Quasar: Sublinear Multi-Cast Commitment Mixing in Recursive Accumulation

Tianyu Zheng[1], Shang Gao[1], Sherman S. M. Chow[2], Yu Guo[3], and Bin Xiao[1]

[1] Dept. of Comp., The Hong Kong Polytechnic University, Hong Kong
tian-yu.zheng@connect.polyu.hk, {shang-jason.gao, b.xiao}@polyu.edu.hk
[2] Dept. of Info. Engg., The Chinese University of Hong Kong, Hong Kong
smchow@ie.cuhk.edu.hk
[3] SECBIT Labs, Suzhou, China
yu.guo@secbit.io

Abstract. Sitting at the core of folding-based recursion, multi-instance accumulation still forces the recursive verifier to combine ℓ committed objects, incurring $\Theta(\ell)$ circuit-expensive commitment random linear combinations (CRCs). Here ℓ denotes the number of instances accumulated per step, and a CRC is an in-circuit random linear combination of commitments. Eliminating the linear-in-ℓ CRC overhead is the central challenge, and we address it with Quasar, reducing in-circuit CRCs from $\Theta(\ell)$ to $O(1)$ per step, while the unavoidable cost of reading and processing public inputs remains linear as usual. Realizing constant-commitment accumulation, Quasar's *multi-cast reduction* commits to a *union* polynomial and verifies a random partial evaluation, yielding a reduced instance with $O(1)$ commitments. Moreover, composing multi-cast with a standard 2-to-1 folding reduction yields a multi-instance accumulation scheme whose verifier uses $O(\log \ell)$ field work and $O(1)$ CRCs per step. Across instantiations with suitable polynomial commitments, our instantiation supports linear-time accumulation provers, plausible post-quantum security, and parallelizable proving at each step. Not only does Quasar keep CRCs constant in ℓ, it also yields a multi-instance incrementally verifiable computation with lower recursion overhead.

Keywords: Zero-Knowledge Proofs · Recursive Arguments · Accumulation Schemes.

1 Introduction

Succinct Non-interactive ARguments of Knowledge (SNARKs) enable succinct verification of computations across applications, including zero-knowledge virtual machines [2,46], verifiable databases [58,59], and verifiable inference [45,53]. In a *monolithic* SNARK [50], the prover records the full execution trace and proves correctness end-to-end using a general-purpose proof system [37,9,33]. As computations scale, monolithic proving remains costly despite substantial

prover-side optimizations [6,50,13,35,34]. For example, proving a zkVM execution, *e.g.*, EVM or RISC-V [38,39] expressed as an arithmetic circuit with 2^{26} gates can require around 1 hour of proving time [24] and more than 200 GB of memory [54]. These costs motivate recursive and incremental approaches that avoid materializing the full trace at once, but they introduce a separate per-step verification bottleneck addressed by this work.

IVC and PCD. A standard alternative is to partition a large computation into smaller chunks and represent each chunk as a predicate. The prover produces a proof for each predicate instance and then recursively "squeezes" these proofs into a single one. This avoids holding the full trace at once and enables incremental progress. The sequential setting is *incrementally verifiable computation* (IVC) [55], and its generalization to directed acyclic graphs is *proof-carrying data* (PCD) [26]. Compared to monolithic SNARKs, IVC/PCD reduces peak memory, supports streaming proof production, and can enable prover parallelism in several constructions (discussed below).

Accumulation and folding schemes. Traditional IVC/PCD via recursive composition invokes a SNARK at each step to verify the previous step's proof [10,9,28], which requires embedding a full SNARK verifier into the next-step proving circuit and is typically impractical [28,25]. Accumulation (or folding) schemes instead aggregate per-step proofs into a succinct *accumulator* [14,19,18,12,42], so the recursive circuit verifies only an *accumulation step*. A final decider checks the terminal accumulator in time independent of the number of steps, enabling practical recursion.

1.1 Problem Statements

While IVC/PCD improves memory scalability, the per-step recursive circuit can still dominate cost. In Nova-style frameworks [42,43,41], the per-step circuit typically splits into two components.

- The trace for computing a new predicate input (the *computation overhead*).
- The trace for verifying the accumulation between the new proof and the running accumulator (the *recursion overhead*)[4].

Empirically, for R1CS, the recursion overhead for a single accumulation can consume around 10,000 constraints [44], which is 2–3× larger than the computation overhead [49]. This cost limits the number of recursive steps that can be instantiated in practice.

A common optimization, used in ProtoGalaxy [30] and in PCD solutions [18,50], is to process multiple predicate instances or accumulators per recursive step. We refer to these as *multiple accumulation schemes* of arity ℓ, following prior terminology [41,23]. This reduces the number of recursive steps by a factor of roughly ℓ, but it does not necessarily reduce the dominant cost inside each recursive step.

[4] In practice, the recursion overhead also encompasses auxiliary operations such as hashing of instances.

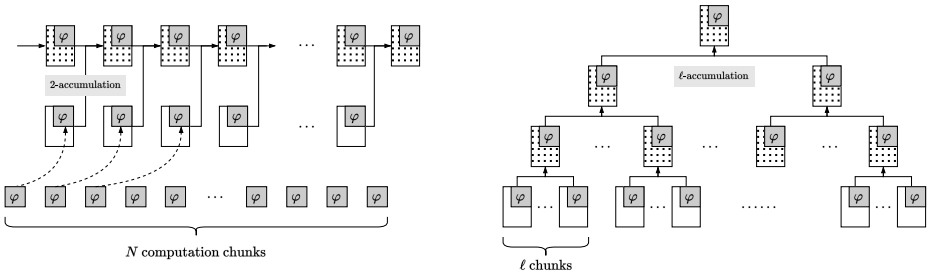


Fig. 1: Workflows of IVC and PCD. The grey block denotes the predicate φ for each computation chunk. In IVC (left), φ is embedded into a single recursive circuit across steps. In PCD (right), φ is embedded into a tree of recursive circuits that can expose parallelism.

The core issue is that recursion overhead is often dominated by *commitment random linear combination* (CRC) operations, which are circuit-inefficient. For curve-based commitments, CRC typically includes *non-native* elliptic-curve operations. For code-based commitments, CRC typically includes Merkle path verification and related hashing. Existing multiple accumulation schemes [19,18,12,42,23] require the verifier to perform $\Theta(\ell)$ CRC operations per step.

Formally, let CRC denote the cost of computing one random linear combination $C \coloneqq r_1 \cdot C_1 + r_2 \cdot C_2$. To prove N computation chunks, an IVC protocol employing a 2-accumulation (left of Figure 1) requires N steps and a total of $N \cdot t$ CRC operations in the recursive circuits, where t is the maximum number of commitments in an accumulator. Thus, though multiple accumulation (right of Figure 1) reduces the number of recursive steps, it does not remove the dominant in-circuit CRC work, since existing ℓ-ary verifiers still incur $\Theta(\ell)$ such operations per step. In recursive settings where CRC is circuit-expensive, this leaves the dominant recursion overhead asymptotically unchanged in ℓ.

These considerations motivate the following question.

Motivation. Can we design a multiple accumulation scheme whose recursive verifier uses a constant number of *in-circuit* CRC operations in the arity ℓ? Equivalently, can we make the dominant in-circuit commitment-combination cost scale as $O(1)$ in ℓ, while treating the linear cost of reading and processing public inputs as orthogonal and unchanged?

1.2 Contributions

Quasar is a multiple accumulation scheme whose recursive verifier uses a constant number of CRC operations in the arity ℓ. Quasar answers the above question by keeping the dominant in-circuit commitment-combination cost constant in ℓ.

New interactive oracle reduction. Building on oracle batching [22], we identify a recursion-friendly reduction interface—multi-cast—that compresses ℓ in-

stances into a constant-commitment derived instance suitable for folding. At a high level, multi-cast reduction transforms ℓ input instances into a single derived instance while ensuring that the output instance-proof pair contains only a constant number of polynomial commitments. Technically, Quasar achieves this via an efficient polynomial partial evaluation check that lets the verifier validate the required commitment relations with $O(1)$ CRC operations, plus $O(\log \ell)$ field work. This targets the dominant circuit cost in recursive settings, while leaving the linear cost of reading and processing public inputs unchanged. The key structural point is that multi-cast outputs a constant-size committed interface that is preserved under folding, rather than merely providing a batched check over ℓ instances.

Accumulation schemes. We propose a framework for constructing accumulation schemes from (i) a *multi-cast reduction* and (ii) a 2-*to*-1 *reduction* for special-sound protocols (SPS) [16]. Main Theorem 1 formalizes the composition and isolates the ℓ-dependence to $O(\log \ell)$ explicit field work, while CRC remains independent of ℓ. For simplicity, we focus on the setting where each recursive step takes ℓ NARK (Non-interactive ARguments of Knowledge) proofs and one running accumulator as inputs. In this setting, each fold level uses $O(1)$ CRC operations and the $O(\log \ell)$ levels contribute $O(\log \ell)$ explicit field work, excluding linear public-input processing. Since our construction is phrased over polynomial oracles, it is compatible with a variety of polynomial commitments, and can achieve *plausible post-quantum security* based on cryptographic hash functions or lattice-based assumptions [22,11].

Instantiations and applications. To demonstrate applicability, we construct a multi-instance IVC protocol that, at each step, takes ℓ predicate instances and a single running accumulator as inputs. We instantiate the underlying polynomial commitment scheme (PCS) using linear-time encodable codes [35,56], and employ a NARK for multivariate Plonkish constraint systems [24], yielding a post-quantum IVC with a linear-time prover. Although this multi-instance IVC does not have the tree structure of PCD [50], it enables parallel execution of the accumulation prover within each step. Applying the same technique to PCD can reduce the recursion costs by a constant factor; however, achieving a PCD with an accumulation verifier that is sublinear in ℓ in the strong sense remains open.

1.3 Performance Analysis

We analyze the recursive verification cost by separating (i) in-circuit commitment combinations and (ii) public-input processing. Throughout, ℓ denotes the accumulation arity. We write CRC for one in-circuit commitment random linear combination, *e.g.*, computing $C \coloneqq r_1 \cdot C_1 + r_2 \cdot C_2$. In curve-based instantiations, CRC induces non-native elliptic-curve arithmetic inside the recursive circuit. In hash-based instantiations, CRC induces hashing work, *e.g.*, Merkle-path verification. Quasar reduces the in-circuit CRC cost from $\Theta(\ell)$ to $O(1)$ per recursive

step, while public-input processing remains linear in ℓ. Equivalently, Quasar moves the ℓ-dependence from in-circuit CRC work to $O(\log \ell)$ random-oracle / sum-check work and PCS evaluation checks.

We summarize asymptotic verifier costs in Table 1, and we instantiate them by compiling accumulation verifiers into recursive circuits under a fixed toolchain with matched parameters. To estimate recursion overhead in IVC, we compile Quasar and representative baselines into recursive circuits in both R1CS and Plonkish using the Nova toolchain [49], and we report recursive circuit size (constraint counts) and related resource footprints across arities in Table 2. The results show that Quasar reduces recursion overhead relative to prior multi-accumulation schemes, including the DL-based HyperNova [41] and the code-based Arc [22], consistent with the cost shift above. Methodology and parameters appear in Appendix C.

Hash versus commitment combinations. Table 1 reports verifier cost in terms of random-oracle queries (denoted by RO) and group operations (denoted by G). For curve-based schemes, we interpret G as the in-circuit elliptic-curve operations induced by CRC. For ProtoGalaxy, the $O(\ell \cdot d)$ group-operation term can be organized as a multi-scalar multiplication (MSM), *e.g.*, via Pippenger, which improves native constant factors but does not remove the ℓ dependence. Inside recursion, that ℓ dependence remains circuit-expensive. Quasar instead keeps the number of in-circuit CRC operations $O(1)$ in ℓ and pays $O(\log \ell)$ RO work from sum-check. Table 2 instantiates this asymptotic distinction with compiled-circuit footprints under a fixed toolchain across arities.

1.4 Scope

Although folding/accumulation schemes enable practical IVC/PCD, key deployment requirements remain orthogonal to the bottleneck studied here. First, zkVM-style workloads (*e.g.*, EVM, RISC-V, Wasm) often yield heterogeneous circuits, whereas many recursive systems are optimized for more uniform constraint structures. Second, many IVC/PCD constructions target a single prover and do not provide per-step zero-knowledge for intermediate witnesses (*e.g.*, Nova [42]), limiting use in privacy-sensitive settings.

Our goal is to introduce a clean primitive (multi-cast reduction) and a modular composition theorem that yields $O(1)$ CRC per recursive step. Accordingly, our evaluation is scoped to validating the circuit footprint implied by the cost model, rather than end-to-end system throughput. In particular, we measure constraint counts (and related resource usage) for the accumulation-verification logic embedded in a recursive circuit under varying arity, and we treat these results as sanity-check datapoints for the predicted ℓ-dependence. We do not report end-to-end multi-instance IVC performance, since a complete prototype would require a recursive wrapper together with an in-circuit verifier for each PCS backend, and these engineering components are orthogonal to the primitive and composition results established here. Concrete wall-clock performance

Table 1: Comparisons between multi-accumulation schemes

Schemes	Constraints	PCS type	PQ	Verifier cost
ProtoGalaxy [30]	Plonkish	Elliptic curves	No	$O(1)\mathsf{Hash}, O(\ell \cdot d)\mathbb{G}$
HyperNova [41]	CCS	Elliptic curves	No	$O(\log n)\mathsf{Hash}, O(\ell)\mathbb{G}$
Quasar (curve)	Plonkish	Elliptic curves	No	$O(\log \ell)\mathsf{Hash}, O(1)\mathbb{G}$
Arc [21]	R1CS	RS codes	Yes	$O(\ell \cdot \frac{\lambda}{\log(1/\rho)} \cdot \log n)\mathsf{Hash}$
WARP [17]	PESAT	Linear codes	Yes	$O(\ell \cdot \frac{\lambda}{\log(1/\rho)} \cdot \log n)\mathsf{Hash}$
Quasar (code)	Plonkish	Linear codes	Yes	$O(\frac{\lambda}{\log(1/\rho)} \cdot (\log n + \log \ell))\mathsf{Hash}$

Hash: hash query, $\mathbb{G}$: group operation, ℓ: number of input predicate instances (arity), d: maximum degree of the relation, n: witness length, ρ: code rate.

Table 2: Concrete recursion overhead for multi-instance IVC with varying arity

	R1CS			Plonkish		
Arity	**HyperNova [41]**	**Quasar**	**Ratio**	**Arc [22]**	**Quasar**	**Ratio**
2	1.48×2^{13}	1.48×2^{13}	1.01×	1.13×2^{17}	1.13×2^{17}	1.01×
4	1.91×2^{13}	1.61×2^{13}	1.09×	1.09×2^{18}	1.14×2^{17}	1.91×
8	1.32×2^{14}	1.75×2^{13}	1.51×	1.62×2^{18}	1.16×2^{17}	2.79×
16	1.98×2^{14}	1.88×2^{13}	2.10×	1.08×2^{19}	1.18×2^{17}	3.66×
32	1.62×2^{15}	1.01×2^{14}	3.21×	1.34×2^{19}	1.20×2^{17}	4.49×

Ratio: dividing the performance of the comparison scheme (HyperNova or Arc) by that of Quasar.

remains backend-dependent, since Quasar trades ℓ-dependent CRC group operations for $O(\log \ell)$ random-oracle / sum-check work and PCS evaluation checks.

2 Technical Overview

We summarize the recursive-accumulation bottleneck and how Quasar removes it. The core idea is to use polynomial partial evaluation so that ℓ instances can be mixed while committing to only $O(1)$ objects. We then present Quasar as the composition of two primitives, a multi-cast reduction and a 2-to-1 reduction, which together yield a multi-instance accumulation scheme. Finally, we outline the resulting multi-instance IVC application.

2.1 New Accumulation Idea

We begin by recalling the accumulation bottleneck. An accumulation scheme incrementally aggregates NP statements into a succinct *accumulator* such that each step is efficiently verifiable. Most practical constructions [14,19,18,12,42,16] rely on homomorphic commitments, which force the verifier to compute random linear combinations of commitments. In recursive settings, these *commitment random linear combination* (CRC) operations are typically circuit-inefficient.

Baseline: linear commitment combination. Consider ℓ instance-witness pairs $\{(\boldsymbol{x}_k, \boldsymbol{w}_k)\}_{k\in[\ell]}$ satisfying a relation $\mathcal{R}$, and the goal of producing one accumulated pair $(\boldsymbol{x}, \boldsymbol{w})$ for a (possibly different) relation $\mathcal{R}'$. A standard reduction commits to each witness and then combines commitments with random coefficients. Abstractly, for a homomorphic commitment scheme $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$, a minimal three-move template is:

- The prover computes $C_k \leftarrow \mathsf{Commit}(\mathsf{ck}, \boldsymbol{w}_k)$ for each $k \in [\ell]$ and sends $\{C_k\}_{k\in[\ell]}$.
- The verifier samples a random challenge $\tau \leftarrow\!\!\$\, \mathbb{F}$.
- The prover outputs $\boldsymbol{w} \coloneqq \sum_{k\in[\ell]} \tau^{k-1}\boldsymbol{w}_k$. The verifier outputs $(\boldsymbol{x}, C)$ where $\boldsymbol{x} \coloneqq \sum_{k\in[\ell]} \tau^{k-1}\boldsymbol{x}_k$ and $C \coloneqq \sum_{k\in[\ell]} \tau^{k-1}C_k$.

Even if each C_k is succinct, the verifier must compute an $O(\ell)$-term linear combination of commitments, *i.e.*, $\Theta(\ell)$ CRC operations. This is precisely the cost that becomes prohibitive when the verifier is embedded into a recursive circuit. The same issue arises for PCS-based designs once commitments to ℓ polynomials must be combined (in-group or via non-native logic) inside the recursive circuit.

