N_{u,b}(X)/nT_{u,b}(X)$ and $N_{v,b}(X)/nT_{v,b}(X)$,

respectively. Note that the polynomial commitment openings must be the same for all leaves by the binding property of PC. Furthermore, by validity of the $\mathcal{R}_{\mathsf{PCC}}$ statements it must be the case that

$$N_u^*(X) = N_{u,0}(X)T_{u,1}(X) + \gamma N_{u,1}(X)T_{u,0}(X), \qquad T_u^*(X) = T_{u,0}(X)T_{u,1}(X)$$
$$N_v^*(X) = N_{v,0}(X)T_{v,1}(X) + \gamma N_{v,1}(X)T_{v,0}(X), \qquad T_v^*(X) = T_{v,0}(X)T_{v,1}(X)$$

We next show that the extracted witnesses are valid witnesses for the relation $\mathcal{R}_{\mathbf{M}}$. We first argue that they must correspond to polynomials of degree less than n. We use the fact that for all leaves, the corresponding divided polynomials are of degree less than n. Writing these polynomials, we have that:

$$p_{u,j}^*(X) = \frac{N_{u,j}^*(X)}{T_{u,j}^*(X)} = \frac{N_{u,0}(X)}{T_{u,0}(X)} + \gamma_j \frac{N_{u,1}(X)}{T_{u,1}(X)} = p_{u,0}(X) + \gamma_j p_{u,1}(X)$$

for some rational functions $p_{u,0}(X), p_{u,1}(X)$. We can therefore define the system of equations:

$$p_{u,1}^*(X) = p_{u,0}(X) + \gamma_1 p_{u,1}(X)$$
$$p_{u,2}^*(X) = p_{u,0}(X) + \gamma_2 p_{u,1}(X)$$

or equivalently

$$\begin{pmatrix} p_{u,1}^*(X) \\ p_{u,2}^*(X) \end{pmatrix} = \mathbf{\Gamma}_2 \begin{pmatrix} p_{u,0}(X) \\ p_{u,1}(X) \end{pmatrix} \Rightarrow \begin{pmatrix} p_{u,0}(X) \\ p_{u,1}(X) \end{pmatrix} = \mathbf{\Gamma}_2^{-1} \begin{pmatrix} p_{u,1}^*(X) \\ p_{u,2}^*(X) \end{pmatrix}$$

since $\mathbf{\Gamma}_2$ is invertible. Therefore, the rational functions $p_{u,0}(X), p_{u,1}(X)$ are linear combinations of polynomials of degree less than n and therefore they also have degree less than n. Defining the same system of equations to the coefficients $\mathbf{u}_j^*, \mathbf{u}_0, \mathbf{u}_1$, we also deduce that $\mathbf{u}_j^* = \mathbf{u}_0 + \gamma \mathbf{u}_1$. We work similarly to argue about the degree of the other polynomials.

It remains to show that y_0, y_1 are correct, that is, they correspond to $y_b = \mathbf{u}_b^T \mathbf{M} \mathbf{v}_b$. Since the leaves contain valid witnesses, we have that the openings of the divided polynomials $\mathbf{u}_j^*, \mathbf{v}_j^*$ satisfy:

$$\begin{aligned} y_j^* &= \mathbf{u}_j^{*\top} \mathbf{M} \mathbf{v}_j^* = (\mathbf{u_0} + \gamma_j \mathbf{u}_1)^\top \mathbf{M} (\mathbf{v_0} + \gamma_j \mathbf{v}_1) \\ &= \mathbf{u}_0^\top \mathbf{M} \mathbf{v}_0 + \gamma_j (\mathbf{u}_0^\top \mathbf{M} \mathbf{v}_1 + \mathbf{u}_1^\top \mathbf{M} \mathbf{v}_0) + \gamma_j^2 \mathbf{u_1}^\top \mathbf{M} \mathbf{v}_1 \end{aligned}$$

Let $s = (\mathbf{u}_0^\top \mathbf{M} \mathbf{v}_1 + \mathbf{u}_1^\top \mathbf{M} \mathbf{v}_0)$. We have that

$$\begin{pmatrix} y_1^* \\ y_2^* \\ y_3^* \end{pmatrix} = \mathbf{\Gamma}_3 \begin{pmatrix} \mathbf{u}_0^\top \mathbf{M} \mathbf{v}_0 \\ s \\ \mathbf{u}_1^\top \mathbf{M} \mathbf{v}_1 \end{pmatrix} \Rightarrow \begin{pmatrix} \mathbf{u}_0^\top \mathbf{M} \mathbf{v}_0 \\ s \\ \mathbf{u}_1^\top \mathbf{M} \mathbf{v}_1 \end{pmatrix} = \mathbf{\Gamma}_3^{-1} \begin{pmatrix} y_1^* \\ y_2^* \\ y_3^* \end{pmatrix} = \mathbf{\Gamma}_3^{-1} \mathbf{\Gamma}_3 \begin{pmatrix} y_0 \\ z \\ y_1 \end{pmatrix} = \begin{pmatrix} y_0 \\ z \\ y_1 \end{pmatrix}$$

where we used the fact that $\mathbf{\Gamma}_3$ is invertible. Therefore, the extracted witness satisfies: $\mathbf{u}_0^\top \mathbf{M} \mathbf{v}_0 = y_0$ and $\mathbf{u}_1^\top \mathbf{M} \mathbf{v}_1 = y_1$.

Efficiency. The prover needs $O(s_{\mathbf{M}}\ \mathbb{F})$ time to compute the cross-term and it needs to compute and commit to four polynomials of sparsity at most $2s_p$, since they are products of polynomials with sparsity s_p. Computing such polynomials can be done using FFT techniques with $\mathcal{O}(s_p \cdot \log s_p\ \mathbb{F})$ operations and committing to these polynomials takes time $t_{\mathsf{Commit}}(2 \cdot s_p)$. Therefore, its total proving time is $\mathcal{O}((s_{\mathbf{M}} + s_p \log s_p)\ \mathbb{F} + t_{\mathsf{Commit}}(2 \cdot s_p))$. The verifier only needs to compute the new statement y^* which consists of constant field operations. The proof size consists of the field element y and 4 commitments and therefore its size is $\mathcal{O}(1\ \ |\mathsf{PC}| + 1 \cdot \mathbb{F})$ and the statement of $\mathsf{x}_{\mathsf{PCC}}$ involves constant commitments and some constant amount of information to encode the degrees and the constraints, therefore its size is $\mathsf{x}_{\mathsf{PCC}}$ is $\mathcal{O}(1\ \ |\mathsf{PC}| + 1\ \mathbb{F})$. □

8.2 A Local Folding Scheme for Bivariate Polynomial Evaluation

Let $P(X, Y) \in \mathbb{F}[X, Y]$ be a fixed bivariate polynomial of individual degree at most $n - 1$ and sparsity $\|P\| = s$ and consider the relation

$$\mathcal{R}_P = \left\{ (\epsilon, (\alpha, \beta, y), \epsilon) \,\middle|\, P(\alpha, \beta) = y \right\}$$

Note that the relation is efficiently decidable. Next, consider the matrix $\mathbf{M} = (m_{ij}) \in \mathbb{F}^{n \times n}$ where $m_{ij} = P(\mathsf{h}_i, \mathsf{h}_j)$ where $\mathbb{H} = \{\mathsf{h}_i\}_{i=1}^n$ is a subgroup of $\mathbb{F}$. It holds that

$$P(\alpha, \beta) = y \Leftrightarrow \boldsymbol{\lambda}(\alpha)^\top \mathbf{M} \boldsymbol{\lambda}(\beta) = y$$