First attempt: evaluate, but keep all commitments. A natural idea, inspired by oracle batching [22], is to verify linear combinations via polynomial evaluations. Let $\tilde{w}_k(\boldsymbol{X}) \coloneqq \mathsf{mle}[\boldsymbol{w}_k]$ be the multilinear extension of $\boldsymbol{w}_k$, where $\boldsymbol{X} \in \mathbb{F}^{\log n}$ and $\mathsf{mle}[\cdot]$ is as in Section 3.2. One can commit to each $\tilde{w}_k(\boldsymbol{X})$ and to an accumulated polynomial $\tilde{w}(\boldsymbol{X})$, and then check at a random $\boldsymbol{r} \leftarrow\!\!\$\, \mathbb{F}^{\log n}$ that

$$\tilde{w}(\boldsymbol{r}) \stackrel{?}{=} \sum_{k\in[\ell]} \tau_k \tilde{w}_k(\boldsymbol{r})$$

for appropriate random coefficients $\{\tau_k\}$. This avoids combining commitments in the group, but it does *not* reduce recursion overhead. The accumulated instance still contains $\ell + 1$ commitments (the ℓ inputs plus the output), which must be handled by the next accumulation step, reintroducing $\Theta(\ell)$ CRC operations.

Accumulation via partial evaluations: one union commitment. The key observation is that the prover can commit to *all* $\{\boldsymbol{w}_k\}_{k\in[\ell]}$ using a single polynomial. Let $\boldsymbol{Y} \in \mathbb{F}^{\log \ell}$ index $k \in [\ell]$ via the Boolean hypercube, and define the *union polynomial*

$$\tilde{w}_\cup(\boldsymbol{Y}, \boldsymbol{X}) \coloneqq \mathsf{mle}[\{\boldsymbol{w}_k\}_{k\in[\ell]}]$$

over $(\boldsymbol{Y}, \boldsymbol{X}) \in \mathbb{F}^{\log \ell} \times \mathbb{F}^{\log n}$. By construction, for Boolean $\boldsymbol{y}$ encoding k, we have $\tilde{w}_\cup(\boldsymbol{y}, \boldsymbol{X}) = \tilde{w}_k(\boldsymbol{X})$. For a uniformly random $\boldsymbol{\tau} \leftarrow\!\!\$\, \mathbb{F}^{\log \ell}$, the partial evaluation $\tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{X})$ is a random linear combination of the $\tilde{w}_k(\boldsymbol{X})$ with coefficients given by multilinear equality polynomials over the hypercube. Let $\tilde{w}(\boldsymbol{X})$ denote the (honest) accumulated polynomial, defined as

$$\tilde{w}(\boldsymbol{X}) \coloneqq \tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{X}). \tag{1}$$

If the prover provides a commitment to $\tilde{w}_{\cup}(\boldsymbol{Y}, \boldsymbol{X})$ and a commitment to $\tilde{w}(\boldsymbol{X})$, then it suffices for the verifier to check Equation (1) at a single random point $\boldsymbol{r} \leftarrow\!\!\$\ \mathbb{F}^{\log n}$:

$$\tilde{w}(\boldsymbol{r}) \stackrel{?}{=} \tilde{w}_{\cup}(\boldsymbol{\tau}, \boldsymbol{r}).$$

Although PCS interfaces do not natively provide partial evaluation queries, the verifier can open the two commitments at the points $\boldsymbol{r}$ and $(\boldsymbol{\tau}, \boldsymbol{r})$ and compare the resulting field values. Given binding openings, soundness reduces to a polynomial identity test via Schwartz–Zippel. If $\tilde{w}(\boldsymbol{X}) \neq \tilde{w}_{\cup}(\boldsymbol{\tau}, \boldsymbol{X})$ as polynomials, then equality at a uniformly random $\boldsymbol{r}$ holds with probability at most $\deg(\Delta)/|\mathbb{F}|$, where $\Delta(\boldsymbol{X}) \coloneqq \tilde{w}(\boldsymbol{X}) - \tilde{w}_{\cup}(\boldsymbol{\tau}, \boldsymbol{X})$ and $\deg(\Delta)$ denotes the total degree. Since $\tilde{w}$ is multilinear in $\log n$ variables, we have $\deg(\Delta) \leq \log n$, yielding error at most $(\log n)/|\mathbb{F}|$.

Crucially, the accumulated instance now contains only a *constant* number of commitments (to $\tilde{w}_{\cup}$ and $\tilde{w}$, and similarly for instances as needed), so subsequent recursion does not reintroduce $\Theta(\ell)$ CRC operations. We formalize this idea as a new primitive termed *multi-cast reduction*, and instantiate it in Section 4.

2.2 Main Result: Multi-instance Accumulation with Constant CRC

We adopt the accumulation framework of Bünz *et al.* [22,17], which builds accumulation schemes from interactive oracle reductions (IOR; see Section 3.3). At a high level, the framework uses:

- a cast IOR from an NP relation $\mathcal{R}$ to a relation $\mathcal{R}_{\mathsf{acc}}$, and
- a many-to-one IOR from $(\mathcal{R}_{\mathsf{acc}})^{\ell}$ to $\mathcal{R}_{\mathsf{acc}}$.

The bottleneck in prior multi-instance designs is that the many-to-one step combines ℓ *committed* objects, forcing $\Theta(\ell)$ *in-circuit* CRC operations in the recursive verifier. Quasar shifts this combination from the folding stage to a cast stage that takes *non-committed* inputs, so that the recursive verifier pays only a constant number of CRC operations.

To better accomodate to existing proof systems, we follow Protostar [16] and adopt our accumulation to special-sound protocols (SPS; see Section 5.1). The result is formalized as Main Theorem 1. We measure recursive-verifier cost by separating (i) circuit-expensive CRC and (ii) standard processing of public inputs. A CRC is one in-circuit operation computing a random linear combination of two committed objects, *e.g.*, $C \coloneqq r_1 \cdot C_1 + r_2 \cdot C_2$. All bounds below exclude the unavoidable linear cost of reading and processing the ℓ public instances, which remains linear as usual.

Theorem 1 (Composing multi-cast with 2-to-1 folding). *Let Π_{SPS} be a special-sound protocol for a relation R. Assume we are given: (i) a multi-cast reduction $\mathsf{IOR}_{\mathsf{cast\text{-}SPS}}$ (and its Fiat–Shamir compilation $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}$), and (ii) a 2-to-1 folding reduction $\mathsf{IOR}_{\mathsf{fold\text{-}SPS}}$ (and its Fiat–Shamir compilation $\mathsf{NIR}_{\mathsf{fold\text{-}SPS}}$), both instantiated with an extractable PCS and compiled in the random-oracle model. Then the construction in Section 6 yields: (a) a non-interactive argument*

NARK *for ℓ statements of R, and (b) a multi-instance accumulation scheme* ACC *for* NARK*, satisfying perfect completeness and round-by-round (RBR) knowledge soundness in the random-oracle model. Moreover, each fold level invokes the underlying batching verifier a constant number of times (independent of ℓ), and the $O(\log \ell)$ fold levels contribute only explicit field work (e.g., sum-check / partial evaluation), excluding linear public-input processing. (Each batching invocation combines two committed oracles into one; see Fig. 11.)*

Proof idea. The multi-cast reduction compresses ℓ input instances into a single accumulator instance while keeping the accumulator interface constant-size (a constant number of oracle polynomials). The folding reduction then combines two such accumulator instances into one without increasing that interface, since it only requires a constant number of batching checks. Therefore, the number of commitment-combination checks in the recursive circuit is independent of ℓ, while the only ℓ-dependent work introduced by multi-cast is explicit field work (sum-check / partial-evaluation), and linear public-input processing is orthogonal and excluded from the metric.

Why this is not immediate. It is well-known that sum-check and partial-evaluation tests can certify polynomial identities and batched evaluation claims. In particular, prior multi-instance designs typically incur $\Theta(\ell)$ commitment-combination work per step in the recursive verifier, even when their arithmetic can be organized as an MSM. Our main contribution is not the existence of these checks, but an accumulation *interface* that makes them compatible with recursion at arity ℓ. Concretely, a naïve "batch ℓ instances and sum-check" approach still leaves the recursive verifier facing $\Theta(\ell)$ committed objects (or $\Theta(\ell)$ commitment combinations) somewhere in the pipeline, so the in-circuit commitment-combination cost does not disappear. Quasar's multi-cast reduction is designed to output a *constant-size committed instance* (a constant number of oracle polynomials) whose verification requires only $O(1)$ in-circuit CRC, independent of ℓ. The composition theorem then shows that this constant-size interface is preserved under 2-to-1 folding, so $\Theta(\ell)$ committed objects do not reappear across recursive steps. This interface-preservation property, together with a clean cost separation that excludes linear public-input processing, is the structural novelty that yields constant in-circuit CRC in ℓ. We further illustrate this framework in Figure 2.

Multi-cast reduction (informal interface). A multi-cast reduction $\mathsf{IOR}_{\mathsf{cast}}$ takes ℓ pairs $\{(\boldsymbol{x}_k, \boldsymbol{w}_k)\}_{k\in[\ell]}$ for $\mathcal{R}^\ell$ and outputs one pair $(\boldsymbol{x}, \boldsymbol{w})$ for $\mathcal{R}_{\mathsf{acc}}$. For simplicity, assume $\boldsymbol{x}_k \in \mathbb{F}^m$ and $\boldsymbol{w}_k \in \mathbb{F}^n$ with fixed m, n. At a high level:

- The prover commits to union polynomials $\tilde{x}_\cup(\boldsymbol{Y}, \boldsymbol{X}) := \mathsf{mle}[\{\boldsymbol{x}_k\}_{k\in[\ell]}]$ and $\tilde{w}_\cup(\boldsymbol{Y}, \boldsymbol{X}) := \mathsf{mle}[\{\boldsymbol{w}_k\}_{k\in[\ell]}]$, producing commitments C_x and C_w.
- The verifier samples $\boldsymbol{\tau} \leftarrow\$ \mathbb{F}^{\log \ell}$.
- The prover provides the accumulated public instance $\boldsymbol{x}$ (computed by field-only processing of $\{\boldsymbol{x}_k\}_{k\in[\ell]}$) and commits to the partially evaluated polynomial $\tilde{w}(\boldsymbol{X}) := \tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{X})$, producing a commitment C, together with the accumulated witness $\boldsymbol{w}$.

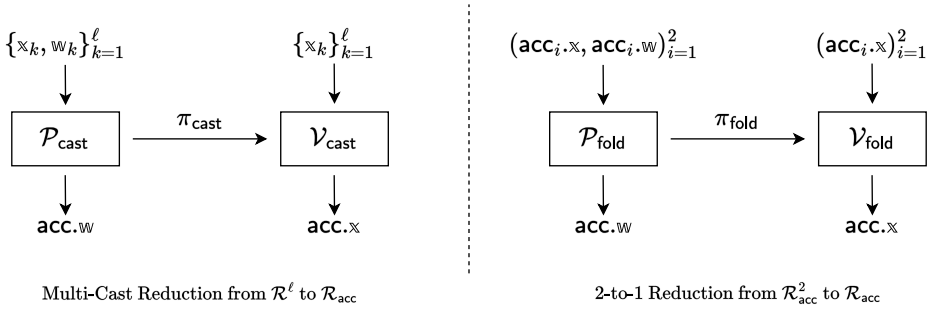


Fig. 2: Quasar shifts the ℓ-way combination away from folding. (The multi-cast reduction $\mathsf{IOR}_{\mathsf{cast}}$ absorbs ℓ non-committed inputs and outputs a constant-size committed instance for $\mathcal{R}_{\mathsf{acc}}$, after which $\mathsf{IOR}_{\mathsf{fold}}$ only performs 2-to-1 folding.)

- The verifier samples $\boldsymbol{r}_x \leftarrow\!\$\ \mathbb{F}^{\log m}$ and $\boldsymbol{r}_w \leftarrow\!\$\ \mathbb{F}^{\log n}$ and defines the reduced instance to include $(C_x, C_w, C, \boldsymbol{\tau}, \boldsymbol{r}_x, \boldsymbol{r}_w)$, together with $\boldsymbol{x}$.

The relation $\mathcal{R}_{\mathsf{acc}}$ enforces that C_x, C_w, C are valid commitments to the claimed polynomials and that

$$\tilde{x}_{\cup}(\boldsymbol{\tau}, \boldsymbol{r}_x) = \tilde{x}(\boldsymbol{r}_x) \quad \text{and} \quad \tilde{w}_{\cup}(\boldsymbol{\tau}, \boldsymbol{r}_w) = \tilde{w}(\boldsymbol{r}_w),$$

via standard PCS openings, where $\tilde{x}(\cdot)$ denotes the multilinear extension of the provided public vector $\boldsymbol{x}$. Computing $\boldsymbol{x}$ from $\{\boldsymbol{x}_k\}_{k\in[\ell]}$ is a field-only operation that is linear in the input length and is not the target of Quasar's constant-CRC improvement.

2-*to*-1 *reduction (informal interface).* The 2-to-1 reduction $\mathsf{IOR}_{\mathsf{fold}}$ takes two instances for $\mathcal{R}_{\mathsf{acc}}$ and outputs one instance for $\mathcal{R}_{\mathsf{acc}}$. In our instantiations, $\mathcal{R}_{\mathsf{acc}}$ is expressed as the conjunction of a constant number of evaluation subrelations of the form

$$\mathcal{R}_{\mathsf{eval}}(\mathsf{pp}) \coloneqq \big\{(C, (\boldsymbol{r}_i, v_i)_i; \tilde{p}(\boldsymbol{X})) \colon C = \mathsf{Commit}(\mathsf{pp}, \tilde{p}(\boldsymbol{X})) \ \wedge\ \tilde{p}(\boldsymbol{r}_i) = v_i\ \forall i\big\}.$$

Folding two $\mathcal{R}_{\mathsf{eval}}$ instances proceeds by (i) reducing two evaluation claims to one via a sum-check-derived random linear combination over the field, and (ii) batching the corresponding commitments via an oracle batching $\mathsf{IOR}_{\mathsf{batch}}$. We require $\mathsf{IOR}_{\mathsf{batch}}$ to be *succinct* (proof size sublinear in the polynomial length), which ensures that the resulting recursive verifier remains efficient. Because $\mathsf{IOR}_{\mathsf{fold}}$ is only 2-to-1 and $\mathcal{R}_{\mathsf{acc}}$ contains a constant number of $\mathcal{R}_{\mathsf{eval}}$, each fold step uses only a constant number of CRC operations, and the dependence on ℓ arises only through the $O(\log \ell)$ number of fold levels.

2.3 Applications to IVC

We next apply Quasar to IVC. We use the notion of *multi-instance IVC* (formalized in Definition 4), where each recursive step processes ℓ predicate instances

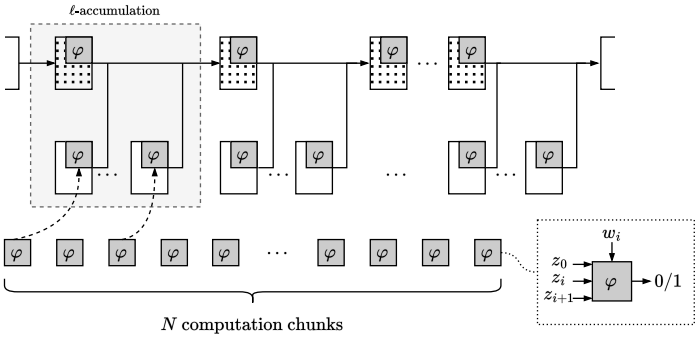


Fig. 3: Overview of multi-instance IVC: φ denotes the predicate for a computation chunk, where $\varphi(z_0, z_i, z_{i+1}, w_i) = 1$ indicates compliance.

together with one running accumulator. This captures settings where the computation remains sequential (as in standard IVC), but batching ℓ instances per step reduces the number of recursive steps and exposes parallelism inside the accumulation prover.

Figure 3 depicts the resulting workflow. The prover updates the running accumulator until ℓ chunk instances have been generated, and then performs a multi-instance accumulation step. This enables parallel execution of the accumulation prover within each step, which is useful when the per-step circuit is dominated by commitment-combination logic. It remains an open problem to achieve a sublinear accumulation scheme for *multiple* accumulators (which is required for fully general PCD).

We emphasize that a multi-instance IVC that accumulates ℓ chunk proofs per step is distinct from an IVC that treats one chunk of size ℓ times larger as a single instance. In the latter, the final statement and proof typically grow proportionally with the enlarged instance, whereas multi-instance IVC keeps the per-chunk interface and accumulates across chunks.

Relation to vector-commitment partial evaluation. The partial-evaluation consistency check above (commit to a union polynomial and verify a random partial evaluation at a random point) also appears outside accumulation, *e.g.*, in HydraProofs [51] for optimally computing all proofs in a vector commitment. Quasar's contribution is not this check in isolation, but using it to build a multi-cast IOR that outputs a constant-commitment reduced instance, preventing $\Theta(\ell)$ committed objects from reappearing in subsequent folding. This structural shift is what yields $O(1)$ CRC operations per recursive step when composing multi-cast with a standard 2-to-1 folding reduction.

3 Preliminaries

3.1 Notations

We use λ to denote the security parameter. A function $\mathsf{negl}(\lambda)$ is negligible in λ. Let $\mathbb{F}$ denote a finite field (typically a prime field $\mathbb{F}_p$ for a large prime p).

Vectors are written in bold, *e.g.*, $\boldsymbol{a} \in \mathbb{F}^n$ with entries $a_1, \ldots, a_n \in \mathbb{F}$. We write a_i or $\boldsymbol{a}[i]$ for the i-th entry. For $n \in \mathbb{N}$, let $[n] \coloneqq \{1, \ldots, n\}$. We write $\{a_i\}_{i\in[n]}$ as shorthand for $\{a_1, \ldots, a_n\}$. A probabilistic algorithm is *probabilistic polynomial time* (PPT) if its running time is bounded by a polynomial in the input length. For a finite set S, we write $x \leftarrow\!\!\$\ S$ for sampling x uniformly at random from S.

Relations and oracles. An indexed NP relation $\mathcal{R}$ parameterized by public parameters pp (*e.g.*, the field $\mathbb{F}$) consists of tuples $(\mathtt{i}, \mathtt{x}, \mathtt{w})$, where $\mathtt{i}$ is an index, $\mathtt{x}$ is a public instance, and $\mathtt{w}$ is a witness. We call $\mathcal{R}$ an *oracle relation* if tuples additionally include an oracle string $\mathtt{y}$, *i.e.*, $(\mathtt{i}, \mathtt{x}, \mathtt{y}, \mathtt{w})$, where $\mathtt{y}$ represents one or more oracle strings to which the verifier has query access.

For an indexed oracle relation

$$\mathcal{R}(\mathsf{pp}) \subseteq \{(\mathtt{i}, \mathtt{x}, \mathtt{y}, \mathtt{w})\},$$

let

$$\mathcal{L}(\mathcal{R}) \coloneqq \{(\mathtt{i}, \mathtt{x}, \mathtt{y}) \colon \exists \mathtt{w} \ \textit{s.t.}\ (\mathtt{i}, \mathtt{x}, \mathtt{y}, \mathtt{w}) \in \mathcal{R}\}$$

denote the induced language (we sometimes omit pp when clear from context). For $\ell \in \mathbb{N}$, we write $\mathcal{R}^\ell$ for the product relation consisting of ℓ independent instances sharing the same index:

$$\mathcal{R}^\ell \coloneqq \Big\{ \big(\mathtt{i}, (\mathtt{x}_1, \ldots, \mathtt{x}_\ell), (\mathtt{y}_1, \ldots, \mathtt{y}_\ell), (\mathtt{w}_1, \ldots, \mathtt{w}_\ell)\big) \colon \forall k \in [\ell],\ (\mathtt{i}, \mathtt{x}_k, \mathtt{y}_k, \mathtt{w}_k) \in \mathcal{R} \Big\}.$$

Given $(\mathtt{i}, \mathtt{x})$, let

$$\mathbb{O}(\mathtt{i}, \mathtt{x}) \coloneqq \{\mathtt{y} \colon (\mathtt{i}, \mathtt{x}, \mathtt{y}) \in \mathcal{L}(\mathcal{R})\}$$

denote the set of oracle strings consistent with $(\mathtt{i}, \mathtt{x})$.

To cover code-based polynomial oracles, we define the distance of a statement $(\mathtt{i}, \mathtt{x}, \mathtt{y})$ from $\mathcal{R}$ as

$$\Delta_{\mathcal{R}}(\mathtt{i}, \mathtt{x}, \mathtt{y}) \coloneqq \min_{c \in \mathbb{O}(\mathtt{i}, \mathtt{x})} \Delta(\mathtt{y}, c),$$

where $\Delta(\cdot, \cdot)$ is the relative Hamming distance between equal-length strings. We defer further discussion of the code-based setting to Appendix D.1.

3.2 Multilinear Extensions and Sum-check Protocols

Multilinear extensions. Let $\tilde{f} \colon \mathbb{F}^m \to \mathbb{F}$ be an m-variate polynomial. Its total degree $\deg(\tilde{f}) \in \mathbb{N}$ is the maximum total degree over all monomials. A polynomial is *multilinear* if every variable has degree at most one.

Let $\boldsymbol{f} \in \mathbb{F}^n$ be a vector. For simplicity, we assume n is a power of two, *i.e.*, $n = 2^m$ for $m \coloneqq \log n$ (otherwise, pad with zeros). Let $\mathsf{Bits}(i) \in \{0,1\}^m$ denote the m-bit binary representation of an index $i \in \{0,\dots,n-1\}$. The *multilinear extension* (MLE) of $\boldsymbol{f}$ is the unique multilinear polynomial $\tilde{f}\colon \mathbb{F}^m \to \mathbb{F}$ such that

$$\tilde{f}(\mathsf{Bits}(i)) = \boldsymbol{f}[i+1] \quad \text{for all } i \in \{0,\dots,n-1\}.$$

We write $\tilde{f} \coloneqq \mathsf{mle}[\boldsymbol{f}]$. Equivalently,

$$\tilde{f}(\boldsymbol{X}) = \sum_{i=0}^{n-1} \boldsymbol{f}[i+1] \cdot \tilde{eq}(\boldsymbol{X}, \mathsf{Bits}(i)),$$

where $\tilde{eq}(\boldsymbol{X}, \boldsymbol{b})$ is the multilinear extension of the equality predicate on the Boolean hypercube. We also write

$$\tilde{eq}_i(\boldsymbol{X}) \coloneqq \tilde{eq}(\boldsymbol{X}, \mathsf{Bits}(i)) = \prod_{j=1}^{m} \big(\mathsf{Bits}(i)[j] \cdot X_j + (1 - \mathsf{Bits}(i)[j]) \cdot (1 - X_j)\big).$$

For $d, n \in \mathbb{N}$, we write $\mathcal{F}_n^{(<d)}$ as shorthand for the set of n-variate polynomials over $\mathbb{F}$ with individual degree $< d$, *i.e.*,

$$\mathcal{F}_n^{(<d)} \coloneqq \mathbb{F}^{(<d)}[\boldsymbol{X}] = \{f \in \mathbb{F}[X_1,\dots,X_n] \colon \deg_{X_j}(f) < d \text{ for all } j \in [n]\}.$$

Sum-check protocol. The sum-check protocol is an interactive proof of Lund *et al.* [47]. Let $\tilde{f}(\boldsymbol{X})$ be an m-variate polynomial with individual degree $< d$. The prover aims to convince the verifier of the claim

$$\mathsf{sum} = \sum_{\boldsymbol{x} \in \{0,1\}^m} \tilde{f}(\boldsymbol{x}).$$

The protocol has m rounds. In round $i \in [m]$, given challenges $r_1,\dots,r_{i-1}$ from previous rounds, the prover sends a univariate polynomial

$$f_i(X) \coloneqq \sum_{x_{i+1},\dots,x_m \in \{0,1\}^{m-i}} \tilde{f}(r_1,\dots,r_{i-1}, X, x_{i+1},\dots,x_m).$$

The verifier checks $\mathsf{sum} = f_1(0) + f_1(1)$ when $i = 1$, and checks $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$ otherwise. It then samples $r_i \leftarrow\!\$\, \mathbb{F}$ and continues. At the end, the verifier checks $f_m(r_m) = \tilde{f}(r_1,\dots,r_m)$. We refer to security definitions of the sum-check protocol from [24] and detail them in Appendix D.2.

Lemma 1 (DeMillo–Lipton–Schwartz–Zippel lemma). *Let $\tilde{f} \in \mathcal{F}_n^{(<d)}$ be a non-zero polynomial. Then*

$$\Pr_{\boldsymbol{r} \leftarrow\!\$\, \mathbb{F}^n} \big[\tilde{f}(\boldsymbol{r}) = 0\big] \le \frac{n \cdot (d-1)}{|\mathbb{F}|}.$$

3.3 Interactive Oracle Reductions

Interactive oracle reductions (IOR) [7,22] are information-theoretic proof systems that generalize interactive oracle proofs to the language of reductions. An IOR verifier may output claims about oracle strings without reading them in full.

We consider public-coin polynomial-time IORs from an oracle relation $\mathcal{R}$ to an oracle relation $\mathcal{R}'$ with μ rounds of interaction. In round $i \in [\mu]$, the prover sends either a standard message or an oracle string, the verifier responds with a uniformly random challenge, and the verifier may query previously received oracles.

Definition 1 (Interactive oracle reductions [22]). *A public-coin interactive oracle reduction from oracle relation $\mathcal{R}$ to $\mathcal{R}'$ is a tuple of polynomial-time algorithms $\mathsf{IOR} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ with the following syntax.*

- *The indexer $\mathcal{I}$ is deterministic. On input an index $\mathtt{i}$ for $\mathcal{R}$, it outputs a short index ι, an index oracle $\mathbb{I}$, and a new index $\mathtt{i}'$ for $\mathcal{R}'$.*
- *The prover $\mathcal{P}$ is interactive. On input $(\mathtt{i}, \mathtt{x}, \mathtt{y}, \mathtt{w})$, it engages in μ rounds. In round i, it sends either a non-oracle message m_i or an oracle π_i, and then receives a challenge r_i.*
- *The verifier $\mathcal{V}$ is interactive. On input $(\iota, \mathtt{x})$ and query access to $\mathtt{y}$ and $\mathbb{I}$, it engages in μ rounds. In round i, it receives either m_i or query access to π_i, samples a uniformly random challenge r_i from a specified challenge space (typically $\mathbb{F}$ or $\{0,1\}^{\rho_i}$ for some $\rho_i \in \mathbb{N}$), and sends r_i. After the interaction, it queries $\mathbb{I}$, $\mathtt{y}$, and $\boldsymbol{\pi}$, and then either rejects or outputs a new statement $(\mathtt{x}', \mathtt{y}')$.*
- *The prover outputs a witness $\mathtt{w}'$ such that $(\mathtt{i}', \mathtt{x}', \mathtt{y}', \mathtt{w}') \in \mathcal{R}'$.*

We assume perfect completeness and soundness with proximity radius $\delta^* \in [0,1]$, as defined in Definitions 18 and 19 in Appendix D.5. To justify Fiat–Shamir in the ROM (in particular, to rule out state-restoration attacks), we use the stronger notion of round-by-round (RBR) knowledge soundness [8]. We refer to Definition 21 in Appendix D.5 for formal definitions.

4 New Technique: Multi-Cast Reduction

This section presents the *multi-cast reduction*, the core device behind Quasar's constant-CRC recursion overhead. As discussed in Section 2.1, existing multiple accumulation schemes [16,41,22] require the recursive verifier to compute a random linear combination over the commitments of all ℓ input instances, *i.e.*, $\Theta(\ell)$ CRC operations. In recursive circuits, CRC is often the dominant cost for curve-based and Merkle-hash-based commitments, whereas in lattice/module settings the dominant cost may shift. Quasar avoids $\Theta(\ell)$ in-circuit commitment combinations by moving the ℓ-way combination into field-only polynomial relations and outputting only a constant number of committed objects. Concretely, our multi-cast reduction reduces the number of committed objects that must

be combined from $\Theta(\ell)$ to $O(1)$, while the reduction verifier's *non-linear* work is $O(\log \ell)$ field operations (excluding the linear-time public-input processing discussed in Section 2.1).

We formalize the technique as a *multi-cast interactive oracle reduction* in Section 4.1. We then prove round-by-round (RBR) knowledge soundness in Section 4.2.

4.1 Multi-cast Reductions from Polynomial Evaluations

We present our multi-cast reduction in the IOR model [22]. At this level, we work with polynomial *oracles* (truth tables), and we defer compilation to polynomial commitments to Section 6.1. A key point is that an IOR verifier may *output* evaluation claims about oracle strings without querying them. Accordingly, the reduction below outputs a constant-size oracle statement whose verification is delegated to later folding steps.

A generic high-degree relation. Let $F\colon \mathbb{F}^m \times \mathbb{F}^n \to \mathbb{F}$ be a polynomial map of total degree at most d. We write its homogeneous decomposition as

$$F(\boldsymbol{x}, \boldsymbol{w}) \coloneqq \sum_{j=0}^{d} f_j^F(\boldsymbol{x}, \boldsymbol{w}),$$

where f_j^F is homogeneous of degree j. We use the indexed relation $\mathcal{R}_F$ to represent an *evaluation constraint*:

$$\mathcal{R}_F(\mathsf{pp}) \coloneqq \{(\mathtt{i}, \mathbb{x} = (\boldsymbol{x}, e), \mathbb{w} = \boldsymbol{w})\colon F(\boldsymbol{x}, \boldsymbol{w}) = e\}.$$

(The NP satisfaction constraint corresponds to the special case $e = 0$.)

Goal. Given ℓ tuples $\{(\mathtt{i}, \mathbb{x}^{(k)}, \mathbb{w}^{(k)})\}_{k\in[\ell]}$ for the multi-instance relation $\mathcal{R}_F^\ell$, we construct an IOR $\mathsf{IOR}_{\mathsf{cast}}\colon \mathcal{R}_F^\ell \to \mathcal{R}_{\mathsf{acc}}$ that outputs a reduced statement containing only a constant number of polynomial oracles. The protocol is given in Figure 4. The prover receives $\mathtt{i}$, instances $\{\mathbb{x}^{(k)}\}_{k\in[\ell]}$, and witnesses $\{\mathbb{w}^{(k)}\}_{k\in[\ell]}$. The verifier receives $(\iota, \mathbb{I}, \mathtt{i}') \leftarrow \mathcal{I}(\mathtt{i})$ with $\mathtt{i}' = \mathtt{i}$, and instances $\{\mathbb{x}^{(k)}\}_{k\in[\ell]}$.

Interaction phase. We assume ℓ, m, n are powers of two for simplicity. For each $k \in [\ell]$, write $\boldsymbol{x}^{(k)} \in \mathbb{F}^m$ and $\boldsymbol{w}^{(k)} \in \mathbb{F}^n$. Let $\tilde{x}^{(k)}\colon \mathbb{F}^{\log m} \to \mathbb{F}$ and $\tilde{w}^{(k)}\colon \mathbb{F}^{\log n} \to \mathbb{F}$ denote their multilinear extensions (see Section 3.2), *i.e.*,

$$\tilde{x}^{(k)}(\mathsf{Bits}(j)) = \boldsymbol{x}^{(k)}[j+1] \text{ for } j \in \{0, \ldots, m-1\},$$
$$\tilde{w}^{(k)}(\mathsf{Bits}(j)) = \boldsymbol{w}^{(k)}[j+1] \text{ for } j \in \{0, \ldots, n-1\}.$$

The interaction has two objectives:

- produce an accumulated pair $(\boldsymbol{x}, \boldsymbol{w})$ from $\{(\boldsymbol{x}^{(k)}, \boldsymbol{w}^{(k)})\}_{k\in[\ell]}$ using only a constant number of committed objects, and
- derive an evaluation value $e = F(\boldsymbol{x}, \boldsymbol{w})$ via a sum-check-based reduction, without placing an ℓ-term commitment combination in the recursive verifier.

$\mathcal{P}$'s inputs: $\mathbb{i}, \{\mathbb{x}^{(i)}\}_{i\in[\ell]}; \{\mathbb{w}^{(i)}\}_{i\in[\ell]}$

$\mathcal{V}$'s inputs: $\iota, \mathbb{I}, \{\mathbb{x}^{(i)}\}_{i\in[\ell]}$

Interaction phase.

1: $\mathcal{P}$ computes:

$\tilde{x}_{\cup}(\boldsymbol{Y}, \boldsymbol{X}) \coloneqq \sum_{i\in[\ell]} \tilde{eq}_{i-1}(\boldsymbol{Y}) \cdot \tilde{x}^{(i)}(\boldsymbol{X})$,

$\tilde{w}_{\cup}(\boldsymbol{Y}, \boldsymbol{X}) \coloneqq \sum_{i\in[\ell]} \tilde{eq}_{i-1}(\boldsymbol{Y}) \cdot \tilde{w}^{(i)}(\boldsymbol{X})$.

2: $\mathcal{P} \to \mathcal{V}$: oracles $[\![\tilde{x}_{\cup}]\!], [\![\tilde{w}_{\cup}]\!]$.

3: $\mathcal{V} \to \mathcal{P}$: $\boldsymbol{r}_y \leftarrow\!\$\, \mathbb{F}^{\log \ell}$.

4: $\mathcal{P}$ computes $\tilde{\boldsymbol{x}}(\boldsymbol{Y}), \tilde{\boldsymbol{w}}(\boldsymbol{Y})$ as:

$\tilde{\boldsymbol{x}}(\boldsymbol{Y}) \coloneqq \sum_{i\in[\ell]} \tilde{eq}_{i-1}(\boldsymbol{Y}) \cdot \boldsymbol{x}^{(i)} \in (\mathcal{F}^{(<2)}_{\log \ell})^m$,

$\tilde{\boldsymbol{w}}(\boldsymbol{Y}) \coloneqq \sum_{i\in[\ell]} \tilde{eq}_{i-1}(\boldsymbol{Y}) \cdot \boldsymbol{w}^{(i)} \in (\mathcal{F}^{(<2)}_{\log \ell})^n$.

5: $\mathcal{P}$ computes $G(\boldsymbol{Y}) \coloneqq F(\tilde{\boldsymbol{x}}(\boldsymbol{Y}), \tilde{\boldsymbol{w}}(\boldsymbol{Y})) \cdot \tilde{eq}(\boldsymbol{Y}, \boldsymbol{r}_y)$.

6: $\mathcal{P}$ and $\mathcal{V}$ engage in a $(\log \ell)$-round sum-check for $\sum_{\boldsymbol{y}\in B_{\log \ell}} G(\boldsymbol{y}) = 0$, where the prover sends $\log \ell$ sum-check polynomials and the verifier replies with $\boldsymbol{\tau} \in \mathbb{F}^{\log \ell}$.

7: $\mathcal{V}$ checks:

Reject if $G_1(0) + G_1(1) \neq 0$.

Reject if $G_{i+1}(0) + G_{i+1}(1) \neq G_i(\tau_i)$ for any $i \in [\log \ell - 1]$.

8: $\mathcal{P}$ computes: $\tilde{x}(\boldsymbol{X}) \coloneqq \tilde{x}_{\cup}(\boldsymbol{\tau}, \boldsymbol{X})$, $\tilde{w}(\boldsymbol{X}) \coloneqq \tilde{w}_{\cup}(\boldsymbol{\tau}, \boldsymbol{X})$.

9: $\mathcal{P} \to \mathcal{V}$: a message $\tilde{x}$ and an oracle $[\![\tilde{w}]\!]$.

10: $\mathcal{V}$ samples: $\boldsymbol{r}_x \leftarrow\!\$\, \mathbb{F}^{\log m}$, $\boldsymbol{r}_w \leftarrow\!\$\, \mathbb{F}^{\log n}$.

Output phase.

11: Define $\boldsymbol{x}$ as the coefficient vector of $\tilde{x}$.

12: Define $e \coloneqq G_{\log \ell}(\tau_{\log \ell}) \cdot \tilde{eq}^{-1}(\boldsymbol{\tau}, \boldsymbol{r}_y)$.