We can therefore reduce membership to $\mathcal{R}_P$ to membership in $\mathcal{R}_\mathbf{M}$ by computing

$$N_u(X) = (X^n - 1)\alpha - (\alpha^n - 1)X, \qquad T_u(X) = (X - \alpha)$$
$$N_v(X) = (X^n - 1)\beta - (\beta^n - 1)X, \qquad T_v(X) = (X - \beta)$$

where N_u, N_v are of degree n and have sparsity 3 and T_u, T_v are of degree 1 and have sparsity 2, and the corresponding commitments $\mathsf{c}_n, \mathsf{c}_t, \mathsf{d}_n, \mathsf{d}_t$. Setting $\mathsf{x}_\mathbf{M} := (\mathsf{c}_n, \mathsf{c}_t, \mathsf{d}_n, \mathsf{d}_t, \mathbf{d}, y)$ we have that

$$(\alpha, \beta, y) \in \mathcal{R}_P \Leftrightarrow \exists \mathsf{w}_\mathbf{M} \text{ s.t. } (\mathsf{pp}, \mathsf{x}_\mathbf{M}, \mathsf{w}_\mathbf{M}) \in \mathcal{R}_\mathbf{M}$$

where pp are the parameters of PC. Indeed, this holds since $N_u(X)/nT_u(X) = \boldsymbol{\lambda}(\alpha)^\top \boldsymbol{\lambda}(X)$ and respectively for β, and therefore

$$y = \boldsymbol{\lambda}(\alpha)^\top \mathbf{M} \boldsymbol{\lambda}(\beta) \Rightarrow P(\alpha, \beta) = y$$

Computing the new statement from (α, β, y) consists of computing the four sparse polynomials –which involves constant field operations– and committing them. The verifier can therefore reduce its bivariate polynomial statement to a statement of (the more general) $\mathcal{R}_\mathbf{M}$ with a constant number of group operations. Its new goal is to verify (with the help of the prover) the latter which would imply the correctness of the bivariate evaluation claim. Computing the corresponding witness can be done in $\mathcal{O}(n \log n)$ operations by evaluating the vectors $\boldsymbol{\lambda}(\alpha), \boldsymbol{\lambda}(\beta)$.

We can now apply the compiler of Sec. 7.2 to the reduction of knowledge $\Pi_\mathbf{M}$ to derive a K-Local folding scheme for $\mathcal{R}_\mathbf{M}$. We are interested in the corresponding efficiency measure when we use the scheme with inputs as above.

We first present a lemma bounding the degrees and maximum sparsity of the polynomials N_u, N_v, T_u, T_v during the execution of Λ'_κ.

Lemma 8.2.1. *Let $\kappa \in \mathbb{N}$ and $(\mathsf{pp}, \mathsf{x}_b, \mathsf{w}_b)_{\mathbf{b} \in \{0,1\}^\kappa}$ be statements of $\mathcal{R}_\mathbf{M}$ derived from statements (α, β, y) of $\mathcal{R}_P$. Consider an honest execution of the $\mathcal{R}_{\mathsf{PCC}}$-promise local folding scheme $\Lambda'_\kappa.\mathsf{LProve}$ derived by applying the transformation of Sec. 7.1 with these inputs. Then for all $1 \leq i \leq \kappa$ the polynomials N_u, N_v, T_u, T_v produced as part of the $\mathsf{x}_{\mathsf{PCC}}$ statement during the execution of $\Lambda'_i.\mathsf{LProve}$ satisfy the following:*

- *the degrees of T_u, T_v are at most 2^i and their sparsity at most $2^i + 1$,*
- *the degrees of N_u, N_v are at most $n + 2^i$ and their sparsity at most $2^{i+1} + 2$,*

Proof. We only consider the polynomials T_u, N_u since they are of the exact same form as T_v, N_v. We start by showing the claim for T_u. Recall that $T_u^*(X) = T_{u,0}(X) \cdot T_{u,1}(X)$. Since in the initial round the polynomials are linear and in each round their degree doubles, the claim follows. For the polynomial $N_u^*(X)$ we show something stronger, namely, we claim that N_u^* can be written as $N_u^*(X) = p^*(X) \cdot X^n + q^*(X)$ where the degrees of p, q are at most 2^i. We show this by induction on i.

Base Case: Note that the initial polynomials are of the form $N_{u,b}(X) = p_b(X)X^n + q_b(X)$ where the degrees of p_b, q_b are at most 1. The fresh polynomial is constructed as

$$\begin{aligned} N_u^*(X) &= N_{u,0}(X)T_{u,1}(X) + \gamma N_{u,1}(X)T_{u,0}(X) \\ &= (p_0(X)X^n + q_0(X))T_{u,1}(X) + \gamma(p_1(X)X^n + q_1(X))T_{u,0}(X) \\ &= (p_0(X) + \gamma p_1(X))T_{u,1}(X)X^n + (q_0(X) + \gamma q_1(X))T_{u,0}(X) \end{aligned}$$

Define

$$p^*(X) = (p_0(X) + \gamma p_1(X))T_{u,1}(X), \qquad q^*(X) = (q_0(X) + \gamma q_1(X))T_{u,0}(X)$$

and note that $N_u^*(X) = p^*(X)X^n + q^*(X)$ and also that $\deg(p^*), \deg(q^*) \leq 2$ since $\deg(p_b) \leq 1$ and $\deg(T_{u,b}) \leq 1$.

Inductive Case: We work similarly. In the i-th iteration, the fresh polynomial is constructed as

$$\begin{aligned} N_u^*(X) &= N_{u,0}(X)T_{u,1}(X) + \gamma N_{u,1}(X)T_{u,0}(X) \\ &= (p_0(X)X^n + q_0(X))T_{u,1}(X) + \gamma(p_1(X)X^n + q_1(X))T_{u,0}(X) \\ &= (p_0(X) + \gamma p_1(X))T_{u,1}(X)X^n + (q_0(X) + \gamma q_1(X))T_{u,0}(X) \end{aligned}$$

where $N_{u,0}(X), N_{u,1}(X), T_{u,0}(X), T_{u,1}(X)$ are the polynomials of the previous iteration. By induction hypothesis we have that $N_{u,b}(X) = p_b(X)X^n + q_b(X)$ where $\deg(p_b), \deg(q_b) \leq 2^{i-1}$. Define

$$p^*(X) = (p_0(X) + \gamma p_1(X))T_{u,1}(X), \qquad q^*(X) = (q_0(X) + \gamma q_1(X))T_{u,0}(X)$$

and note that $N_u^*(X) = p^*(X)X^n + q^*(X)$ and also that $\deg(p^*), \deg(q^*) \leq 2^i$ since $\deg(p_b) \leq 2^{i-1}$ and $\deg(T_{u,1}) \leq 2^{i-1}$. This gives $\deg(N_u^*) \leq n + 2^i$ and sparsity at most $2^{i+1} + 2$, as claimed.

□

In the next corollary we consider this efficiency when we instantiate the compiler of Sec. 7.2 with the KZG commitment scheme [KZG10].

Corollary 8.2.1. *Let $\mathbf{M} \in \mathbb{F}^{n \times n}$ be a matrix with sparsity $\|\mathbf{M}\| = s_{\mathbf{M}} = \omega(K)$ and let $\Lambda_{\mathbf{M},\kappa} = \mathsf{L}(\mathsf{FS}(\Pi_{\mathbf{M}}), \kappa)$ using the KZG commitment scheme. Let $(\mathsf{pp}, \mathsf{x}_b, \mathsf{w}_b)_{\mathbf{b}\in\{0,1\}^\kappa}$ be statements derived from statements (α, β, y) of $\mathcal{R}_P$ using the KZG commitment scheme and consider an execution of $\Lambda_{\mathbf{M},\kappa}.\mathsf{LProve}$ and $\Lambda_{\mathbf{M},\kappa}.\mathsf{LVerify}$ with such inputs. Then*

- *the complexity of LProve is dominated by*

$$\Big(K \cdot s_{\mathbf{M}} + K \cdot \log^2 K + (n + K) \cdot \log(n + K)\Big)\,\mathbb{F} + (n + K \log K)\,\mathbb{G}$$

  the complexity of $\mathsf{LVerify}$ is dominated by $1 \cdot \mathsf{P} + \log K \cdot (\mathbb{F} + \mathbb{G} + \rho)$, and the proof size is $\lambda \cdot \log K$.

where $\mathbb{G}$, $\mathbb{F}$ denote group and field operations, P pairing operations, and ρ random oracle queries.

Proof. The proof size and the verification time can be derived directly by Thm 7.2.1 by noting that (1) all reductions of knowledge have constant-time verification and that all statements/proofs shared are of size λ when using KZG. For the complexity of the prover, we need a fine-grained analysis of the derived construction that takes into account the sparsity of the polynomials and how it grows during the recursive calls; we use Lemma 8.2.1 for this. We next analyze the proving time by considering the proving time during each of the five steps of $\Lambda_{\mathbf{M},\kappa}.\mathsf{Prove}$.

Invocation of Λ'_κ.Prove: We start by considering the first part of the compiler, namely calling the promise local folding scheme derived from $\mathcal{R}_{\mathbf{M}}$. We claim that in the i-th iteration, the proving cost for a single iteration of $\mathsf{FS}(\Pi_{\mathbf{M}})$.Prove is $(s_{\mathbf{M}} + i \cdot 2^i)\mathbb{F} + 2^i\mathbb{G}$. Indeed, by Lemma 8.2.1, during iteration i, the polynomials involved have sparsity $\mathcal{O}(2^i)$. Since in KZG, committing to an s_i sparse polynomial needs s_i group operations, by Thm. 8.1.1 the proving cost is $\mathcal{O}((s_{\mathbf{M}} + i \cdot 2^i) \cdot \mathbb{F} + t_{\mathsf{Commit}}(2^i))$. Finally, on the i-th iteration, $\mathsf{FS}(\Pi_{\mathbf{M}})$.Prove is invoked $\mathcal{O}(2^{\kappa-i})$ times. Putting it all together, the total asymptotic cost of the i-th iteration is

$$2^{\kappa-i} \cdot \Big((s_{\mathbf{M}} + i \cdot 2^i)\mathbb{F} + 2^i\mathbb{G}\Big) = (2^{\kappa-i} \cdot s_{\mathbf{M}} + i \cdot 2^\kappa)\mathbb{F} + 2^\kappa\mathbb{G}$$

Summing over all κ rounds we have that the total asymptotic proving cost for Λ'_κ is

$$\begin{aligned}\sum_{i=1}^{\kappa} \Big(2^{\kappa-i} \cdot s_{\mathbf{M}}\ \mathbb{F} + i \cdot 2^\kappa\ \mathbb{F} + 2^\kappa\ \mathbb{G}\Big) &= \sum_{i=1}^{\kappa} 2^{\kappa-i} \cdot s_{\mathbf{M}}\ \mathbb{F} + 2^\kappa \sum_{i=1}^{\kappa} i\ \mathbb{F} + 1 \cdot \mathbb{G} \\ &= 2^\kappa \cdot s_{\mathbf{M}}\ \mathbb{F} + 2^\kappa \cdot \kappa^2\ \mathbb{F} + 2^\kappa \cdot \kappa\mathbb{G} \\ &= K \cdot s_{\mathbf{M}}\ \mathbb{F} + K \cdot \log^2 K\ \mathbb{F} + K \log K\mathbb{G}\end{aligned}$$

Invocation of $\mathsf{NIRoK}_{\mathsf{D}}$: The prover scales each polynomial by multiplying it with an appropriate power X^k. The sparsity of each polynomial is preserved and the scaling is dominated by an MSM of size the sparsity of the polynomial. Arguing as in the previous case, this corresponds to $\mathcal{O}(K \cdot \log K\ \mathbb{F} + K \log K\mathbb{G})$.

Invocation of $\widetilde{\mathsf{NIRoK}}_{\mathsf{PC}}$: First, the prover makes $\mathcal{O}(K)$ calls to the random oracles to compute the common challenge. Then, for each statement, it performs an execution of $\mathsf{FS}(\Pi_{\mathsf{PC}})$.Prove that involves $\mathcal{O}(\log K)$ polynomials with sparsity at most $\mathcal{O}(K)$. Each execution is dominated by evaluating these polynomials which can be done with $\mathcal{O}(K \log K)$ field operations. Noting also that for each execution, the sparsities of the $\log K$ polynomials are $1, 2, 4, \ldots, K$ (omitting some constant additive terms) the total time to evaluate them is $\mathcal{O}(K)$. Then, it executes $\mathsf{FS}(\Pi_{\mathsf{PCO}})$.Prove to reduce each claim to a single $\mathcal{R}_{\mathsf{PCO}}$ claim. Each execution needs to homomorphically combine $\mathcal{O}(1)$ polynomials of sparsity at most $\mathcal{O}(K)$. This can be done in $\mathcal{O}(K^2\ \mathbb{F} + K\ \mathbb{G})$ operations. Therefore, the total time of the prover during this step is $\mathcal{O}(K^2\ \mathbb{F} + K\ \mathbb{G})$.

Invocation of $\Lambda_{\mathsf{PCO},\kappa}$: The prover performs $\mathcal{O}(K)$ executions of $\mathsf{FS}(\Pi_{\mathsf{PCO}(x)})$. In each, it homomorphically combines two polynomials of sparsity at most $\mathcal{O}(K)$. Therefore, the total time of the prover during this step is $\mathcal{O}(K^2\ \mathbb{F} + K\ \mathbb{G})$

PC evaluation: The evaluation proof of the final commitment $\mathsf{x}_{\mathsf{PCO}}$ involves opening a polynomial commitment, which by Lemma 8.2.1 is of degree at most $n + K$. This can be done by $(n + K) \cdot \log(n + K)\ \mathbb{F} + (n + K)\mathbb{G}$ operations.

Summing all these up and taking into account that $s = \omega(K)$, we conclude the total proving time is

$$\Big(K \cdot s_{\mathbf{M}} + K \cdot \log^2 K + (n + K) \cdot \log(n + K)\Big)\ \mathbb{F} + (n + K \log K)\ \mathbb{G}$$

□

Acknowledgments

This work is part of Maria de Maeztu Units of Excellence Programme CEX2021-001195-M, funded by MICIU/AEI /10.13039/501100011033. This work was partially supported by grant AEI-PID2021-128521OB-I00 of the Spanish Ministry of Science and Innovation and by the ARTEMISA International

Chair in Cybersecurity (C057/23), funded by the Spanish National Cybersecurity Institute (INCIBE) through the European Union NextGenerationEU and the Recovery, Transformation and Resilience Plan. This research was partially funded by SPRIND (the German Federal Agency for Breakthrough Innovation) and Common Codes. The authors are solely responsible for the content of this publication; the positions presented here do not reflect the views of SPRIND or Common Codes.