13: $\mathcal{V}$ outputs:

$\mathbb{x} \coloneqq (\boldsymbol{x}, \boldsymbol{\tau}, \boldsymbol{r}_x, \boldsymbol{r}_w, e)$,

$\mathbb{y} \coloneqq ([\![\tilde{x}_{\cup}]\!], [\![\tilde{w}_{\cup}]\!], [\![\tilde{w}]\!])$.

Fig. 4: Interactive Oracle Reduction $\mathsf{IOR}_{\mathsf{cast}}$ from $\mathcal{R}^{\ell}_{F}$ to $\mathcal{R}_{\mathsf{acc}}$

Task (1): one union oracle and partial evaluation claims. Let $\boldsymbol{Y} \in \mathbb{F}^{\log \ell}$ and $\boldsymbol{X}$ be the evaluation variables for the MLEs. The prover forms the union polynomials

$$\tilde{x}_{\cup}(\boldsymbol{Y}, \boldsymbol{X}) \coloneqq \mathsf{mle}[\{\boldsymbol{x}^{(k)}\}_{k\in[\ell]}], \qquad \tilde{w}_{\cup}(\boldsymbol{Y}, \boldsymbol{X}) \coloneqq \mathsf{mle}[\{\boldsymbol{w}^{(k)}\}_{k\in[\ell]}].$$

By construction, for Boolean $\boldsymbol{y}$ encoding index $i - 1$, we have

$$\tilde{x}_{\cup}(\boldsymbol{y}, \boldsymbol{X}) = \tilde{x}^{(k)}(\boldsymbol{X}), \qquad \tilde{w}_{\cup}(\boldsymbol{y}, \boldsymbol{X}) = \tilde{w}^{(k)}(\boldsymbol{X}).$$

The prover sends oracles $[\![\tilde{x}_{\cup}]\!]$ and $[\![\tilde{w}_{\cup}]\!]$ at Step 2 in Figure 4.

Let $\boldsymbol{\tau} \leftarrow\!\!\$\ \mathbb{F}^{\log \ell}$ denote the (vector) challenge used to define the accumulation point. Define the accumulated polynomials by partial evaluation at $\boldsymbol{Y} = \boldsymbol{\tau}$:

$$\tilde{x}(\boldsymbol{X}) \coloneqq \tilde{x}_{\cup}(\boldsymbol{\tau}, \boldsymbol{X}), \qquad \tilde{w}(\boldsymbol{X}) \coloneqq \tilde{w}_{\cup}(\boldsymbol{\tau}, \boldsymbol{X}). \tag{2}$$

Polynomial oracle interfaces do not provide a primitive that returns the partially evaluated polynomial $\tilde{x}_{\cup}(\boldsymbol{\tau}, \cdot)$ as a new oracle. Instead, the prover supplies $\tilde{x}$ as an explicit message (equivalently, the coefficient vector of the MLE of $\boldsymbol{x}$), and supplies $[\![\tilde{w}]\!]$ as an oracle. The reduced statement will include *evaluation claims* tying $\tilde{x}$ and $[\![\tilde{w}]\!]$ to the union oracles at the single points $(\boldsymbol{\tau}, \boldsymbol{r}_x)$ and $(\boldsymbol{\tau}, \boldsymbol{r}_w)$, where $\boldsymbol{r}_x \leftarrow\!\!\$\ \mathbb{F}^{\log m}$ and $\boldsymbol{r}_w \leftarrow\!\!\$\ \mathbb{F}^{\log n}$. These evaluation claims are later checked by PCS openings in the compiled scheme.

Concretely, the accumulated vectors are defined as

$$\boldsymbol{x} \coloneqq \sum_{k \in [\ell]} \tilde{eq}_{i-1}(\boldsymbol{\tau}) \cdot \boldsymbol{x}^{(k)}, \qquad \boldsymbol{w} \coloneqq \sum_{k \in [\ell]} \tilde{eq}_{i-1}(\boldsymbol{\tau}) \cdot \boldsymbol{w}^{(k)},$$

where $\tilde{eq}_{i-1}$ is the MLE of the equality predicate on $\{0,1\}^{\log \ell}$ (see Section 3.2).

Task (2): enforcing $e = F(\boldsymbol{x}, \boldsymbol{w})$ via sum-check. Define the selector-based vector polynomials

$$\tilde{\boldsymbol{x}}(\boldsymbol{Y}) \coloneqq \sum_{k \in [\ell]} \tilde{eq}_{i-1}(\boldsymbol{Y}) \cdot \boldsymbol{x}^{(k)}, \qquad \tilde{\boldsymbol{w}}(\boldsymbol{Y}) \coloneqq \sum_{k \in [\ell]} \tilde{eq}_{i-1}(\boldsymbol{Y}) \cdot \boldsymbol{w}^{(k)}.$$

For Boolean $\boldsymbol{y} \in \{0,1\}^{\log \ell}$, the vector $(\tilde{\boldsymbol{x}}(\boldsymbol{y}), \tilde{\boldsymbol{w}}(\boldsymbol{y}))$ equals one of the input pairs $(\boldsymbol{x}^{(k)}, \boldsymbol{w}^{(k)})$.

We use the following pointwise facts about equality polynomials on the Boolean hypercube.

Corollary 1 (Adopted from [30]). *For any $i \in \{0, \ldots, \ell - 1\}$ and any $\boldsymbol{b} \in \{0,1\}^{\log \ell}$, the equality polynomial satisfies:*

- $\big(\tilde{eq}_i(\boldsymbol{b})\big)^d = \tilde{eq}_i(\boldsymbol{b})$ *for all $d \in \mathbb{N}_{\geq 1}$;*
- $\tilde{eq}_i(\boldsymbol{b}) \cdot \tilde{eq}_j(\boldsymbol{b}) = 0$ *for all $j \neq i$.*

Proof. Fix $\boldsymbol{b} \in \{0,1\}^{\log \ell}$. By definition, $\tilde{eq}_i(\boldsymbol{b}) \in \{0,1\}$ and exactly one index i has $\tilde{eq}_i(\boldsymbol{b}) = 1$. Both claims follow immediately. □

Let $H(\boldsymbol{Y}) \coloneqq F(\tilde{\boldsymbol{x}}(\boldsymbol{Y}), \tilde{\boldsymbol{w}}(\boldsymbol{Y}))$. By Corollary 1 and the assumption that each input tuple satisfies $\mathcal{R}_F$, we have $H(\boldsymbol{y}) = 0$ for all $\boldsymbol{y} \in \{0,1\}^{\log \ell}$. To reduce this to a single evaluation claim, the verifier samples $\boldsymbol{r}_y \leftarrow\!\!\$\ \mathbb{F}^{\log \ell}$ and defines

$$G(\boldsymbol{Y}) \coloneqq H(\boldsymbol{Y}) \cdot \tilde{eq}(\boldsymbol{Y}, \boldsymbol{r}_y),$$

which has individual degree at most $d + 1$. The parties then run a $\log \ell$-round sum-check to establish

$$\sum_{\boldsymbol{y} \in \{0,1\}^{\log \ell}} G(\boldsymbol{y}) = 0.$$

At the end of sum-check, the verifier obtains a claim of the form

$$G(\boldsymbol{\tau}) = H(\boldsymbol{\tau}) \cdot \tilde{eq}(\boldsymbol{\tau}, \boldsymbol{r}_y),$$

where $\boldsymbol{\tau} \in \mathbb{F}^{\log \ell}$ is the vector of sum-check challenges. If $\tilde{eq}(\boldsymbol{\tau}, \boldsymbol{r}_y) = 0$, the verifier rejects (or, equivalently, re-samples $\boldsymbol{r}_y$); this event occurs with probability at most $(\log \ell)/|\mathbb{F}|$. Otherwise, the verifier defines

$$e \coloneqq H(\boldsymbol{\tau}) = G(\boldsymbol{\tau}) \cdot \tilde{eq}(\boldsymbol{\tau}, \boldsymbol{r}_y)^{-1},$$

which yields the desired constraint

$$F(\boldsymbol{x}, \boldsymbol{w}) = F(\tilde{\boldsymbol{x}}(\boldsymbol{\tau}), \tilde{\boldsymbol{w}}(\boldsymbol{\tau})) = e.$$

Since G is determined by $\tilde{x}$, $[\![\tilde{w}_\cup]\!]$, and $\boldsymbol{r}_y$, the reduced oracle string need not include $[\![G]\!]$

Output phase. The verifier derives $\boldsymbol{x}$ and e at Steps 11–12 and outputs a reduced statement whose relation is

$$\mathcal{R}_{\mathsf{acc}}(\mathsf{pp}) \coloneqq \left\{ \begin{array}{c} \mathbb{i}, \mathbb{x} = (\boldsymbol{x}, \boldsymbol{\tau}, \boldsymbol{r}_x, \boldsymbol{r}_w, e), \\ \mathbb{y} = ([\![\tilde{x}_\cup]\!], [\![\tilde{w}_\cup]\!], [\![\tilde{w}]\!]), \mathbb{w} = \boldsymbol{w} \end{array} : \begin{array}{c} F(\boldsymbol{x}, \boldsymbol{w}) = e \\ \wedge \tilde{x}_\cup(\boldsymbol{\tau}, \boldsymbol{r}_x) = \tilde{x}(\boldsymbol{r}_x) \\ \wedge \tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{r}_w) = \tilde{w}(\boldsymbol{r}_w) \end{array} \right\}.$$

For later folding, it is convenient to isolate evaluation subrelations. Let $v_x \coloneqq \tilde{x}(\boldsymbol{r}_x)$ and $v_w \coloneqq \tilde{w}(\boldsymbol{r}_w)$. We view $\mathcal{R}_{\mathsf{acc}}$ as the conjunction $\mathcal{R}_{\mathsf{eval}}^3 \times \mathcal{R}_F$ of the following statements:

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{\tau}, \boldsymbol{r}_x, v_x), \mathbb{y} = [\![\tilde{x}_\cup]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}}, \tag{3}$$

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{\tau}, \boldsymbol{r}_w, v_w), \mathbb{y} = [\![\tilde{w}_\cup]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}}, \tag{4}$$

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{r}_w, v_w), \mathbb{y} = [\![\tilde{w}]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}}, \tag{5}$$

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{x}, e), \mathbb{y} = \bot, \mathbb{w} = \boldsymbol{w}) \in \mathcal{R}_F. \tag{6}$$

Here, $\mathcal{R}_{\mathsf{eval}}$ is the oracle relation for a list of evaluation claims:

$$\mathcal{R}_{\mathsf{eval}}(\mathsf{pp}) \coloneqq \{(\mathbb{i}, \mathbb{x} = \{(\boldsymbol{r}_j, v_j)\}_j, \mathbb{y} = [\![\tilde{f}]\!], \mathbb{w} = \bot) \colon \tilde{f}(\boldsymbol{r}_j) = v_j \ \forall j\}.$$

Lemma 2. *The protocol* $\mathsf{IOR}_{\mathsf{cast}}$ *is an interactive oracle reduction from the multi-instance relation* $\mathcal{R}_F^\ell$ *to the indexed oracle relation* $\mathcal{R}_{\mathsf{acc}}$*. It satisfies perfect completeness and RBR knowledge soundness. The complexity is measured as follows:*

- *The protocol has* $\log \ell + 2$ *rounds.*
- *The explicit proof messages consist of* $O(d \log \ell)$ *field elements for sum-check and* m *field elements for the message* $\tilde{x}$*.*
- *The oracle proof length is* $\ell \cdot (m + n) + n$ *field elements for* $[\![\tilde{x}_\cup]\!], [\![\tilde{w}_\cup]\!]$*, and* $[\![\tilde{w}]\!]$*.*
- *The reduction verifier makes no oracle queries; it outputs a reduced statement containing evaluation claims that are later checked by* PCS *openings.*

- *Excluding linear-time public-input processing (e.g., computing $\boldsymbol{x}$ from $\{\boldsymbol{x}^{(k)}\}_{k\in[\ell]}$), the verifier's non-linear work is dominated by $O(d \log \ell)$ field operations in the sum-check.*
- *The reduced relation $\mathcal{R}_{\mathsf{acc}}$ contains* ***three polynomial oracles****.*

Suppose that PCS *satisfies extractability. The Fiat–Shamir transform* $\mathsf{FS}[\mathsf{IOR}^{\mathsf{PCS}}_{\mathsf{cast}}]$ *yields a non-interactive reduction of knowledge in the ROM.*

4.2 Round-By-Round Soundness

We prove RBR knowledge soundness for the multi-cast IOR $\mathsf{IOR}_{\mathsf{cast}}$ in Lemma 2. We follow the standard RBR proof pattern for public-coin IORs [8,1,22]. We omit the completeness proof.

Proof. Fix an adversarial prover $\mathcal{P}^*$. We bound the probability that $\mathcal{P}^*$ produces a transcript causing the verifier to output a reduced statement in $\mathcal{L}(\mathcal{R}_{\mathsf{acc}})$ without enabling extraction of a corresponding witness. We decompose the analysis into three stages aligned with the protocol structure.

Stage 1: sum-check soundness. Let $G(\boldsymbol{Y})$ be the degree-$(d+1)$ polynomial defined in the interaction phase. If $\sum_{\boldsymbol{y}\in\{0,1\}^{\log \ell}} G(\boldsymbol{y}) \neq 0$, then the $\log \ell$-round sum-check accepts with probability at most

$$\epsilon^{\mathsf{sc}} \leq \frac{\log \ell \cdot (d+1)}{|\mathbb{F}|},$$

by the standard sum-check soundness bound (Definition 12 in Appendix D.2, instantiated with individual degree $d+1$). Conditioned on sum-check acceptance, the transcript defines $(\boldsymbol{\tau}, G(\boldsymbol{\tau}))$ such that

$$G(\boldsymbol{\tau}) = H(\boldsymbol{\tau}) \cdot \tilde{eq}(\boldsymbol{\tau}, \boldsymbol{r}_y)$$

for some $H(\boldsymbol{Y}) = F(\tilde{\boldsymbol{x}}(\boldsymbol{Y}), \tilde{\boldsymbol{w}}(\boldsymbol{Y}))$ consistent with the prover's earlier messages.

Stage 2: defining e and handling division. The verifier defines e by dividing by $\tilde{eq}(\boldsymbol{\tau}, \boldsymbol{r}_y)$. If $\tilde{eq}(\boldsymbol{\tau}, \boldsymbol{r}_y) = 0$, the verifier rejects. Since $\tilde{eq}(\boldsymbol{\tau}, \boldsymbol{r}_y)$ is a non-zero polynomial of total degree $\log \ell$ in $\boldsymbol{r}_y$, the probability that a uniformly random $\boldsymbol{r}_y \leftarrow\!\!\$\ \mathbb{F}^{\log \ell}$ yields $\tilde{eq}(\boldsymbol{\tau}, \boldsymbol{r}_y) = 0$ is at most

$$\epsilon^{\mathsf{div}} \leq \frac{\log \ell}{|\mathbb{F}|},$$

by Schwartz–Zippel. Conditioned on non-rejection, the reduced instance includes a value e that is consistent with the sum-check transcript, *i.e.*, $e = H(\boldsymbol{\tau})$.

Stage 3: partial-evaluation consistency. The reduced relation includes the two evaluation equalities

$$\tilde{x}_{\cup}(\boldsymbol{\tau}, \boldsymbol{r}_x) = \tilde{x}(\boldsymbol{r}_x), \qquad \tilde{w}_{\cup}(\boldsymbol{\tau}, \boldsymbol{r}_w) = \tilde{w}(\boldsymbol{r}_w),$$

with independently sampled $\boldsymbol{r}_x \leftarrow\!\$\ \mathbb{F}^{\log m}$ and $\boldsymbol{r}_w \leftarrow\!\$\ \mathbb{F}^{\log n}$. If either equality is false as a polynomial identity in $\boldsymbol{X}$, then it fails at a random evaluation point with probability at least $1 - \deg/|\mathbb{F}|$. Since the involved polynomials are multilinear in $\boldsymbol{X}$, we obtain the bound

$$\epsilon^{\mathsf{pe}} \leq \frac{\log m + \log n}{|\mathbb{F}|}.$$

Extraction. Conditioned on the complement of the bad events in Stages 1–3, the reduced statement is consistent with the prover's oracle strings. An extractor can output the witness $\boldsymbol{w}$ by reading $[\![\tilde{w}]\!]$ on the Boolean hypercube:

$$\boldsymbol{w}[j+1] \coloneqq \tilde{w}(\mathsf{Bits}(j)) \text{ for } j \in \{0, \ldots, n-1\},$$

and similarly recover the per-instance witnesses from $[\![\tilde{w}_{\cup}]\!]$ if needed. The relation $\mathcal{R}_{\mathsf{acc}}$ then holds by construction of e and by the evaluation equalities above. The extractor runs in time polynomial in the oracle lengths.

Conclusion. Combining the bounds yields an RBR knowledge soundness error

$$\epsilon \leq \epsilon^{\mathsf{sc}} + \epsilon^{\mathsf{div}} + \epsilon^{\mathsf{pe}} \leq \frac{\log \ell \cdot (d+1)}{|\mathbb{F}|} + \frac{\log \ell}{|\mathbb{F}|} + \frac{\log m + \log n}{|\mathbb{F}|},$$

as required. □

5 Building Blocks for Accumulation

We present the two building blocks used to instantiate Quasar for special-sound protocols (SPS). First, we recap the SPS interface and the standard "compress-ν-checks-to-one" transformation in Section 5.1. Then, we specialize our multi-cast reduction to SPS transcripts in Section 5.2. Finally, we define the 2-to-1 folding IOR for SPS-based reduced statements in Section 5.3.

5.1 Recap of Special-Sound Protocols

We adopt the notation of Protostar and ProtoGalaxy [16,30]; see Definition 23 for the formal definition. Let Π_{sps} be a public-coin special-sound protocol for an indexed relation $\mathcal{R}$. Fix public parameters pp (including μ and the field $\mathbb{F}$). Let the public instance be $\mathbb{x} \coloneqq \boldsymbol{x} \in \mathbb{F}^m$ and the witness be $\mathbb{w} \coloneqq \boldsymbol{w} \in \mathbb{F}^n$.