References

[ABST23] Miguel Ambrona, Marc Beunardeau, Anne-Laure Schmitt, and Raphael R. Toledo. "aPlonK: Aggregated PlonK from Multi-polynomial Commitment Schemes". In: *IWSEC 23*. Ed. by Junji Shikata and Hiroki Kuzuno. Vol. 14128. LNCS. Springer, Cham, Aug. 2023, pp. 195–213. DOI: 10.1007/978-3-031-41326-1_11.

[ABVT26] Mohammed Alghazwi, Tariq Bontekoe, Leon Visscher, and Fatih Turkmen. "Collaborative CP-NIZKs: modular, composable proofs for distributed secrets". In: *Int. J. Inf. Secur.* 25.2 (Mar. 2026). ISSN: 1615-5262. DOI: 10.1007/s10207-026-01209-1. URL: https://doi.org/10.1007/s10207-026-01209-1.

[ACGS24] Diego F. Aranha, Anamaria Costache, Antonio Guimarães, and Eduardo Soria-Vazquez. "HELIOPOLIS: Verifiable Computation over Homomorphically Encrypted Data from Interactive Oracle Proofs is Practical". In: *ASIACRYPT 2024, Part V*. Ed. by Kai-Min Chung and Yu Sasaki. Vol. 15488. LNCS. Springer, Singapore, Dec. 2024, pp. 302–334. DOI: 10.1007/978-981-96-0935-2_10.

[BC23] Benedikt Bünz and Binyi Chen. "Protostar: Generic Efficient Accumulation/Folding for Special-Sound Protocols". In: *ASIACRYPT 2023, Part II*. Ed. by Jian Guo and Ron Steinfeld. Vol. 14439. LNCS. Springer, Singapore, Dec. 2023, pp. 77–110. DOI: 10.1007/978-981-99-8724-5_3.

[BC25a] Dan Boneh and Binyi Chen. "LatticeFold: A Lattice-Based Folding Scheme and Its Applications to Succinct Proof Systems". In: *ASIACRYPT 2025, Part III*. Ed. by Goichiro Hanaoka and Bo-Yin Yang. Vol. 16247. LNCS. Springer, Singapore, Dec. 2025, pp. 330–362. DOI: 10.1007/978-981-95-5099-9_11.

[BC25b] Dan Boneh and Binyi Chen. "LatticeFold+: Faster, Simpler, Shorter Lattice-Based Folding for Succinct Proof Systems". In: *CRYPTO 2025, Part VII*. Ed. by Yael Tauman Kalai and Seny F. Kamara. Vol. 16006. LNCS. Springer, Cham, Aug. 2025, pp. 327–361. DOI: 10.1007/978-3-032-01907-3_11.

[BCL+21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. "Proof-Carrying Data Without Succinct Arguments". In: *CRYPTO 2021, Part I*. Ed. by Tal Malkin and Chris Peikert. Vol. 12825. LNCS. Virtual Event: Springer, Cham, Aug. 2021, pp. 681–710. DOI: 10.1007/978-3-030-84242-0_24.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. "Recursive Proof Composition from Accumulation Schemes". In: *TCC 2020, Part II*. Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12551. LNCS. Springer, Cham, Nov. 2020, pp. 1–18. DOI: 10.1007/978-3-030-64378-2_1.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. "Aurora: Transparent Succinct Arguments for R1CS". In: *EUROCRYPT 2019, Part I*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11476. LNCS. Springer, Cham, May 2019, pp. 103–128. DOI: 10.1007/978-3-030-17653-2_4.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *TCC 2016-B, Part II*. Ed. by Martin Hirt and Adam D. Smith. Vol. 9986. LNCS. Springer, Berlin, Heidelberg, 2016, pp. 31–60. DOI: 10.1007/978-3-662-53644-5_2.

[BCS21] Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. "Sumcheck Arguments and Their Applications". In: *CRYPTO 2021, Part I*. Ed. by Tal Malkin and Chris Peikert. Vol. 12825. LNCS. Virtual Event: Springer, Cham, Aug. 2021, pp. 742–773. DOI: 10.1007/978-3-030-84242-0_26.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. *Halo: Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Report 2019/1021. 2019. URL: https://eprint.iacr.org/2019/1021.

[BMM+21] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. "Proofs for Inner Pairing Products and Applications". In: *ASIACRYPT 2021, Part III*. Ed. by Mehdi Tibouchi and Huaxiong Wang. Vol. 13092. LNCS. Springer, Cham, Dec. 2021, pp. 65–97. DOI: 10.1007/978-3-030-92078-4_3.

[BMNW25a] Benedikt Bünz, Pratyush Mishra, Wilson Nguyen, and William Wang. "Accumulation Without Homomorphism". In: *ITCS 2025*. Ed. by Raghu Meka. Vol. 325. LIPIcs, Jan. 2025, 23:1–23:25. DOI: 10.4230/LIPIcs.ITCS.2025.23.

[BMNW25b] Benedikt Bünz, Pratyush Mishra, Wilson Nguyen, and William Wang. "Arc: Accumulation for Reed-Solomon Codes". In: *CRYPTO 2025, Part VII*. Ed. by Yael Tauman Kalai and Seny F. Kamara. Vol. 16006. LNCS. Springer, Cham, Aug. 2025, pp. 128–160. DOI: 10.1007/978-3-032-01907-3_5.

[CFF+21] Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. "Lunar: A Toolbox for More Efficient Universal and Updatable zkSNARKs and Commit-and-Prove Extensions". In: *ASIACRYPT 2021, Part III*. Ed. by Mehdi Tibouchi and Huaxiong Wang. Vol. 13092. LNCS. Springer, Cham, Dec. 2021, pp. 3–33. DOI: 10.1007/978-3-030-92078-4_1.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *EUROCRYPT 2020, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. LNCS. Springer, Cham, May 2020, pp. 738–768. DOI: 10.1007/978-3-030-45721-1_26.

[CLMZ23] Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. "Eos: Efficient Private Delegation of zkSNARK Provers". In: *USENIX Security 2023*. Ed. by Joseph A. Calandrino and Carmela Troncoso. USENIX Association, Aug. 2023, pp. 6453–6469. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/chiesa.

[CNR+22] Matteo Campanelli, Anca Nitulescu, Carla Ràfols, Alexandros Zacharakis, and Arantxa Zapico. "Linear-Map Vector Commitments and Their Practical Applications". In: *ASIACRYPT 2022, Part IV*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13794. LNCS. Springer, Cham, Dec. 2022, pp. 189–219. DOI: 10.1007/978-3-031-22972-5_7.

[CT10] Alessandro Chiesa and Eran Tromer. "Proof-Carrying Data and Hearsay Arguments from Signature Cards". In: *ICS 2010*. Ed. by Andrew Chi-Chih Yao. Tsinghua University Press, Jan. 2010, pp. 310–331.

[CY24] Alessandro Chiesa and Eylon Yogev. *Building Cryptographic Proofs from Hash Functions*. 2024. URL: https://github.com/hash-based-snargs-book.

[EG23] Liam Eagen and Ariel Gabizon. *ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances*. Cryptology ePrint Archive, Report 2023/1106. 2023. URL: https://eprint.iacr.org/2023/1106.

[FS87] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. LNCS. Springer, Berlin, Heidelberg, Aug. 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12.

[GBK+25] Mariana Gama, Emad Heydari Beni, Jiayi Kang, Jannik Spiessens, and Frederik Vercauteren. "Blind zkSNARKs: for Private Proof Delegation and Verifiable Computation over Encrypted Data". In: *CiC* 2.3 (2025), p. 10. DOI: 10.62056/akgyl8n4e.