Transcript structure. The prover sends μ messages $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_\mu$. After receiving $\boldsymbol{m}_i$, the verifier samples a public-coin challenge $r_i \leftarrow\!\$\ \mathbb{F}$. Thus the transcript contains $(\{\boldsymbol{m}_i\}_{i\in[\mu]}, \{r_i\}_{i\in[\mu]})$, where $\boldsymbol{m}_{i+1}$ may depend on r_i for all $i \in [\mu - 1]$. We write $\boldsymbol{r} \coloneqq (r_1, \ldots, r_{\mu-1})$ for the challenges that affect the prover's subsequent messages, and we reuse the final challenge as $\alpha \coloneqq r_\mu$.

Homogeneous decomposition. Since $\mathsf{V}_{\mathsf{sps}}$ is polynomial of degree at most d, we can write it as a sum of homogeneous components:

$$\mathsf{V}_{\mathsf{sps}}(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i\in[\mu]}, \boldsymbol{r}) \coloneqq \sum_{j=0}^{d} f_j^{\mathsf{V}_{\mathsf{sps}}}(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i\in[\mu]}, \boldsymbol{r}), \tag{7}$$

where each $f_j^{\mathsf{V}_{\mathsf{sps}}}$ is homogeneous of degree j.

Compressing ν checks to one. Accumulation schemes benefit from replacing the vector check $\mathsf{V}_{\mathsf{sps}}(\cdot) = \mathbf{0}^{\nu}$ with a single scalar equation. We use the standard random linear combination technique [16]. We compress ν verifier equations into one by a random linear combination. We set $\alpha \coloneqq r_\mu$ (the final public-coin challenge) and define $\boldsymbol{\alpha} \coloneqq (\alpha, \alpha^2, \alpha^4, \ldots, \alpha^{2^{\log \nu - 1}})$, together with constraints $\boldsymbol{\alpha}[0] = r_\mu$ and $\boldsymbol{\alpha}[i] = (\boldsymbol{\alpha}[i-1])^2$ for all $i \in [\log \nu - 1]$.

To keep the resulting relation algebraic with low degree, we represent powers via a short vector

$$\boldsymbol{\alpha} \coloneqq (\alpha, \alpha^2, \alpha^4, \ldots, \alpha^{2^{\log \nu - 1}}),$$

assuming ν is a power of two. For each $j \in [\nu]$, let $S \subseteq \{0, \ldots, \log \nu - 1\}$ be such that $j - 1 = \sum_{k\in S} 2^k$. Define $\mathsf{pow}_j(\boldsymbol{X}) \coloneqq \prod_{k\in S} X_k$, so that $\mathsf{pow}_j(\boldsymbol{\alpha}) = \alpha^{j-1}$. We then define the scalar check

$$F(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i\in[\mu]}, \boldsymbol{r}, \boldsymbol{\alpha}) \coloneqq \sum_{j\in[\nu]} \mathsf{pow}_j(\boldsymbol{\alpha}) \cdot \mathsf{V}_{\mathsf{sps}}(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i\in[\mu]}, \boldsymbol{r})[j] = 0. \tag{8}$$

The degree of F is at most $D \coloneqq d + \log \nu$.

We denote the resulting transformed protocol by $\mathsf{CV}[\Pi_{\mathsf{sps}}]$. It augments the transcript with α (or equivalently $\boldsymbol{\alpha}$) and enforces the consistency constraints

$$\boldsymbol{\alpha}[0] = \alpha, \qquad \boldsymbol{\alpha}[i] = (\boldsymbol{\alpha}[i-1])^2 \quad \forall i \in [\log \nu - 1].$$

We provide the step-by-step description in Figure 5.

Lemma 3. *If Π_{sps} is $(k_1, \ldots, k_\mu)$-special-sound for $\mathcal{R}$, then $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ is $(k_1, \ldots, k_\mu, \nu + 1)$-special-sound for $\mathcal{R}$.*

Proof sketch. Given an extractor $\mathcal{E}_{\mathsf{sps}}$ for Π_{sps}, an extractor for $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ invokes $\mathcal{E}_{\mathsf{sps}}$ on the depth-μ transcript subtree to recover the witness. To argue validity, for fixed $(\boldsymbol{x}, \{\boldsymbol{m}_i\}, \boldsymbol{r})$, the scalar $F(\cdot, \boldsymbol{\alpha})$ is a univariate polynomial in α of degree at most $\nu - 1$. If $F(\cdot, \boldsymbol{\alpha}) = 0$ holds at $\nu + 1$ distinct points, then all ν coordinates of $\mathsf{V}_{\mathsf{sps}}(\cdot)$ must be zero (see [16] for the full argument).

5.2 Component 1: Multi-Cast IOR for SPS

We now specialize the multi-cast reduction of Section 4 to SPS transcripts. We view a completed $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ transcript as an index-instance-witness tuple satisfying the relation

$$\mathcal{R}_{\mathsf{sps}} \coloneqq \left\{ \begin{array}{c} \mathtt{i}, \mathtt{x} = (\boldsymbol{x}, \boldsymbol{r}, \alpha, \boldsymbol{\alpha}), \\ \mathtt{y} = \bot, \mathtt{w} = \{\boldsymbol{m}_i\}_{i\in[\mu]} \end{array} : \begin{array}{c} F(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i\in[\mu]}, \boldsymbol{r}, \boldsymbol{\alpha}) = 0 \quad \wedge \\ \boldsymbol{\alpha}[0] = \alpha \ \wedge \ \boldsymbol{\alpha}[i] = (\boldsymbol{\alpha}[i-1])^2 \ \forall i \in [\log \nu - 1] \end{array} \right\},$$

where $\mathtt{i}$ includes the field $\mathbb{F}$, the function $F(\cdot)$, and the degree bounds.

$\mathcal{P}$'s inputs: $\mathsf{pp}, \mathbb{x}; \mathbb{w}$

$\mathcal{V}$'s inputs: $\mathsf{pp}, \mathbb{x}$

1: For $i \in [\mu]$:
   1: $\boldsymbol{m}_i \leftarrow \mathsf{P}_{\mathsf{SPS}}(\boldsymbol{x}, \boldsymbol{w}, \{\boldsymbol{m}_j, r_j\}_{j=1}^{i-1})$.
   2: $\mathcal{P} \to \mathcal{V} : \boldsymbol{m}_i$.
   3: $\mathcal{V} \to \mathcal{P} : r_i \leftarrow\!\$\ \mathbb{F}$.
2: Define $\boldsymbol{r} \coloneqq (r_1, \ldots, r_{\mu-1})$.
3: Set $\alpha \coloneqq r_\mu$.
4: $\mathcal{P}$ computes $\boldsymbol{\alpha} \coloneqq (\alpha, \alpha^2, \alpha^4, \ldots, \alpha^{2^{\log \nu - 1}})$.
5: $\mathcal{P} \to \mathcal{V}$: $\boldsymbol{\alpha}$.
6: $\mathcal{V}$ checks that
   1: $F(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i\in[\mu]}, \boldsymbol{r}, \boldsymbol{\alpha}) = 0$.
   2: $\boldsymbol{\alpha}[0] = r_\mu$.
   3: $\boldsymbol{\alpha}[i] = (\boldsymbol{\alpha}[i-1])^2\ \forall i \in [\log \nu - 1]$.

Fig. 5: Transformed special-sound protocol $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ for relation $\mathcal{R}$.

Why we interleave SPS *with multi-cast.* A naïve construction of a multi-cast IOR for $\mathcal{R}^\ell$ would (i) run $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ independently for each of the ℓ instances, and then (ii) apply $\mathsf{IOR}_{\mathsf{cast}}$ to the resulting ℓ transcripts. After Fiat–Shamir, this would force the RO to absorb the concatenation of ℓ prover messages per round, *i.e.*, inputs of size $\Theta(\ell \cdot |\boldsymbol{m}_i|)$, which is undesirable.

Quasar avoids this by *interleaving* the ℓ executions. In round $i \in [\mu]$, the prover computes all $\{\boldsymbol{m}_i^{(k)}\}_{k\in[\ell]}$ and sends a single union oracle encoding them. This ensures that each challenge r_i is bound to the entire round-i batch while keeping the Fiat–Shamir transcript compact (after PCS compilation, the RO absorbs only succinct digests of the union oracles, rather than $\Theta(\ell)$ raw messages).

We present the protocol $\mathsf{IOR}_{\mathsf{cast\text{-}SPS}}$ in Figure 6. The indexer outputs $(\iota, \mathbb{I}, \mathbb{i}') \leftarrow \mathcal{I}(\mathbb{i})$ with $\mathbb{I} = \bot$ and $\mathbb{i}' = \mathbb{i}$. The prover receives $(\mathbb{i}, \{(\mathbb{x}^{(k)}, \mathbb{w}^{(k)})\}_{k\in[\ell]})$, and the verifier receives $(\iota, \{\mathbb{x}^{(k)}\}_{k\in[\ell]})$.

At a high level, the reduced statement produced by $\mathsf{IOR}_{\mathsf{cast\text{-}SPS}}$ is a conjunction of evaluation subrelations and one F-constraint. Concretely, the reduced statement satisfies $\mathcal{L}(\mathcal{R}_{\mathsf{acc}})$, where $\mathcal{R}_{\mathsf{acc}} \coloneqq \mathcal{R}_{\mathsf{eval}} \times \mathcal{R}_{\mathsf{eval}}^{\mu} \times \mathcal{R}_{\mathsf{eval}}^{\mu} \times \mathcal{R}_F$ with:

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{x}, \boldsymbol{\tau}, \boldsymbol{r}_x, v_x), \mathbb{y} = [\![\tilde{x}_\cup]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}}, \tag{9}$$

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{\tau}, \boldsymbol{r}_w, v_t), \mathbb{y} = [\![\tilde{m}_{\cup,t}]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}}\ \ \forall t \in [\mu], \tag{10}$$

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{r}_w, v_t), \mathbb{y} = [\![\tilde{m}_t]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}}\ \ \forall t \in [\mu], \tag{11}$$

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{x}, \boldsymbol{r}, \alpha, \boldsymbol{\alpha}, e), \mathbb{y} = \bot, \mathbb{w} = \{\boldsymbol{m}_t\}_{t\in[\mu]}) \in \mathcal{R}_F. \tag{12}$$

$\mathcal{P}$'s inputs: $(\mathbb{i}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_{k\in[\ell]})$.

$\mathcal{V}$'s inputs: $\{\mathbb{i}, \mathbb{x}^{(k)}\}_{k\in[\ell]}$.

Interaction phase.

1: For each $k \in [\ell]$, set $\boldsymbol{m}_1^{(k)} \leftarrow \mathsf{P}_{\mathsf{sps}}(\boldsymbol{x}^{(k)}, \boldsymbol{w}^{(k)})$.

2: $\mathcal{P}$ computes:

$\tilde{x}_{\cup}(\boldsymbol{Y}, \boldsymbol{X}) \coloneqq \sum_{k\in[\ell]} \tilde{eq}_{k-1}(\boldsymbol{Y}) \cdot \tilde{x}^{(k)}(\boldsymbol{X})$,

$\tilde{m}_{\cup,1}(\boldsymbol{Y}, \boldsymbol{X}) \coloneqq \sum_{k\in[\ell]} \tilde{eq}_{k-1}(\boldsymbol{Y}) \cdot \tilde{m}_1^{(k)}(\boldsymbol{X})$.

3: $\mathcal{P} \to \mathcal{V}$: $[\![\tilde{x}_{\cup}]\!], [\![\tilde{m}_{\cup,1}]\!]$.

4: $\mathcal{V} \to \mathcal{P} : r_1 \leftarrow\!\$\, \mathbb{F}$.

5: For $i \in [2, \mu]$:

a: For each $k \in [\ell]$, set $\boldsymbol{m}_i^{(k)} \leftarrow \mathsf{P}_{\mathsf{sps}}(\boldsymbol{x}^{(k)}, \boldsymbol{w}^{(k)}, \{\boldsymbol{m}_j^{(k)}, r_j\}_{j=1}^{i-1})$.

b: $\mathcal{P}$ computes $\tilde{m}_{\cup,i}(\boldsymbol{Y}, \boldsymbol{X}) \coloneqq \sum_{k\in[\ell]} \tilde{eq}_{k-1}(\boldsymbol{Y}) \cdot \tilde{m}_i^{(k)}(\boldsymbol{X})$.

c: $\mathcal{P} \to \mathcal{V}$: $[\![\tilde{m}_{\cup,i}]\!]$.

d: $\mathcal{V} \to \mathcal{P}$: $r_i \leftarrow\!\$\, \mathbb{F}$

6: Set $\alpha \coloneqq r_\mu$.

7: $\mathcal{P}$ computes $\boldsymbol{\alpha} \coloneqq (\alpha, \alpha^2, \alpha^4, \ldots, \alpha^{2^{\log \nu - 1}})$.

8: $\mathcal{P} \to \mathcal{V}$: $\boldsymbol{\alpha}$.

9: $\mathcal{V}$ checks:

a: Reject if $\boldsymbol{\alpha}[0] \neq r_\mu$.

b: Reject if $\boldsymbol{\alpha}[i] \neq (\boldsymbol{\alpha}[i-1])^2$ for any $i \in [\log \nu - 1]$.

10: $\mathcal{P}$ and $\mathcal{V}$ run the remaining process of $\mathsf{IOR}_{\mathsf{cast}}$ (Step 4–10).

Output phase.

11: Define $\boldsymbol{x}$ as the coefficient vector of $\tilde{x}$.

12: Define $\boldsymbol{r} \coloneqq (r_1, \ldots, r_{\mu-1})$.

13: Define $e \coloneqq G_{\log \ell}(\tau_{\log \ell}) \cdot \tilde{eq}^{-1}(\boldsymbol{\tau}, \boldsymbol{r}_y)$.

14: $\mathcal{V}$ outputs $\mathbb{x} \coloneqq (\boldsymbol{x}, \boldsymbol{r}, \alpha, \boldsymbol{\alpha}, \boldsymbol{\tau}, \boldsymbol{r}_x, \boldsymbol{r}_w, e)$,

$\mathbb{y} \coloneqq ([\![\tilde{x}_{\cup}]\!], \{[\![\tilde{m}_{\cup,i}]\!]\}_{i\in[\mu]}, \{[\![\tilde{m}_i]\!]\}_{i\in[\mu]})$.

Fig. 6: Interactive oracle reduction $\mathsf{IOR}_{\mathsf{cast\text{-}SPS}}$ from $\mathcal{R}^\ell$ to $\mathcal{R}_{\mathsf{acc\text{-}SPS}}$.

According to Theorem 5, $\mathsf{IOR}_{\mathsf{cast\text{-}SPS}}$ compiles to a non-interactive oracle proof $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}$; see Appendix E.1. Lemma 4 summarizes the resulting security and complexity.

Lemma 4. *Let* $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ *be* $(k_0, \ldots, k_{\mu-1}, \nu+1)$*-special-sound for* $\mathcal{R}$*. Let* $\mathsf{IOR}_{\mathsf{cast}}$ *be an interactive oracle reduction from* $\mathcal{R}_F^\ell$ *to* $\mathcal{R}_{\mathsf{acc}}$*. Then* $\mathsf{IOR}_{\mathsf{cast\text{-}SPS}}$ *is an interactive oracle reduction from* $\mathcal{R}^\ell$ *to* $\mathcal{R}_{\mathsf{acc}}$ *satisfying perfect completeness and RBR knowledge soundness. Moreover, compiling with an extractable* PCS *and applying*

Fiat–Shamir yields a non-interactive reduction $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}} \coloneqq \mathsf{FS}[\mathsf{IOR}_{\mathsf{cast\text{-}SPS}}]$ *in the ROM, with complexity:*

- *The explicit proof contains* m *field elements for the accumulated instance* $\tilde{x}$ *and* $O(d \log \ell)$ *field elements for sum-check.*
- *The oracle string contains one oracle* $[\![\tilde{x}_{\cup}]\!]$ *of size* ℓm*,* μ *oracles* $\{[\![\tilde{m}_{\cup,i}]\!]\}_{i\in[\mu]}$ *of size* ℓn*, and* μ *oracles* $\{[\![\tilde{m}_i]\!]\}_{i\in[\mu]}$ *of size* n*.*
- *The RO is queried* $O(\mu+\log \ell)$ *times on inputs consisting of a constant number of succinct oracle digests and* $O(1)$ *field elements (after* PCS *compilation).*
- *Excluding linear-time public-input processing, the verifier's non-linear work is* $O(d \log \ell)$ *field operations.*
- *The reduced relation* $\mathcal{R}_{\mathsf{acc}}$ *contains* $2\mu + 1$ ***polynomial oracles****.*

5.3 Component 2: 2-to-1 IOR for SPS

We now define the 2-to-1 folding reduction, which maps $(\mathcal{R}_{\mathsf{acc}})^2$ to $\mathcal{R}_{\mathsf{acc}}$. The main tool is *oracle batching* [12,17], which merges two multilinear-oracle evaluation relations into one.

Definition 2 (Adopted from [17]). *An* ***oracle batching protocol*** $\mathsf{IOR}_{\mathsf{batch}}$ *with proximity radius* δ^* *is an interactive oracle reduction from an indexed oracle relation* $\mathcal{R}_0$ *to an indexed oracle relation* $\mathcal{R}_1$*, where:*

$$\mathcal{R}_0 \coloneqq \left\{ \begin{array}{c} \mathbb{i}, \mathbb{x} = (\boldsymbol{x}, v, r), \\ \mathbb{y} = ([\![\tilde{f}_0]\!], [\![\tilde{f}_1]\!]), \mathbb{w} = (\boldsymbol{f}_0, \boldsymbol{f}_1) \end{array} : \sum_{i=0}^{1} \tilde{eq}_i(r) \cdot \tilde{f}_i(\boldsymbol{x}) = v \right\},$$

$$\mathcal{R}_1 \coloneqq \left\{ \mathbb{i}, \mathbb{x} = \{(\boldsymbol{x}_j, v_j)\}_j, \mathbb{y} = [\![\tilde{f}]\!], \mathbb{w} = \bot : \tilde{f}(\boldsymbol{x}_j) = v_j \right\}.$$

The verifier receives $(\boldsymbol{x}, v, r)$ *and oracle access to* $\tilde{f}_0, \tilde{f}_1 \in \mathcal{F}^{(<2)}_{\log n}$*. The prover additionally receives the witnesses* $\boldsymbol{f}_0, \boldsymbol{f}_1$*. After interaction, the verifier outputs an oracle* $[\![\tilde{f}]\!]$ *and a list of evaluation claims* $\tilde{f}(\boldsymbol{x}_j) = v_j$*. We require:*

- *Completeness: if* $\sum_{i=0}^{1} \tilde{eq}_i(r) \cdot \tilde{f}_i(\boldsymbol{x}) = v$ *holds, then the output evaluation claims hold.*
- *Soundness: for every* $\delta \in (0, \delta^*)$*, if* $\Delta_{\mathcal{R}_0}(\mathbb{x}, \mathbb{y}) > \delta$*, then except with error* ϵ_0*,* $\Delta_{\mathcal{R}_1}(\mathbb{x}, \mathbb{y}) > \delta$*.*
- *Succinctness: the proof* π_{batch} *(excluding oracles) and verification time* t_{batch} *are sublinear in the oracle length* $|\tilde{f}|$*.*

Folding $(\mathcal{R}_{\mathsf{acc}})^2$ *to* $\mathcal{R}_{\mathsf{acc}}$*.* We describe the 2-to-1 reduction $\mathsf{IOR}_{\mathsf{fold\text{-}SPS}}$ shown in Fig. 7. The prover receives two valid tuples $(\mathbb{i}_{\mathsf{acc}}, \mathbb{x}^{(k)}_{\mathsf{acc}}, \mathbb{y}^{(k)}_{\mathsf{acc}}, \mathbb{w}^{(k)}_{\mathsf{acc}}) \in \mathcal{R}_{\mathsf{acc}}$ for $k \in \{0, 1\}$, and the verifier receives the corresponding public parts.

To aggregate the two instances algebraically, we introduce a selector variable $Z \in \{0, 1\}$ and define multilinear "folded" values via equality polynomials, *e.g.*,

$$\tilde{\boldsymbol{x}}(Z) \coloneqq \sum_{k=0}^{1} \tilde{eq}_k(Z) \cdot \boldsymbol{x}^{(k)}.$$

We analogously define folded versions of all scalar/vector components of $\mathbb{x}_{\mathsf{acc}}^{(k)}$, and define folded (virtual) oracles for the oracle strings. For $z \in \{0,1\}$, each folded object evaluates to the corresponding object of the selected input $k = z$.

Using these folded objects, the prover and verifier reduce the conjunction of the $2\mu + 1$ defining constraints of $\mathcal{R}_{\mathsf{acc}}$ to a single low-degree check over Z, using a 1-round sum-check for $\sum_{z\in\{0,1\}} G(z) = 0$ as in Section 4.1. Finally, they invoke $2\mu + 1$ instances of $\mathsf{IOR}_{\mathsf{batch}}$ in parallel to batch the paired oracles into a single oracle string for the output statement.

Remark 1. The protocol $\mathsf{IOR}_{\mathsf{fold\text{-}SPS}}$ is a 2-to-1 reduction from $(\mathcal{R}_{\mathsf{acc}})^2$ to $\mathcal{R}_{\mathsf{acc}}$, where $\mathcal{R}_{\mathsf{acc}} = \mathcal{R}_{\mathsf{eval}} \times \mathcal{R}_{\mathsf{eval}}^{\mu} \times \mathcal{R}_{\mathsf{eval}}^{\mu} \times \mathcal{R}_F$. The oracle batching step may output a *list* of evaluation claims (indexed by α, β, γ in Fig. 7) rather than a single claim. This is resolved by adding a constant number of auxiliary evaluation claims at a fixed point (*e.g.*, 0), as in [22,17].

We state the security and complexity of $\mathsf{IOR}_{\mathsf{fold\text{-}SPS}}$ in Lemma 5; the proof appears in Appendix F.1. By Theorem 5, $\mathsf{IOR}_{\mathsf{fold\text{-}SPS}}$ compiles to a non-interactive oracle proof $\mathsf{NIR}_{\mathsf{fold\text{-}SPS}}$; see Appendix E.2.

Lemma 5. *Let* $\mathsf{IOR}_{\mathsf{batch}}$ *be an interactive oracle reduction from* $\mathcal{R}_0$ *to* $\mathcal{R}_1$*. Then* $\mathsf{IOR}_{\mathsf{fold\text{-}SPS}}$ *is an interactive oracle reduction from* $\mathcal{R}_{\mathsf{acc}}^2$ *to* $\mathcal{R}_{\mathsf{acc}}$ *satisfying perfect completeness and RBR knowledge soundness. Compiling with an extractable* PCS *and applying Fiat–Shamir yields a non-interactive reduction* $\mathsf{NIR}_{\mathsf{fold\text{-}SPS}} \coloneqq \mathsf{FS}[\mathsf{IOR}_{\mathsf{fold\text{-}SPS}}^{\mathsf{PCS}}]$ *in the ROM, with complexity:*

- *The explicit proof contains* $O(d+\mu\cdot\delta)$ *field elements for sum-check and opened evaluations, where* δ *denotes the maximum number of evaluation claims output by one invocation of* $\mathsf{IOR}_{\mathsf{batch}}$*.*
- *The oracle proof length is* $O(\mu\ell n)$ *field elements across* $[\![\tilde{x}_{\cup}]\!]$*,* $\{[\![\tilde{m}_{\cup,i}]\!]\}_{i\in[\mu]}$*, and* $\{[\![\tilde{m}_i]\!]\}_{i\in[\mu]}$*.*
- *The verifier time is dominated by* $(2\mu + 1) \cdot t_{\mathsf{batch}}$*.*
- *The reduced relation* $\mathcal{R}_{\mathsf{acc}}$ *contains* $2\mu + 1$ ***polynomial oracles****.*

6 Quasar: Accumulation for SPS

This section instantiates Quasar as a multi-instance accumulation scheme for special-sound protocols (SPS) and proves Main Theorem 1. We first present the composed construction that combines our multi-cast reduction with our 2-to-1 folding reduction, and state its formal guarantees as Theorem 2. We then explain how the construction is compiled under different polynomial commitment schemes (PCS) and where the PCS choice affects the verifier and extractor. Finally, we give the concrete NARK and accumulation algorithms induced by the two non-interactive reductions from Section 5.

6.1 Overview

We recall the syntax of an accumulation scheme (formalized in Appendix D.4). An *accumulator* is a tuple $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ such that $(\mathbb{i}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w}) \in \mathcal{R}_{\mathsf{acc}}$ for the target accumulator relation $\mathcal{R}_{\mathsf{acc}}$. The definition below fixes the syntax.

Definition 3 (Accumulation scheme (informal)). *An accumulation scheme* $\mathsf{ACC} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V}, \mathcal{D})$ *for a non-interactive argument* $\mathsf{NARK} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *consists of:*

- $\mathsf{ACC}.\mathcal{G}(1^\lambda)$*, which outputs public parameters* pp*.*
- $\mathsf{ACC}.\mathcal{I}(\mathsf{pp}, \mathbb{i})$*, which outputs keys* $(\mathsf{apk}, \mathsf{avk}, \mathsf{adk})$*.*
- $\mathsf{ACC}.\mathcal{P}(\mathsf{apk}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi, \mathsf{acc}^{(0)})$*, which outputs an updated accumulator* acc *and an accumulation proof* pf*.*
- $\mathsf{ACC}.\mathcal{V}(\mathsf{avk}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi.\mathbb{x}, \mathsf{acc}^{(0)}.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$*, which checks that* $\mathsf{acc}.\mathbb{x}$ *is a valid update with respect to the public inputs and* pf*.*
- $\mathsf{ACC}.\mathcal{D}(\mathsf{adk}, \mathsf{acc})$*, which decides whether* acc *is valid.*

We call ACC multi-instance *if one accumulation step can absorb ℓ NARK statements $\{\mathbb{x}^{(k)}\}_{k\in[\ell]}$. We call* ACC sublinear *if the* verification overhead beyond reading the public inputs *is $o(n \cdot \ell)$.*

Theorem 2 (Instantiation of Main Theorem 1 for SPS). *There exists a polynomial-time transformation T such that the following holds. Let $\mathcal{R}$ be an indexed relation parameterized by* pp*. Let $\mathcal{R}_{\mathsf{acc}}$ be an indexed oracle relation parameterized by the same* pp *and implicitly relative to the random oracle. Given the following non-interactive reductions in the random-oracle model:*

- $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}$*, a multi-instance reduction from $(\mathcal{R}^{\mathsf{PCS}})^\ell$ to $\mathcal{R}^{\mathsf{PCS}}_{\mathsf{acc}}$, and*
- $\mathsf{NIR}_{\mathsf{fold\text{-}SPS}}$*, a 2-to-1 reduction from $(\mathcal{R}^{\mathsf{PCS}}_{\mathsf{acc}})^2$ to $\mathcal{R}^{\mathsf{PCS}}_{\mathsf{acc}}$ with the same generator algorithm as* $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}$*,*

the transformation outputs $T[\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}, \mathsf{NIR}_{\mathsf{fold\text{-}SPS}}, \mathcal{R}_{\mathsf{acc}}] = (\mathsf{NARK}, \mathsf{ACC})$, where NARK *is a non-interactive argument for $\mathcal{R}$ and* ACC *is a* ***multi-instance*** *accumulation scheme for* NARK*.*

Let m be the instance length (over $\mathbb{F}$) and let each SPS *transcript consist of μ prover messages of length at most n (over $\mathbb{F}$). Then the resulting accumulation scheme admits the following performance bounds. In particular, each invocation of the batching verifier $\mathsf{NIR}_{\mathsf{batch}}.\mathcal{V}$ combines only two committed oracles (Fig. 11), and thus uses $O(1)$* CRC *operations independent of ℓ.*

- *The accumulation prover cost is dominated by $O(\ell m + \ell\mu n)$ field operations to process the ℓ inputs, plus $\mu + \log \ell$ random-oracle queries, plus the cost of running $2\mu + 1$ instances of the batching prover $\mathsf{NIR}_{\mathsf{batch}}.\mathcal{P}$.*
- *The accumulation verifier cost is dominated by $O(\log \ell)$ field operations (excluding linear public-input processing), plus $\mu + \log \ell$ random-oracle queries, plus the cost of running $2\mu + 1$ instances of the batching verifier $\mathsf{NIR}_{\mathsf{batch}}.\mathcal{V}$, which contributes $O(\mu)$* CRC *operations in total and is independent of ℓ.*

Table 3: Comparisons between Multi-Accumulation Schemes (n: polynomial size)

Schemes	**Assump.**	**Prover**	**Proof size**	**Batch proof size**
IPA [18]	DL	$O(n \log n)$	$O(\log n)$	$O(\log n)$
Mercury [31]	DL	$O(n \log n)$	$O(1)$	$O(1)$
RS codes [22]	Hash	$O(n \log n)$	$O(\log n)$	$O(\frac{\lambda}{\log(1/\rho)} \cdot \log n)$
Linear codes [17]	Hash	$O(n)$	$O(\log n)$	$O(\frac{\lambda}{\log(1/\rho)} \cdot \log n)$

- *The decider cost is dominated by checking the $O(\mu)$ evaluation claims in $\mathcal{R}_{\mathsf{acc}}$ via* $\mathsf{PCS.Verify}$.

For simplicity, we focus on distance-preserving accumulation in this paper. The extension to promise relations follows the proof strategy of Bünz *et al.* [22].

Where the PCS *matters.* In our construction, the PCS choice only affects the instantiation of the oracle batching protocol $\mathsf{IOR}_{\mathsf{batch}}$ (Definition 2) and the resulting costs of $\mathsf{PCS.Open}$ and $\mathsf{PCS.Verify}$ for the induced evaluation claims. Quasar's multi-cast step reduces the number of *committed objects* that must be combined across recursion, so the recursive overhead is governed by the batching interface rather than by ℓ-way commitment combinations. Accordingly, the key requirement on the PCS instantiation is that the batching protocol is *succinct* in the sense of Definition 2, *i.e.*, its proof size and verification time are sublinear in the underlying polynomial length.

For homomorphic PCS, batching is direct because commitments can be linearly combined in the commitment group. For non-homomorphic PCS (*e.g.*, code-based commitments [5,35,56,57]), recent work develops succinct batching IORs and their Fiat–Shamir instantiations [22,17]. We summarize representative instantiations in Table 3, and provide additional details for code-based instantiations in Appendix A.

6.2 Construction for Accumulation Scheme

We follow the roadmap of Bünz *et al.* [22] and describe (i) the induced NARK and (ii) the induced accumulation scheme.

NARK construction. We construct $\mathsf{NARK} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ by wrapping the reduction $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}$. Intuitively, $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}$ reduces ℓ SPS statements to one accumulator statement in $\mathcal{R}_{\mathsf{acc}}^{\mathsf{PCS}}$, and the NARK proof consists of the reduction transcript together with a witness for the reduced relation.

- $\mathsf{NARK}.\mathcal{G}(1^\lambda) \coloneqq \mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{G}(1^\lambda)$.
- $\mathsf{NARK}.\mathcal{I}(\mathsf{pp}, \mathbb{i})$ runs $(\mathsf{pk}_{\mathsf{cast}}, \mathsf{vk}_{\mathsf{cast}}, \mathbb{i}') \leftarrow \mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{I}(\mathsf{pp}, \mathbb{i})$ and outputs $(\mathsf{pk}, \mathsf{vk}) \coloneqq (\mathsf{pk}_{\mathsf{cast}}, (\mathsf{vk}_{\mathsf{cast}}, \mathbb{i}', \mathsf{pp}))$.
- $\mathsf{NARK}.\mathcal{P}(\mathsf{pk}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_{k\in[\ell]})$ runs $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{P}(\mathsf{pk}_{\mathsf{cast}}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_{k\in[\ell]})$ to obtain $(\pi_{\mathsf{cast}}, \mathbb{w}_{\mathsf{acc}})$ and outputs $\pi \coloneqq (\pi.\mathbb{x}, \pi.\mathbb{w}) \coloneqq (\pi_{\mathsf{cast}}, \mathbb{w}_{\mathsf{acc}})$.

- $\mathsf{NARK}.\mathcal{V}(\mathsf{vk}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi)$ parses $\mathsf{vk} = (\mathsf{vk}_{\mathsf{cast}}, \mathbb{i}', \mathsf{pp})$ and $\pi = (\pi_{\mathsf{cast}}, \mathbb{w}_{\mathsf{acc}})$, computes $\mathbb{x}_{\mathsf{acc}} \leftarrow \mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{V}(\mathsf{vk}_{\mathsf{cast}}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi_{\mathsf{cast}})$, and accepts iff $(\mathbb{i}', \mathbb{x}_{\mathsf{acc}}, \mathbb{w}_{\mathsf{acc}}) \in \mathcal{R}_{\mathsf{acc}}^{\mathsf{PCS}}(\mathsf{pp})$.

Proof sketch. Completeness is immediate from the completeness of $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}$. Knowledge soundness follows by a straight-line reduction from any NARK adversary to an adversary against $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}$, and then invoking the extractor guaranteed by Lemma 4.

Accumulation scheme. We construct $\mathsf{ACC} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V}, \mathcal{D})$ by combining $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}$ and $\mathsf{NIR}_{\mathsf{fold\text{-}SPS}}$. Given a NARK proof π for ℓ statements, the prover first derives the corresponding reduced accumulator instance via $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{V}$, and then folds it with the running accumulator via $\mathsf{NIR}_{\mathsf{fold\text{-}SPS}}$.

- $\mathsf{ACC}.\mathcal{G}(1^\lambda) \coloneqq \mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{G}(1^\lambda)$.
- $\mathsf{ACC}.\mathcal{I}(\mathsf{pp}, \mathbb{i})$ sequentially runs
  $(\mathsf{pk}_{\mathsf{cast}}, \mathsf{vk}_{\mathsf{cast}}, \mathbb{i}') \leftarrow \mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{I}(\mathsf{pp}, \mathbb{i})$ and
  $(\mathsf{pk}_{\mathsf{fold}}, \mathsf{vk}_{\mathsf{fold}}, \mathbb{i}') \leftarrow \mathsf{NIR}_{\mathsf{fold\text{-}SPS}}.\mathcal{I}(\mathsf{pp}, \mathbb{i})$ and outputs
  $\mathsf{apk} \coloneqq (\mathsf{vk}_{\mathsf{cast}}, \mathsf{pk}_{\mathsf{fold}}, \mathsf{vk}_{\mathsf{fold}})$, $\mathsf{avk} \coloneqq (\mathsf{vk}_{\mathsf{cast}}, \mathsf{vk}_{\mathsf{fold}})$, and $\mathsf{adk} \coloneqq (\mathbb{i}', \mathsf{pp})$.
- $\mathsf{ACC}.\mathcal{P}(\mathsf{apk}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi, \mathsf{acc}^{(0)})$ parses $\pi = (\pi_{\mathsf{cast}}, \mathbb{w}_{\mathsf{acc}}^{(1)})$ and:
  - computes $\mathsf{acc}^{(1)}.\mathbb{x} \leftarrow \mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{V}(\mathsf{vk}_{\mathsf{cast}}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi_{\mathsf{cast}})$
  - sets $\mathsf{acc}^{(1)}.\mathbb{w} \coloneqq \mathbb{w}_{\mathsf{acc}}^{(1)}$,
  - runs $(\mathsf{pf}, \mathsf{acc}.\mathbb{w}) \leftarrow \mathsf{NIR}_{\mathsf{fold\text{-}SPS}}.\mathcal{P}(\mathsf{pk}_{\mathsf{fold}}, \{\mathsf{acc}^{(0)}, \mathsf{acc}^{(1)}\})$,
  - computes $\mathsf{acc}.\mathbb{x} \leftarrow \mathsf{NIR}_{\mathsf{fold\text{-}SPS}}.\mathcal{V}(\mathsf{vk}_{\mathsf{fold}}, \{\mathsf{acc}^{(0)}.\mathbb{x}, \mathsf{acc}^{(1)}.\mathbb{x}\}, \mathsf{pf})$,
  - outputs $(\mathsf{acc}, \mathsf{pf})$.
- $\mathsf{ACC}.\mathcal{V}(\mathsf{avk}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi_{\mathsf{cast}}, \mathsf{acc}^{(0)}.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$:
  - computes $\mathsf{acc}^{(1)}.\mathbb{x} \leftarrow \mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{V}(\mathsf{vk}_{\mathsf{cast}}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi_{\mathsf{cast}})$,
  - checks that $\mathsf{acc}.\mathbb{x} = \mathsf{NIR}_{\mathsf{fold\text{-}SPS}}.\mathcal{V}(\mathsf{vk}_{\mathsf{fold}}, \{\mathsf{acc}^{(0)}.\mathbb{x}, \mathsf{acc}^{(1)}.\mathbb{x}\}, \mathsf{pf})$.
- $\mathsf{ACC}.\mathcal{D}(\mathsf{adk} = (\mathbb{i}', \mathsf{pp}), \mathsf{acc})$: accepts iff $(\mathbb{i}', \mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w}) \in \mathcal{R}_{\mathsf{acc}}^{\mathsf{PCS}}(\mathsf{pp})$.