[GGJ+23] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. "zkSaaS: Zero-Knowledge SNARKs as a Service". In: *USENIX Security 2023*. Ed. by Joseph A. Calandrino and Carmela Troncoso. USENIX Association, Aug. 2023, pp. 4427–4444. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/garg.

[GGJ+25] Sanjam Garg, Aarushi Goel, Abhishek Jain, Bhaskar Roberts, and Sruthi Sekar. "Malicious Security in Collaborative zk-SNARKs: More than Meets the Eye". In: *CRYPTO 2025, Part VII*. Ed. by Yael Tauman Kalai and Seny F. Kamara. Vol. 16006. LNCS. Springer, Cham, Aug. 2025, pp. 396–428. DOI: 10.1007/978-3-032-01907-3_13.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating computation: interactive proofs for muggles". In: *40th ACM STOC*. Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 113–122. DOI: 10.1145/1374376.1374396.

[GMN22] Nicolas Gailly, Mary Maller, and Anca Nitulescu. "SnarkPack: Practical SNARK Aggregation". In: *FC 2022*. Ed. by Ittay Eyal and Juan A. Garay. Vol. 13411. LNCS. Springer, Cham, May 2022, pp. 203–229. DOI: 10.1007/978-3-031-18283-9_10.

[GPPS26] Chaya Ganesh, Sikhar Patranabis, Shubh Prakash, and Nitin Singh. "GAPP: Generic Aggregation of Polynomial IOPs". In: *CRYPTO 2026, Part IX*. Ed. by Nadia Heninger and Mike Rosulek. Vol. 16808. LNCS. Springer, Cham, Aug. 2026, pp. 330–361. DOI: 10.1007/978-3-032-35424-2_12.

[Gro16] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *EUROCRYPT 2016, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. LNCS. Springer, Berlin, Heidelberg, May 2016, pp. 305–326. DOI: 10.1007/978-3-662-49896-5_11.

[GW20] Ariel Gabizon and Zachary J. Williamson. *plookup: A simplified polynomial protocol for lookup tables*. Cryptology ePrint Archive, Report 2020/315. 2020. URL: https://eprint.iacr.org/2020/315.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Report 2019/953. 2019. URL: https://eprint.iacr.org/2019/953.

[HMW+25] Yuncong Hu, Pratyush Mishra, Xiao Wang, Jie Xie, Kang Yang, Yu Yu, and Yuwen Zhang. "DFS: Delegation-friendly zkSNARK and Private Delegation of Provers". In: *USENIX Security 2025*. Ed. by Lujo Bauer and Giancarlo Pellegrino. USENIX Association, Aug. 2025, pp. 2065–2084. URL: https://www.usenix.org/conference/usenixsecurity25/presentation/hu-yuncong.

[Kil92] Joe Kilian. "A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract)". In: *24th ACM STOC*. ACM Press, May 1992, pp. 723–732. DOI: 10.1145/129712.129782.

[Kot24] Abhiram Kothapalli. "A Theory of Composition for Proofs of Knowledge". PhD thesis. Carnegie Mellon University, USA, 2024. DOI: 10.1184/R1/25900222.V1. URL: https://doi.org/10.1184/r1/25900222.v1.

[KP23] Abhiram Kothapalli and Bryan Parno. "Algebraic Reductions of Knowledge". In: *CRYPTO 2023, Part IV*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14084. LNCS. Springer, Cham, Aug. 2023, pp. 669–701. DOI: 10.1007/978-3-031-38551-3_21.

[KS24] Abhiram Kothapalli and Srinath T. V. Setty. "HyperNova: Recursive Arguments for Customizable Constraint Systems". In: *CRYPTO 2024, Part X*. Ed. by Leonid Reyzin and Douglas Stebila. Vol. 14929. LNCS. Springer, Cham, Aug. 2024, pp. 345–379. DOI: 10.1007/978-3-031-68403-6_11.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. “Nova: Recursive Zero-Knowledge Arguments from Folding Schemes”. In: *CRYPTO 2022, Part IV*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13510. LNCS. Springer, Cham, Aug. 2022, pp. 359–388. DOI: 10.1007/978-3-031-15985-5_13.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. “Constant-Size Commitments to Polynomials and Their Applications”. In: *ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, Berlin, Heidelberg, Dec. 2010, pp. 177–194. DOI: 10.1007/978-3-642-17373-8_11.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. “Algebraic Methods for Interactive Proof Systems”. In: *Journal of the ACM* 39.4 (1992), pp. 859–868.

[LXZ+24] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. “Pianist: Scalable zkRollups via Fully Distributed Zero-Knowledge Proofs”. In: *2024 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2024, pp. 1777–1793. DOI: 10.1109/SP54263.2024.00035.

[LZL+25] Chongrong Li, Pengfei Zhu, Yun Li, Cheng Hong, Wenjie Qu, and Jiaheng Zhang. “HyperPianist: Pianist with Linear-Time Prover and Logarithmic Communication Cost”. In: *2025 IEEE Symposium on Security and Privacy*. Ed. by Marina Blanton, William Enck, and Cristina Nita-Rotaru. IEEE Computer Society Press, May 2025, pp. 3383–3401. DOI: 10.1109/SP61157.2025.00202.

[LZW+25] Xuanming Liu, Zhelei Zhou, Yinghao Wang, Yanxin Pang, Jinye He, Bingsheng Zhang, Xiaohu Yang, and Jiaheng Zhang. “Scalable Collaborative zk-SNARK and Its Application to Fully Distributed Proof Delegation”. In: *USENIX Security 2025*. Ed. by Lujo Bauer and Giancarlo Pellegrino. USENIX Association, Aug. 2025, pp. 2025–2044. URL: https://www.usenix.org/conference/usenixsecurity25/presentation/liu-xuanming.

[Mic94] Silvio Micali. “CS Proofs (Extended Abstracts)”. In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 436–453. DOI: 10.1109/SFCS.1994.365746.

[NPR26] Anca Nitulescu, Nikitas Paslis, and Carla Ràfols. “FLIP-and-Prove R1CS”. In: *CiC* 2.4 (2026), p. 6. DOI: 10.62056/ayc3tx4e-.

[OB22] Alex Ozdemir and Dan Boneh. “Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets”. In: *USENIX Security 2022*. Ed. by Kevin R. B. Butler and Kurt Thomas. USENIX Association, Aug. 2022, pp. 4291–4308. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/ozdemir.

[PRZ26] Nikitas Paslis, Carla Ràfols, and Alexandros Zacharakis. “Proof-Carrying Data via Holography Accumulation”. In: *SCN 2026*. Ed. by Clemente Galdi and Luisa Siniscalchi. Vol. 16810. LNCS. Springer, 2026.

[RZ21] Carla Ràfols and Arantxa Zapico. “An Algebraic Framework for Universal and Updatable SNARKs”. In: *CRYPTO 2021, Part I*. Ed. by Tal Malkin and Chris Peikert. Vol. 12825. LNCS. Virtual Event: Springer, Cham, Aug. 2021, pp. 774–804. DOI: 10.1007/978-3-030-84242-0_27.

[RZ23] Carla Ràfols and Alexandros Zacharakis. “Folding Schemes with Selective Verification”. In: *LATINCRYPT 2023*. Ed. by Abdelrahaman Aly and Mehdi Tibouchi. Vol. 14168. LNCS. Springer, Cham, Oct. 2023, pp. 229–248. DOI: 10.1007/978-3-031-44469-2_12.

[SCP+22] Shravan Srinivasan, Alexander Chepurnoy, Charalampos Papamanthou, Alin Tomescu, and Yupeng Zhang. “Hyperproofs: Aggregating and Maintaining Proofs in Vector Commitments”. In: *USENIX Security 2022*. Ed. by Kevin R. B. Butler and Kurt Thomas.

USENIX Association, Aug. 2022, pp. 3001–3018. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/srinivasan.

[Set20] Srinath Setty. "Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup". In: *CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, Cham, Aug. 2020, pp. 704–737. DOI: 10.1007/978-3-030-56877-1_25.

[STW23] Srinath Setty, Justin Thaler, and Riad Wahby. *Customizable constraint systems for succinct arguments*. Cryptology ePrint Archive, Report 2023/552. 2023. URL: https://eprint.iacr.org/2023/552.

[SVV16] Berry Schoenmakers, Meilof Veeningen, and Niels de Vreede. "Trinocchio: Privacy-Preserving Outsourcing by Distributed Verifiable Computation". In: *ACNS 2016*. Ed. by Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider. Vol. 9696. LNCS. Springer, Cham, June 2016, pp. 346–366. DOI: 10.1007/978-3-319-39555-5_19.

[TCZ+20] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan-Gueta, and Srinivas Devadas. "Towards Scalable Threshold Cryptosystems". In: *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2020, pp. 877–893. DOI: 10.1109/SP40000.2020.00059.

[Tha13] Justin Thaler. "Time-Optimal Interactive Proofs for Circuit Evaluation". In: *CRYPTO 2013, Part II*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8043. LNCS. Springer, Berlin, Heidelberg, Aug. 2013, pp. 71–89. DOI: 10.1007/978-3-642-40084-1_5.

[TKPS22] Ioanna Tzialla, Abhiram Kothapalli, Bryan Parno, and Srinath T. V. Setty. "Transparency Dictionaries with Succinct Proofs of Correct Operation". In: *NDSS 2022*. The Internet Society, Apr. 2022.

[Val08] Paul Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *TCC 2008*. Ed. by Ran Canetti. Vol. 4948. LNCS. Springer, Berlin, Heidelberg, Mar. 2008, pp. 1–18. DOI: 10.1007/978-3-540-78524-8_1.

[WJB+17] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, abhi shelat, Justin Thaler, Michael Walfish, and Thomas Wies. "Full Accounting for Verifiable Outsourcing". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, 2017, pp. 2071–2086. DOI: 10.1145/3133956.3133984.

[WTs+18] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. "Doubly-Efficient zkSNARKs Without Trusted Setup". In: *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 926–943. DOI: 10.1109/SP.2018.00060.

[WZC+18] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. "DIZK: A Distributed Zero Knowledge Proof System". In: *USENIX Security 2018*. Ed. by William Enck and Adrienne Porter Felt. USENIX Association, Aug. 2018, pp. 675–692. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/wu.

[YLF+22] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. "hbACSS: How to Robustly Share Many Secrets". In: *NDSS 2022*. The Internet Society, Apr. 2022.

[ZLW+21] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. "Doubly Efficient Interactive Proofs for General Arithmetic Circuits with Linear Prover Time". In: *ACM CCS 2021*. Ed. by Giovanni Vigna and Elaine Shi. ACM Press, Nov. 2021, pp. 159–177. DOI: 10.1145/3460120.3484767.

[ZWL+25] Xinxuan Zhang, Ruida Wang, Zeyu Liu, Binwu Xiang, Yi Deng, Ben Fisch, and Xianhui Lu. "Phalanx: An FHE-Friendly SNARK for Verifiable Computation on Encrypted Data". In: *ACM CCS 2025*. Ed. by Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng

Shieh, David Lie, and Véronique Cortier. ACM Press, Oct. 2025, pp. 4035–4048. DOI: 10.1145/3719027.3765226.

[ZXH+22] Jiaheng Zhang, Tiancheng Xie, Thang Hoang, Elaine Shi, and Yupeng Zhang. "Polynomial Commitment with a One-to-Many Prover and Applications". In: *USENIX Security 2022*. Ed. by Kevin R. B. Butler and Kurt Thomas. USENIX Association, Aug. 2022, pp. 2965–2982. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/zhang-jiaheng.

Appendix

A Deferred Proofs

A.1 Proof of Thm. 4.4.1

Proof. Sequential Composition. Let $\mathcal{A}_{s,1}, \mathcal{A}_{s,2}$ be PPT adversaries against knowledge soundness of NIRoK_s. We show that there exists an extractor $\mathcal{E}_s$ that outputs a valid witness w_1 when $\mathcal{A}_{s,1}, \mathcal{A}_{s,2}$ output a valid proof and statement/witness pair $\mathsf{x}_3, \mathsf{w}_3$. First, we define a pair of adversaries $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ for NIRoK_2:

$\underline{\mathcal{B}_1^{\rho(\cdot)}(\mathsf{pp})}$:
$\mathsf{x}_1, \mathsf{st} \leftarrow \mathcal{A}_{s,1}^{\rho(\cdot)}(\mathsf{pp})$
$\mathsf{x}_3, \mathsf{w}_3, \pi \leftarrow \mathcal{A}_{s,2}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})$
Parse $\pi := ((\mathsf{x}_2, \pi_1), \pi_2)$
Output $\mathsf{x}_2, (\pi_2, \mathsf{x}_3, \mathsf{w}_3)$

$\underline{\mathcal{B}_2^{\rho(\cdot)}(\mathsf{pp}, (\pi_2, \mathsf{x}_3, \mathsf{w}_3))}$:
Output $\pi_2, \mathsf{x}_3, \mathsf{w}_3$

By construction of NIRoK_s and $\mathcal{B}$:

$$\Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}_3, \mathsf{w}_3) \in \mathcal{R}_3 \wedge \\ \mathsf{NIRoK}_2.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_2, \mathsf{x}_3, \pi_2) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NIRoK}_2.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_2, \mathsf{st} \leftarrow \mathcal{B}_1^{\rho(\cdot)}(\mathsf{pp}) \\ \pi_2, \mathsf{x}_3, \mathsf{w}_3 \leftarrow \mathcal{B}_2^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}_3, \mathsf{w}_3) \in \mathcal{R}_3 \wedge \\ \mathsf{NIRoK}_2.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_2, \mathsf{x}_3, \pi_2) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NIRoK}_2.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_1, \mathsf{st} \leftarrow \mathcal{A}_{s,1}^{\rho(\cdot)}(\mathsf{pp}) \\ \mathsf{x}_3, \mathsf{w}_3, ((\mathsf{x}_2, \pi_1), \pi_2) \leftarrow \mathcal{A}_{s,2}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}_3, \mathsf{w}_3) \in \mathcal{R}_3 \wedge \\ \mathsf{NIRoK}_s.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_3, \pi) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NIRoK}_s.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_1, \mathsf{st} \leftarrow \mathcal{A}_{s,1}^{\rho(\cdot)}(\mathsf{pp}) \\ \mathsf{x}_3, \mathsf{w}_3, \pi \leftarrow \mathcal{A}_{s,2}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right]$$

and by knowledge soundness of NIRoK_2, there exists an extractor $\mathcal{E}_2$ s.t.