Proof sketch. Completeness is immediate from the completeness of $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}$ and $\mathsf{NIR}_{\mathsf{fold\text{-}SPS}}$. Knowledge soundness follows by reducing any successful accumulator forger to a forger against $\mathsf{NIR}_{\mathsf{fold\text{-}SPS}}$, and then invoking the extractor guaranteed by Lemma 5, while using $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{V}$ to define the folded public instance consistently.

7 Conclusion

Multi-instance recursion is attractive because it reduces the number of recursive steps while exposing parallelism within each step. Its main practical bottleneck is that existing designs often pay a linear number of circuit-expensive commitment-mixing operations as the number of inputs grows. Quasar addresses this bottleneck by introducing a multi-cast reduction that compresses ℓ commitment-combination checks into a constant-size committed interface. As a result, the recursive verifier incurs $O(1)$ in-circuit CRC operations per step (*i.e.*, commitment

random linear combinations), while public-input processing remains a separate linear pass as usual.This separation provides a clean cost model for reasoning about recursion overhead in settings where commitment mixing is the dominant in-circuit expense. We instantiate Quasar with multiple PCS backends and report circuit-level evidence that the CRC layer can be made substantially cheaper than in prior multi-instance folding baselines.

Quasar is intentionally scoped to reduce recursion overhead inside the circuit, rather than to eliminate the inevitable linear work of reading and validating public inputs. When CRC is not the dominant in-circuit cost for a given backend, the end-to-end benefit can be correspondingly smaller, and our analysis makes this dependency explicit. A natural next direction is to integrate Quasar into fully end-to-end recursive IVC stacks across more backends and to tighten the constant factors in the RO and field-work components. Another direction is to extend the multi-cast perspective to workflows that maintain multiple running states, where interface design and accumulation structure become more subtle. Overall, Quasar reframes multi-instance recursion around a simple principle, namely, preserve a constant-size committed interface under composition, and provides a concrete route to constant-CRC recursion overhead in practice.

References

1. Arnon, G., Chiesa, A., Fenzi, G., Yogev, E.: STIR: Reed-Solomon proximity testing with fewer queries. In: CRYPTO Part X. pp. 380–413 (2024). https://doi.org/10.1007/978-3-031-68403-6_12
2. Arun, A., Setty, S., Thaler, J.: Jolt: SNARKs for virtual machines via lookups. In: EUROCRYPT Part VI. pp. 3–33 (2024). https://doi.org/10.1007/978-3-031-58751-1_1
3. Attema, T., Cramer, R., Kohl, L.: A compressed Σ-protocol theory for lattices. In: CRYPTO Part II. pp. 549–579 (2021). https://doi.org/10.1007/978-3-030-84245-1_19
4. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: STOC. pp. 21–31 (1991). https://doi.org/10.1145/103418.103428
5. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast Reed-Solomon interactive oracle proofs of proximity. In: ICALP. pp. 14:1–14:14 (2018). https://doi.org/10.4230/LIPICS.ICALP.2018.14
6. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: CRYPTO Part III. pp. 701–732 (2019). https://doi.org/10.1007/978-3-030-26954-8_23
7. Ben-Sasson, E., Chiesa, A., Goldberg, L., Gur, T., Riabzev, M., Spooner, N.: Linear-size constant-query IOPs for delegating computation. In: TCC B Part II. pp. 494–521 (2019). https://doi.org/10.1007/978-3-030-36033-7_19
8. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: TCC. pp. 31–60 (2016). https://doi.org/10.1007/978-3-662-53644-5_2
9. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. Algorithmica **79**(4), 1102–1160 (2017). https://doi.org/10.1007/S00453-016-0221-0
10. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKs and proof-carrying data. In: STOC. pp. 111–120 (2013). https://doi.org/10.1145/2488608.2488623
11. Boneh, D., Chen, B.: LatticeFold: A lattice-based folding scheme and its applications to succinct proof systems. In: ASIACRYPT Part III. pp. 330–362 (2025)
12. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo infinite: Proof-carrying data from additive polynomial commitments. In: CRYPTO Part I. pp. 649–680 (2021). https://doi.org/10.1007/978-3-030-84242-0_23
13. Bootle, J., Chiesa, A., Hu, Y., Orrù, M.: Gemini: Elastic SNARKs for diverse environments. In: EUROCRYPT Part II. pp. 427–457 (2022). https://doi.org/10.1007/978-3-031-07085-3_15
14. Bowe, S., Grigg, J., Hopwood, D.: Recursive proof composition without a trusted setup. IACR Cryptol. ePrint Arch. 2019/1021 (2019), https://eprint.iacr.org/2019/1021
15. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: S&P. pp. 315–334 (2018), https://doi.org/10.1109/SP.2018.00020
16. Bünz, B., Chen, B.: Protostar: Generic efficient accumulation/folding for special-sound protocols. In: ASIACRYPT Part II. pp. 77–110 (2023). https://doi.org/10.1007/978-981-99-8724-5_3
17. Bünz, B., Chiesa, A., Fenzi, G., Wang, W.: Linear-time accumulation schemes. In: TCC Part I. pp. 11–43 (2025). https://doi.org/10.1007/978-3-032-12287-2_13

18. Bünz, B., Chiesa, A., Lin, W., Mishra, P., Spooner, N.: Proof-carrying data without succinct arguments. In: CRYPTO Part I. pp. 681–710 (2021). https://doi.org/10.1007/978-3-030-84242-0_24
19. Bünz, B., Chiesa, A., Mishra, P.: Recursive proof composition from accumulation schemes. In: TCC Part II. pp. 1–37 (2020). https://doi.org/10.1007/978-3-030-64378-2_1
20. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Proof-carrying data from accumulation schemes. Cryptology ePrint Archive, Report 2020/499 (2020), https://eprint.iacr.org/2020/499
21. Bünz, B., Mishra, P., Nguyen, W., Wang, W.: Accumulation without homomorphism. In: ITCS. pp. 23:1–23:25 (2025). https://doi.org/10.4230/LIPICS.ITCS.2025.23
22. Bünz, B., Mishra, P., Nguyen, W., Wang, W.: Arc: Accumulation for Reed-Solomon codes. In: CRYPTO Part VII. pp. 128–160 (2025). https://doi.org/10.1007/978-3-032-01907-3_5
23. Chen, B.: Symphony: Scalable SNARKs in the random oracle model from lattice-based high-arity folding. Cryptology ePrint Archive, Report 2025/1905 (2025), https://eprint.iacr.org/2025/1905
24. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: HyperPlonk: PlonK with linear-time prover and high-degree custom gates. In: EUROCRYPT Part II. pp. 499–530 (2023). https://doi.org/10.1007/978-3-031-30617-4_17
25. Chen, W., Chiesa, A., Dauterman, E., Ward, N.P.: Reducing participation costs via incremental verification for ledger systems. Cryptology ePrint Archive, Report 2020/1522 (2020), https://eprint.iacr.org/2020/1522
26. Chiesa, A.: Proof-Carrying Data. Ph.D. thesis, Massachusetts Institute of Technology (2010)
27. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: EUROCRYPT Part I. pp. 738–768 (2020). https://doi.org/10.1007/978-3-030-45721-1_26
28. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: EUROCRYPT Part I. pp. 769–793 (2020). https://doi.org/10.1007/978-3-030-45721-1_27
29. Chiesa, A., Tromer, E., Virza, M.: Cluster computing in zero knowledge. In: EUROCRYPT. pp. 371–403 (2015). https://doi.org/10.1007/978-3-662-46803-6_13
30. Eagen, L., Gabizon, A.: Protogalaxy: Efficient protostar-style folding of multiple instances. Cryptology ePrint Archive, Report 2023/1106 (2023), https://eprint.iacr.org/2023/1106
31. Eagen, L., Gabizon, A.: MERCURY: A multilinear polynomial commitment scheme with constant proof size and no prover FFTs. Cryptology ePrint Archive, Report 2025/385 (2025), https://eprint.iacr.org/2025/385
32. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO. pp. 186–194 (1986). https://doi.org/10.1007/3-540-47721-7_12
33. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch 2019/953 (2019)
34. Ganesh, C., Patranabis, S., Singh, N.: Samaritan: Linear-time prover SNARK from new multilinear polynomial commitments. In: ASIACRYPT Part V. pp. 456–489 (2025), https://doi.org/10.1007/978-981-95-5116-3_15

35. Golovnev, A., Lee, J., Setty, S., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In: CRYPTO Part II. pp. 193–226 (2023). https://doi.org/10.1007/978-3-031-38545-2_7
36. Grassi, L., Khovratovich, D., Schofnegger, M.: Poseidon2: A faster version of the poseidon hash function. In: AFRICACRYPT. pp. 177–203 (2023), https://doi.org/10.1007/978-3-031-37679-5_8
37. Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT Part II. pp. 305–326 (2016). https://doi.org/10.1007/978-3-662-49896-5_12
38. Hirai, Y.: Defining the Ethereum virtual machine for interactive theorem provers. In: FC. pp. 520–535 (2017). https://doi.org/10.1007/978-3-319-70278-0_33
39. Kalapothas, S., Galetakis, M., Flamis, G., Plessas, F., Kitsos, P.: A survey on RISC-V-based machine learning ecosystem. Information **14**(2), 64 (2023). https://doi.org/10.3390/INF014020064
40. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: ASIACRYPT. pp. 177–194 (2010). https://doi.org/10.1007/978-3-642-17373-8_11
41. Kothapalli, A., Setty, S.: HyperNova: Recursive arguments for customizable constraint systems. In: CRYPTO Part X. pp. 345–379 (2024). https://doi.org/10.1007/978-3-031-68403-6_11
42. Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive zero-knowledge arguments from folding schemes. In: CRYPTO Part IV. pp. 359–388 (2022). https://doi.org/10.1007/978-3-031-15985-5_13
43. Kothapalli, A., Setty, S.T.V.: SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Report 2022/1758 (2022), https://eprint.iacr.org/2022/1758
44. Kothapalli, A., Setty, S.T.V.: CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. Cryptology ePrint Archive, Report 2023/1192 (2023), https://eprint.iacr.org/2023/1192, https://eprint.iacr.org/2023/1192
45. Liu, T., Xie, X., Zhang, Y.: zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In: CCS. pp. 2968–2985 (2021). https://doi.org/10.1145/3460120.3485379
46. Liu, T., Zhang, Z., Zhang, Y., Hu, W., Zhang, Y.: Ceno: Non-uniform, segment and parallel zero-knowledge virtual machine. J. Cryptol. **38**(2), 17 (2025). https://doi.org/10.1007/s00145-024-09533-2
47. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM **39**(4), 859–868 (1992). https://doi.org/10.1145/146585.146605
48. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: CCS. pp. 2111–2128 (2019). https://doi.org/10.1145/3319535.3339817
49. Microsoft: Nova: Recursive SNARKs without trusted setup. https://github.com/Microsoft/Nova
50. Nguyen, W., Datta, T., Chen, B., Tyagi, N., Boneh, D.: Mangrove: A scalable framework for folding-based SNARKs. In: CRYPTO Part X. pp. 308–344 (2024). https://doi.org/10.1007/978-3-031-68403-6_10
51. Pappas, C., Papadopoulos, D., Papamanthou, C.: HydraProofs: Optimally computing all proofs in a vector commitment (with applications to efficient zk-SNARKs over data from multiple users). In: S&P. pp. 3421–3439 (2025), https://doi.org/10.1109/SP61157.2025.00204

52. Setty, S.T.V.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: CRYPTO Part III. pp. 704–737 (2020). https://doi.org/10.1007/978-3-030-56877-1_25
53. Sun, H., Li, J., Zhang, H.: zkLLM: Zero knowledge proofs for large language models. In: CCS. pp. 4405–4419 (2024). https://doi.org/10.1145/3658644.3670334
54. Systems, P., Derei, T., Geren, C., Kaufman, M., Klein, J., Islam, R.: Benchmarking the Plonk, TurboPlonk, and UltraPlonk proving systems. https://api.semanticscholar.org/CorpusID:259902935 (2023)
55. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: TCC. pp. 1–18 (2008). https://doi.org/10.1007/978-3-540-78524-8_1
56. Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. In: CRYPTO Part IV. pp. 299–328 (2022). https://doi.org/10.1007/978-3-031-15985-5_11
57. Zeilberger, H., Chen, B., Fisch, B.: BaseFold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In: CRYPTO Part X. pp. 138–169 (2024). https://doi.org/10.1007/978-3-031-68403-6_5
58. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In: SP. pp. 863–880 (2017). https://doi.org/10.1109/SP.2017.43
59. Zhou, W., Cai, Y., Peng, Y., Wang, S., Ma, K., Li, F.: VeriDB: An SGX-based verifiable database. In: SIGMOD. pp. 2182–2194 (2021). https://doi.org/10.1145/3448016.3457308

A Multi-instance IVC

In this section, we instantiate multi-instance IVC from our sublinear accumulation scheme. We first define multi-instance IVC and its security properties. We then sketch the standard black-box route from a NARK plus an accumulation scheme to (multi-instance) IVC/PCD. Finally, we instantiate the required components using a multilinear plonkish constraint system (HyperPlonk) and a code-based PCS.

A.1 Definitions for Multi-Instance IVC

We extend the IVC definition of Kothapalli *et al.* [42] from one instance per step to ℓ instances per step by following [18]. To support multiple predicates, we allow the prover to choose at each step i a predicate φ_i from a specified set Φ.

Definition 4 (Multi-instance IVC). *A tuple of PPT algorithms defines a multi-instance incrementally verifiable computation (IVC) as* $\mathsf{IVC} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *with the following interfaces:*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$ *samples and outputs public parameters* pp.
- $\mathcal{I}(\mathsf{pp}, \boldsymbol{\varphi}) \to (\mathsf{ipk}, \mathsf{ivk})$ *takes* pp *and a predicate vector* $\boldsymbol{\varphi} \in \Phi^*$*, and outputs a prover key* ipk *and a verifier key* ivk.

- $\mathcal{P}(\mathsf{ipk}, i, \{z_i^{(k)}, w_i^{(k)}, z_{i+1}^{(k)}\}_k, \Pi_i) \to \Pi_{i+1}$ *takes* ipk*, a step counter* i*,* ℓ *local transitions* $\{(z_i^{(k)}, w_i^{(k)}, z_{i+1}^{(k)})\}_k$*, and a proof* Π_i*, and outputs a new proof* Π_{i+1} *attesting the correctness of* $\{z_{i+1}^{(k)}\}_k$*.*
- $\mathcal{V}(\mathsf{ivk}, i, \{z_i^{(k)}\}_k, \Pi_i) \to b$ *takes* ivk*, a counter* i*,* ℓ *public states* $\{z_i^{(k)}\}_k$*, and a proof* Π_i*, and outputs* $b \in \{0,1\}$ *indicating reject or accept.*

A multi-instance IVC quadruple $(\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ should satisfy perfect completeness, knowledge soundness, and zero knowledge. We recall the first two properties below.

Definition 5 (Perfect completeness). IVC *has perfect completeness if for every PPT adversary* $\mathcal{A}$*, the probability* $\Pr[E_1 \mid E_2]$ *equals* 1*, where*

$$E_1 = \left\{ \begin{array}{l} \varphi_i \in \Phi \ \wedge \ \mathcal{V}(\mathsf{ivk}, i, \{z_i^{(k)}\}_k, \Pi_i) = 1 \ \wedge \ \varphi_i\Big(i, \{z_i^{(k)}, w_i^{(k)}, z_{i+1}^{(k)}\}_k\Big) = 1 \\ \implies \ \mathcal{V}(\mathsf{ivk}, i+1, \{z_{i+1}^{(k)}\}_k, \Pi_{i+1}) = 1 \end{array} \right\},$$

and

$$E_2 = \left\{ \begin{array}{r} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (\boldsymbol{\varphi}, i, \{z_i^{(k)}, w_i^{(k)}, z_{i+1}^{(k)}\}_k, \Pi_i) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{pp}, \boldsymbol{\varphi}) \\ \Pi_{i+1} \leftarrow \mathcal{P}(\mathsf{ipk}, i, \{z_i^{(k)}, w_i^{(k)}, z_{i+1}^{(k)}\}_k, \Pi_i) \end{array} \right\}.$$

Definition 6 (Knowledge soundness). IVC *has knowledge soundness (w.r.t. an auxiliary-input distribution* $\mathcal{D}$*) if for every expected PPT adversary* $\tilde{\mathcal{P}}$*, there exists an expected PPT extractor* $\mathsf{Ext}_{\tilde{\mathcal{P}}}$ *such that for every set* Z*, the difference of the following two probabilities is negligible:*

$$\Pr\left[\begin{array}{l} \boldsymbol{\varphi} \subseteq \Phi \\ \wedge \ (\mathsf{pp}, \mathsf{ai}, \boldsymbol{\varphi}, \{z_n^{(k)}\}_k, \mathsf{ao}) \in Z \\ \wedge \ \forall i, \ \varphi_i\Big(i, \{z_i^{(k)}, w_i^{(k)}, z_{i+1}^{(k)}\}_k\Big) = 1 \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}) \\ (\boldsymbol{\varphi}, \mathsf{out}, \mathsf{ao}) \leftarrow \mathsf{Ext}_{\tilde{\mathcal{P}}}(\mathsf{pp}, \mathsf{ai}) \\ \textit{where } \mathsf{out} := \{z_i^{(k)}, w_i^{(k)}, z_{i+1}^{(k)}\}_{k, i \in [n-1]} \end{array} \right],$$

and

$$\Pr\left[\begin{array}{l} \boldsymbol{\varphi} \subseteq \Phi \\ \wedge \ (\mathsf{pp}, \mathsf{ai}, \boldsymbol{\varphi}, \{z_n^{(k)}\}_k, \mathsf{ao}) \in Z \\ \wedge \ \mathcal{V}(\mathsf{ivk}, n, \{z_n^{(k)}\}_k, \Pi_n) = 1 \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}) \\ (\boldsymbol{\varphi}, \{z_n^{(k)}\}_k, \Pi_n, \mathsf{ao}) \leftarrow \tilde{\mathcal{P}}(\mathsf{pp}, \mathsf{ai}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{pp}, \boldsymbol{\varphi}) \end{array} \right].$$

Here ai *and* ao *denote auxiliary input and auxiliary output. This formulation follows the strong extraction style of [10].*

By [18], an IVC/PCD can be constructed from a NARK together with an accumulation scheme. We recall the corresponding statement for multi-instance IVC.