$$\Pr\left[(\mathsf{pp}, \mathsf{x}_2, \mathsf{w}_2) \in \mathcal{R}_2 \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NIRoK}_2.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_2, \mathsf{st} \leftarrow \mathcal{B}_1^{\rho(\cdot)}(\mathsf{pp}) \\ \mathsf{w}_2 \leftarrow \mathcal{E}_2^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right] + \mathsf{negl}(\lambda)$$

$$\geq \Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}_3, \mathsf{w}_3) \in \mathcal{R}_3 \wedge \\ \mathsf{NIRoK}_2.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_2, \mathsf{x}_3, \pi_2) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NIRoK}_2.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_2, \mathsf{st} \leftarrow \mathcal{B}_1^{\rho(\cdot)}(\mathsf{pp}) \\ \pi_2, \mathsf{x}_3, \mathsf{w}_3 \leftarrow \mathcal{B}_2^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}_3, \mathsf{w}_3) \in \mathcal{R}_3 \wedge \\ \mathsf{NIRoK}_s.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_3, \pi) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NIRoK}_s.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_1, \mathsf{st} \leftarrow \mathcal{A}_{s,1}^{\rho(\cdot)}(\mathsf{pp}) \\ \mathsf{x}_3, \mathsf{w}_3, \pi \leftarrow \mathcal{A}_{s,2}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right]$$

Next, we define a pair of adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ for NIRoK_1:

$\underline{\mathcal{A}_1^{\rho(\cdot)}(\mathsf{pp})}$:
$\mathsf{x}_1, \mathsf{st} \leftarrow \mathcal{A}_{s,1}^{\rho(\cdot)}(\mathsf{pp})$
Output $\mathsf{x}_1, \mathsf{st}$.

$\underline{\mathcal{A}_2^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})}$:
$\mathsf{x}_3, \mathsf{w}_3, \pi \leftarrow \mathcal{A}_{s,2}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})$
Parse $\pi := ((\mathsf{x}_2, \pi_1), \pi_2)$
$\mathsf{w}_2 \leftarrow \mathcal{E}_2^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})$
Output $\pi_1, \mathsf{x}_2, \mathsf{w}_2$

Arguing as in the previous case we have:

$$\Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}_2, \mathsf{w}_2) \in \mathcal{R}_2 \wedge \\ \mathsf{NIRoK}_1.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_2, \pi_1) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NIRoK}_1.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_1, \mathsf{st} \leftarrow \mathcal{A}_1^{\rho(\cdot)}(\mathsf{pp}) \\ \pi_1, \mathsf{x}_2, \mathsf{w}_2 \leftarrow \mathcal{A}_2^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}_2, \mathsf{w}_2) \in \mathcal{R}_2 \wedge \\ \mathsf{NIRoK}_1.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_2, \pi_1) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NIRoK}_1.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_1, \mathsf{st} \leftarrow \mathcal{A}_{s,1}^{\rho(\cdot)}(\mathsf{pp}) \\ \mathsf{x}_3, \mathsf{w}_3, ((\mathsf{x}_2, \pi_1), \pi_2) \leftarrow \mathcal{A}_{s,2}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st}) \\ \mathsf{w}_2 \leftarrow \mathcal{E}_2^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}_3, \mathsf{w}_3) \in \mathcal{R}_3 \wedge \\ \mathsf{NIRoK}_2.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_2, \mathsf{x}_3, \pi_2) = 1 \wedge \\ \mathsf{NIRoK}_1.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_2, \pi_1) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NIRoK}_1.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_1, \mathsf{st} \leftarrow \mathcal{A}_{s,1}^{\rho(\cdot)}(\mathsf{pp}) \\ \mathsf{x}_3, \mathsf{w}_3, ((\mathsf{x}_2, \pi_1), \pi_2) \leftarrow \mathcal{A}_{s,2}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right] + \mathsf{negl}(\lambda)$$

$$\geq \Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}_3, \mathsf{w}_3) \in \mathcal{R}_3 \wedge \\ \mathsf{NIRoK}_s.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_3, \pi) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NIRoK}_s.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_1, \mathsf{st} \leftarrow \mathcal{A}_{s,1}^{\rho(\cdot)}(\mathsf{pp}) \\ \mathsf{x}_3, \mathsf{w}_3, \pi \leftarrow \mathcal{A}_{s,2}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right] + \mathsf{negl}(\lambda)$$

and by knowledge soundness of NIRoK_1, there exists an extractor $\mathcal{E}_1$ s.t.

$$\Pr\left[(\mathsf{pp}, \mathsf{x}_1, \mathsf{w}_1) \in \mathcal{R}_1 \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NIRoK}_1.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_1, \mathsf{st} \leftarrow \mathcal{A}_1^{\rho(\cdot)}(\mathsf{pp}) \\ \mathsf{w}_1 \leftarrow \mathcal{E}_1^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right] + \mathsf{negl}(\lambda)$$

$$\geq \Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}_2, \mathsf{w}_2) \in \mathcal{R}_2 \wedge \\ \mathsf{NIRoK}_1.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_2, \pi_1) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NIRoK}_1.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_1, \mathsf{st} \leftarrow \mathcal{A}_1^{\rho(\cdot)}(\mathsf{pp}) \\ \mathsf{x}_2, \mathsf{w}_2, \pi_1 \leftarrow \mathcal{A}_2^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right] + \mathsf{negl}(\lambda)$$

$$\geq \Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{x}_3, \mathsf{w}_3) \in \mathcal{R}_3 \wedge \\ \mathsf{NIRoK}_s.\mathsf{Verify}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_3, \pi) = 1\end{array} \middle| \begin{array}{r}\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NIRoK}_s.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{x}_1, \mathsf{st} \leftarrow \mathcal{A}_{s,1}^{\rho(\cdot)}(\mathsf{pp}) \\ \mathsf{x}_3, \mathsf{w}_3, \pi \leftarrow \mathcal{A}_{s,2}^{\rho(\cdot)}(\mathsf{pp}, \mathsf{st})\end{array}\right] + \mathsf{negl}(\lambda)$$

and therefore $\mathcal{E}_s := \mathcal{E}_1$ is a valid knowledge extractor against NIRoK_s.

Parallel Composition. Let $\mathcal{A}_{p,1}, \mathcal{A}_{p,2}$ be PPT adversaries against knowledge soundness of NIRoK_p. We show that there exists an extractor $\mathcal{E}_p$ that outputs valid witnesses $\mathsf{w}_1, \mathsf{w}_3$ when $\mathcal{A}_{p,1}, \mathcal{A}_{p,2}$ output a valid proof and statement/witness pair $(\mathsf{x}_2, \mathsf{x}_4), (\mathsf{w}_2, \mathsf{w}_4)$. We first define adversaries $\mathcal{A}^{(1)} = (\mathcal{A}_1^{(1)}, \mathcal{A}_2^{(1)})$, $\mathcal{A}^{(2)} = (\mathcal{A}_1^{(2)}, \mathcal{A}_2^{(2)})$ for NIRoK_1, NIRoK_2 respectively.

$\underline{\mathcal{A}_1^{(1)\,\boldsymbol{\rho}(\cdot)}(\mathsf{pp})}$:

$(\mathsf{x}_1, \mathsf{x}_3), \mathsf{st} \leftarrow \mathcal{A}_{p,1}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp})$

Output $\mathsf{x}_1, \mathsf{st}$

$\underline{\mathcal{A}_2^{(1)\,\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{st})}$:

$(\mathsf{x}_2, \mathsf{x}_4), (\mathsf{w}_2, \mathsf{w}_4), \pi \leftarrow \mathcal{A}_{p,2}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{st})$

Parse $\pi := (\pi_1, \pi_2)$

Output $\pi_1, \mathsf{x}_2, \mathsf{w}_2$

$\underline{\mathcal{A}_1^{(2)\,\boldsymbol{\rho}(\cdot)}(\mathsf{pp})}$:

$(\mathsf{x}_1, \mathsf{x}_3), \mathsf{st} \leftarrow \mathcal{A}_{p,1}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp})$

Output $\mathsf{x}_3, \mathsf{st}$

$\underline{\mathcal{A}_2^{(2)\,\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{st})}$:

$(\mathsf{x}_2, \mathsf{x}_4), (\mathsf{w}_2, \mathsf{w}_4), \pi \leftarrow \mathcal{A}_{p,2}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{st})$

Parse $\pi := (\pi_1, \pi_2)$

Output $\pi_2, \mathsf{x}_4, \mathsf{w}_4$

By knowledge soundness of NIRoK_1 and NIRoK_2 there exist extractors $\mathcal{E}_1, \mathcal{E}_2$ for $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}$ respectively, and we define a corresponding extractor $\mathcal{E}_p$ that works as follows:

$\underline{\mathcal{E}_p^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{st})}$:

$(\mathsf{x}_2, \mathsf{x}_4), (\mathsf{w}_2, \mathsf{w}_4), \pi \leftarrow \mathcal{A}_{p,2}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{st})$

$\mathsf{w}_1, \mathsf{w}_3 \leftarrow \mathcal{E}_1(\mathsf{pp}, \mathsf{st}), \mathcal{E}_2(\mathsf{pp}, \mathsf{st})$

output $\mathsf{w}_1, \mathsf{w}_3$

We have that

$$\Pr\left[\begin{array}{c|r} (\mathsf{pp}, (\mathsf{x}_2, \mathsf{x}_4), (\mathsf{w}_2, \mathsf{w}_4)) \in \mathcal{R}_2 \times \mathcal{R}_4 \wedge & \rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ \mathsf{NIRoK}_p.\mathsf{Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, (\mathsf{x}_1, \mathsf{x}_3), (\mathsf{x}_2, \mathsf{x}_4), \pi) = 1 & \mathsf{pp} \leftarrow \mathsf{NIRoK}_p.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ & \mathsf{x}_1, \mathsf{x}_3, \mathsf{st} \leftarrow \mathcal{A}_{p,1}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}) \\ & \mathsf{x}_2, \mathsf{x}_4, \mathsf{w}_2, \mathsf{w}_4, \pi_1, \pi_2 \leftarrow \mathcal{A}_{p,2}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{st}) \end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{x}_2, \mathsf{w}_2) \in \mathcal{R}_2 \wedge & \rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ (\mathsf{pp}, \mathsf{x}_4, \mathsf{w}_4) \in \mathcal{R}_4 \wedge & \mathsf{pp} \leftarrow \mathsf{NIRoK}_p.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{NIRoK}_1.\mathsf{Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_2, \pi_1) = 1 \wedge & \mathsf{x}_1, \mathsf{x}_3, \mathsf{st} \leftarrow \mathcal{A}_{p,1}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}) \\ \mathsf{NIRoK}_2.\mathsf{Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{x}_3, \mathsf{x}_4, \pi_2) = 1 & \mathsf{x}_2, \mathsf{x}_4, \mathsf{w}_2, \mathsf{w}_4, \pi_1, \pi_2 \leftarrow \mathcal{A}_{p,2}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{st}) \end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{x}_2, \mathsf{w}_2) \in \mathcal{R}_2 \wedge & \rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ (\mathsf{pp}, \mathsf{x}_4, \mathsf{w}_4) \in \mathcal{R}_4 \wedge & \mathsf{pp} \leftarrow \mathsf{NIRoK}_p.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{NIRoK}_1.\mathsf{Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_2, \pi_1) = 1 \wedge & \mathsf{x}_1, \mathsf{x}_3, \mathsf{st} \leftarrow \mathcal{A}_{p,1}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}) \\ \mathsf{NIRoK}_2.\mathsf{Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{x}_3, \mathsf{x}_4, \pi_2) = 1 & \mathsf{x}_2, \mathsf{w}_2, \pi_1 \leftarrow \mathcal{A}_2^{(1)\,\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{st}) \\ & \mathsf{x}_4, \mathsf{w}_4, \pi_2 \leftarrow \mathcal{A}_2^{(2)\,\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{st}) \end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{x}_1, \mathsf{w}_1) \in \mathcal{R}_1 \wedge & \rho_1, \ldots, \rho_\ell \leftarrow \mathcal{U}(\lambda) \\ (\mathsf{pp}, \mathsf{x}_4, \mathsf{w}_4) \in \mathcal{R}_4 \wedge & \mathsf{pp} \leftarrow \mathsf{NIRoK}_p.\mathsf{Gen}(1^\lambda, \mathsf{pp}') \\ \mathsf{NIRoK}_2.\mathsf{Verify}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{x}_3, \mathsf{x}_4, \pi_2) = 1 & \mathsf{x}_1, \mathsf{x}_3, \mathsf{st} \leftarrow \mathcal{A}_{p,1}^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}) \\ & \mathsf{w}_1 \leftarrow \mathcal{E}_1^{\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{st}) \\ & \mathsf{x}_4, \mathsf{w}_4, \pi_2 \leftarrow \mathcal{A}_2^{(2)\,\boldsymbol{\rho}(\cdot)}(\mathsf{pp}, \mathsf{st}) \end{array}\right] + \mathsf{negl}(\lambda)$$

$$\geq \Pr\left[\begin{array}{c|r} (\mathsf{pp},\mathsf{x}_1,\mathsf{w}_1)\in\mathcal{R}_1 \wedge & \rho_1,\ldots,\rho_\ell \leftarrow \mathcal{U}(\lambda) \\ (\mathsf{pp},\mathsf{x}_3,\mathsf{w}_3)\in\mathcal{R}_3 & \mathsf{pp}\leftarrow \mathsf{NIRoK}_p.\mathsf{Gen}(1^\lambda,\mathsf{pp}') \\ & \mathsf{x}_1,\mathsf{x}_3,\mathsf{st}\leftarrow \mathcal{A}_{p,1}^{\rho(\cdot)}(\mathsf{pp}) \\ & \mathsf{w}_1\leftarrow \mathcal{E}_1^{\rho(\cdot)}(\mathsf{pp},\mathsf{st}) \\ & \mathsf{w}_3\leftarrow \mathcal{E}_2^{\rho(\cdot)}(\mathsf{pp},\mathsf{st}) \end{array}\right] + \mathsf{negl}(\lambda)$$

$$\geq \Pr\left[\begin{array}{c|r} (\mathsf{pp},(\mathsf{x}_1,\mathsf{x}_3),(\mathsf{w}_1,\mathsf{w}_3))\in\mathcal{R}_1\times\mathcal{R}_3 & \rho_1,\ldots,\rho_\ell \leftarrow \mathcal{U}(\lambda) \\ & \mathsf{pp}\leftarrow \mathsf{NIRoK}_p.\mathsf{Gen}(1^\lambda,\mathsf{pp}') \\ & \mathsf{x}_1,\mathsf{x}_3,\mathsf{st}\leftarrow \mathcal{A}_{p,1}^{\rho(\cdot)}(\mathsf{pp}) \\ & \mathsf{w}_1,\mathsf{w}_3\leftarrow \mathcal{E}_p^{\rho(\cdot)}(\mathsf{pp},\mathsf{st}) \end{array}\right] + \mathsf{negl}(\lambda)$$

where we used the fact that $\mathsf{NIRoK}_1, \mathsf{NIRoK}_2$ are knowledge sound reductions of knowledge. □