Theorem 3 (Multi-instance IVC from accumulation schemes [18]). *Given a NARK for circuit satisfiability and a multi-instance accumulation scheme for that NARK in the standard model, there exists an efficient transformation that outputs a multi-instance IVC scheme for constant-depth compliance predicates, assuming that the circuit complexity of the accumulation verifier is sublinear in its input. Moreover, if the NARK and accumulation scheme are secure against quantum adversaries, then the resulting PCD is also secure against quantum adversaries. If* both *the NARK and accumulation scheme are zero knowledge, then the resulting PCD is also zero knowledge.*

A.2 IVC Construction

As stated in Theorem 3, there is a generic construction of (multi-instance) IVC from a NARK and a multi-instance accumulation scheme. We sketch the interfaces we use for instantiation.

- Given ℓ predicates and transitions $\{(z_i^{(k)}, w_i^{(k)}, z_{i+1}^{(k)})\}_k$, a NARK proof π, and a prior accumulator acc, the prover runs the accumulation prover $\mathsf{ACC}.\mathcal{P}$ to obtain an updated accumulator acc' and an accumulation proof pf.
- The prover forms ℓ recursive circuits that (i) compute φ_i on the k-th transition, and (ii) embed the accumulation verification for the pair $(\pi.\mathbb{x}, \mathsf{acc}.\mathbb{x})$ and update to $\mathsf{acc}'.\mathbb{x}$ (one circuit can carry this verification, while the remaining circuits can use dummy wiring as needed by the constraint system).
- Each recursive circuit is arithmetized into an instance-witness pair for the underlying constraint system. The prover invokes the proving algorithm of our multi-cast reduction (compiled with a PCS) to produce a reduced proof that is suitable for recursive verification.
- The prover outputs the updated accumulator acc' and the new IVC proof state $\mathit{\Pi}_{i+1}$ induced by the recursive proof.
- The verifier derives the relevant reduced public instance and checks the accumulator update by running the decider on acc'.

In our setting, we additionally target a linear-time prover and plausible post-quantum security. A natural instantiation route is a multilinear plonkish constraint system (HyperPlonk) together with a linear-code-based PCS. We describe the required SPS components and the resulting batching-based accumulation instantiation next.

SPS for HyperPlonk Relations. We first define the multilinear plonkish relation (HyperPlonk) as an indexed oracle relation in Definition 7 and then present the corresponding SPS protocols in Figure 8, 9, and 10.

Definition 7 (Multilinear plonkish relation). *Fix public parameters including the field $\mathbb{F}$, the instance length m, the number of gates μ, the number of selector types s, the fan-in/fan-out parameter n, and an algebraic map*

$f : \mathbb{F}^{s+n} \to \mathbb{F}$ *of degree* d*. The indexed oracle relation* $\mathcal{R}_{\mathsf{plonk}}$ *is the set of all tuples*

$$(\mathbb{i} \coloneqq (\boldsymbol{q}, \sigma), \mathbb{x} = \boldsymbol{p}, \mathbb{y} = [\![\tilde{w}]\!], \mathbb{w} = \boldsymbol{w}),$$

where $\sigma : \{0,1\}^{\log\mu+\log n} \to \{0,1\}^{\log\mu+\log n}$ *is a permutation, and* $\tilde{q} \in \mathcal{F}^{(<2)}_{\log\mu+\log s}$, $\tilde{p} \in \mathcal{F}^{(<2)}_{\log\mu+\log m}$*, and* $\tilde{w} \in \mathcal{F}^{(<2)}_{\log\mu+\log n}$ *are the multilinear extensions of* $\boldsymbol{q}$*,* $\boldsymbol{p}$*, and* $\boldsymbol{w}$*, such that:*

- *Gate identity:* $\tilde{f}(\boldsymbol{x}) = 0$ *for all* $\boldsymbol{x} \in \{0,1\}^{\log\mu}$*, where*

$$\tilde{f}(\boldsymbol{X}) \coloneqq f(\{\tilde{q}(\mathsf{Bits}(j), \boldsymbol{X})\}_{j=0}^{s-1}, \{\tilde{w}(\mathsf{Bits}(j), \boldsymbol{X})\}_{j=0}^{n-1}).$$

- *Wiring identity:* $\tilde{w}(\boldsymbol{x}) = \tilde{w}(\sigma(\boldsymbol{x}))$ *for all* $\boldsymbol{x} \in \{0,1\}^{\log\mu+\log n}$*.*
- *Instance consistency:* $\tilde{p}(\boldsymbol{x}) = \tilde{w}(\boldsymbol{0}^{\log\mu+\log n-\log m}, \boldsymbol{x})$ *for all* $\boldsymbol{x} \in \{0,1\}^{\log m}$*.*

Linear Time Accumulation Scheme. Section 6.1 described instantiations of the multi-instance accumulation scheme with homomorphic PCS. Here we describe an instantiation using a code-based PCS to target linear-time proving and plausible post-quantum security. We first recall notions that are formalized in Appendix D.1.

- A linear code is an injective linear map $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ with message length k and codeword length n.
- The rate of $\mathcal{C}$ is $\rho \coloneqq k/n$.
- For $\boldsymbol{u} \in \mathbb{F}^n$, the list decoding set within distance δ is denoted $\Lambda(\mathcal{C}, \boldsymbol{u}, \delta)$.
- Out-of-domain sampling lets a verifier sample a point $\boldsymbol{\tau} \in \mathbb{F}^{\log n} \setminus \{0,1\}^{\log n}$ and obtain $\tilde{u}(\boldsymbol{\tau})$, where $\tilde{u}$ is the multilinear extension of $\boldsymbol{u}$.

To instantiate Quasar, the key is to implement the oracle batching protocol $\mathsf{NIR}_{\mathsf{batch}}$ from Definition 2. Bünz *et al.* [17] give a practical codeword batching protocol for any linear code, which can be compiled with Merkle commitments via the BCS transform. At a high level, it reduces two proximity claims (that $\boldsymbol{u}_0, \boldsymbol{u}_1$ are δ-close to $\mathcal{C}$) to one proximity claim (that $\boldsymbol{u}$ is δ-close to $\mathcal{C}$), together with an evaluation consistency constraint of the form $\gamma_0 \cdot \tilde{u}_0(\boldsymbol{x}) + \gamma_1 \cdot \tilde{u}_1(\boldsymbol{x}) = \tilde{u}(\boldsymbol{x})$.

For our batching relation $\mathcal{R}_0$ in Definition 2, the evaluation claims are phrased over multilinear polynomials $\tilde{f}_0, \tilde{f}_1$. In contrast, codeword batching operates on codewords $\boldsymbol{u}_i \coloneqq \mathcal{C}(\boldsymbol{f}_i)$ and their multilinear extensions. This mismatch is resolved by assuming that $\mathcal{C}$ is *systematic*, *i.e.*, the first k symbols of $\mathcal{C}(\boldsymbol{x})$ equal $\boldsymbol{x}$.

Assume that $1/\rho$ is a power of two. View a codeword $\boldsymbol{u}_i \in \mathbb{F}^n$ as $1/\rho$ consecutive blocks of length k. Let $\boldsymbol{Y} \in \{0,1\}^{\log(1/\rho)}$ select the block and let $\boldsymbol{X} \in \{0,1\}^{\log k}$ select the index within the block. Then the multilinear extension $\tilde{u}_i(\boldsymbol{Y}, \boldsymbol{X})$ satisfies $\tilde{u}_i(\boldsymbol{0}, \boldsymbol{X}) = \tilde{f}_i(\boldsymbol{X})$ by systematicity. Hence an evaluation claim $\tilde{f}_i(\boldsymbol{x}) = v_i$ can be lifted to the codeword claim $\tilde{u}_i(\boldsymbol{0}, \boldsymbol{x}) = v_i$.

Accordingly, we encode the relation $\mathcal{R}_0$ as the following relation over codeword oracles:

$$\mathcal{R}_0' \coloneqq \left\{ \begin{array}{c} \mathtt{i},\ \mathtt{x} = (\boldsymbol{\alpha} \coloneqq (\mathbf{0}, \boldsymbol{x}), v, r), \\ \mathtt{y} = ([\![\tilde{u}_0]\!], [\![\tilde{u}_1]\!]),\ \mathtt{w} = (\boldsymbol{u}_0, \boldsymbol{u}_1) \end{array} : \sum_{j=0}^{1} \tilde{eq}_j(r) \cdot \tilde{u}_j(\boldsymbol{\alpha}) = v \right\},$$

where $\boldsymbol{u}_j = \mathcal{C}(\boldsymbol{f}_j)$. This lets us instantiate $\mathsf{NIR}_{\mathsf{batch}}$ via codeword batching [17], and then plug it into Quasar's accumulation scheme.

Next, we state the security guarantee for the resulting $\mathsf{NIR}_{\mathsf{batch}}$ and give the corresponding protocol.

Theorem 4. *The reduction* $\mathsf{NIR}_{\mathsf{batch}}$ *based on linear codes has round-by-round knowledge soundness error at most* $2^{-\lambda}$ *for every* $\delta \in (0,1)$*, if the field* $\mathbb{F}$ *and repetition parameters* t, s *satisfy*

$$|\mathbb{F}| \geq 2^{\lambda/s-1} \cdot |L(\mathcal{C}, \delta)|^{2/s} \ \wedge\ t \geq \frac{\lambda}{-\log(1-\delta)}.$$

B Related Work

Recursive compositions. Recursive composition enables proving statements whose verification includes checking a prior proof, formalized by Bitansky *et al.* [10]. A recursive composition must preserve knowledge soundness, so that valid recursive proofs certify an efficiently computable witness at each step [10,29,28]. It must also keep the per-step verifier work small (ideally sublinear in the statement size), since otherwise the cost of verifying the composed computation grows prohibitively with the recursion depth.

Halo-style accumulation. Halo introduced accumulation schemes and sparked a line of practical recursive proof systems [14]. At a high level, these schemes avoid verifying a full SNARK/NARK proof at every recursive step by maintaining a *running instance* that represents deferred verification obligations [20,12,18]. The prover updates this running instance each step, and the recursion checks only the local update rather than re-checking all prior proofs. A single final verification then discharges the accumulated obligations, yielding end-to-end verification comparable to a single step.

Nova-style folding. Nova introduced folding schemes as an alternative accumulation paradigm for IVC [42]. Rather than accumulating already-compiled SNARK/NARK proofs (as in Halo-style approaches, often instantiated via PCS such as KZG [40] or Bulletproofs [15]), Nova folds underlying constraint instances using (typically) random linear combinations. This shifts work from verifying cryptographic proof objects to verifying lightweight consistency checks of folded instances, reducing the dominant overhead inside the recursion. Protostar [16] and Protogalaxy [30] further extend and systematize folding-based IVC, including variants based on special-sound protocols [3].

Hash-based accumulation. Recent work develops accumulation schemes whose dominant verification reduces to hash computations rather than PCS checks. A general construction for error-correcting codes achieves accumulation but is not distance-preserving, with relative distance degrading from δ to $\delta - \epsilon$, and it supports only bounded-depth accumulation [21]. A subsequent construction gives a distance-preserving accumulation scheme for Reed–Solomon codes [22]. This approach is later extended to (classes of) locally decodable codes, yielding a linear-time accumulation prover [17].

Holographic proofs and preprocessing SNARKs. A recurring goal in SNARK design is to reduce online verification cost. In preprocessing SNARKs, the verifier performs an offline phase that preprocesses the circuit into a succinct summary, and then uses this summary to verify many instances efficiently in an online phase. This paradigm underlies several practical systems, including Sonic [48], PLONK [33], and Spartan [52]. Marlin [27] connects preprocessing SNARKs to the information-theoretic notion of holographic proofs [4], in which the verifier queries an encoding of the input rather than reading it in full. This viewpoint clarifies the role of preprocessing and query access in achieving succinct verification.

C Implementations and Evaluations

We evaluate Quasar by measuring the cost of *accumulation verification logic* embedded in a recursive circuit. We report circuit constraint counts, native prover running time, and peak memory usage for the accumulation prover under varying arity. We do *not* report end-to-end recursive IVC performance, because our prototype does not yet include a recursive wrapper (*e.g.*, HyperNova) together with an in-circuit verifier for the code-based PCS.

All experiments are conducted on a MacBook Air (2025 model) running macOS Sequoia 15.6, equipped with an Apple M4 chip (8 cores) and 16 GB RAM. The codebase is compiled in `--release` mode using `rustc` version 1.92.0 with default optimization flags.

For the curve-based polynomial commitment scheme (PCS), we use Mercury PCS [31] instantiated over BN256. For the code-based PCS, we use FRI [5] with Merkle commitments instantiated via Poseidon [36]. Our implementation is structured to admit future linear-time PCS backends, such as Brakedown [35], as drop-in replacements for the PCS layer.

Due to missing toolchain components, notably (i) a recursive construction compatible with our code-based backend and (ii) an in-circuit encoding of the FRI verifier, we leave a full multi-instance IVC prototype as future work. In this paper, we therefore focus on simulating the accumulation verification logic within the circuit and quantifying its constraint footprint. To estimate constraint counts, we use the `bellpepper` crate[5] for circuit synthesis and constraint system instrumentation.

[5] `https://github.com/lurk-lang/bellpepper.git`

Table 4: Circuit size comparison under different arity and degree (ℓ: arity, d: degree).

Arity (ℓ)	**HyperNova** $d=2$	$d=4$	$d=8$	**Quasar** $d=2$	$d=4$	$d=8$
2	12,533	12,539	12,552	12,143	12,149	12,161
4	15,667	15,677	15,697	13,229	13,239	13,259
8	21,629	21,643	21,671	14,315	14,329	14,357
16	32,467	32,485	32,521	15,401	15,419	15,455
32	53,057	53,079	53,123	16,487	16,509	16,553
64	93,151	93,177	93,229	17,573	17,599	17,651

C.1 Parameter Settings and Experimental Results

Unless stated otherwise, we set the security parameter to $\lambda = 128$. We fix the verifier polynomial degree parameter to $d = 2$ and vary the accumulation arity as $\ell \in \{2, 4, 8, 16, 32, 64\}$. For code-based experiments, we use a Reed–Solomon code with message length $k = 2^8$ and rate $\rho = 1/16$, hence codeword length $n = k/\rho = 2^{12}$.

We evaluate prover performance under different arity settings, including: (i) the constraint count of the recursive circuit for accumulation verification, (ii) the running time of the accumulation prover (in seconds), and (iii) the peak memory consumption of the accumulation prover (in MB). As a representative workload, we use a synthetic predicate circuit that incrementally computes a hash chain.

Figure 12a reports results for the curve-based PCS instantiation, comparing Quasar against HyperNova under identical arity and degree settings. Across all arities, Quasar yields lower constraint counts, lower prover time, and lower peak memory in our prototype. The dominant source of savings is the reduced number of in-circuit CRC operations. Concretely, for $\ell \geq 16$, CRC accounts for more than half of the total accumulation-verification constraints in our HyperNova baseline, while it remains below one quarter in Quasar.

Figure 12b reports results for the code-based PCS instantiation, comparing Quasar against Arc under identical arity and degree settings. In our measurements, Arc incurs a 3–5× higher prover cost than Quasar. This gap is primarily driven by the overhead of accumulation in code-based settings, where batching and proximity checks dominate the recursive circuit. We view Quasar as a step toward making accumulation practical for code-based recursive proof systems.

To study scaling with both arity and degree, Table 4 reports constraint counts for HyperNova and Quasar across $\ell \in \{2, 4, 8, 16, 32, 64\}$ and $d \in \{2, 4, 8\}$.

D Formal Definitions

D.1 Linear Codes

Definition 8 (Linear code). *A* code *over an alphabet Σ with message length k and codeword length n is an injective map $\mathcal{C}\colon \Sigma^k \to \Sigma^n$. A code $\mathcal{C}$ is a* linear

code *if* $\Sigma = \mathbb{F}$ *is a field and* $\mathcal{C}$ *is* $\mathbb{F}$*-linear. In this paper, we assume* $n \in 2^{\mathbb{N}}$*. The* minimum distance *of* $\mathcal{C}$ *is defined as* $\delta(\mathcal{C}) \coloneqq \min_{u \neq v \in \mathrm{Im}(\mathcal{C})} \Delta(u, v)$.

We often identify $\mathcal{C}$ with its image $\mathrm{Im}(\mathcal{C}) \subseteq \Sigma^n$. We view a codeword $u \in \mathrm{Im}(\mathcal{C})$ as a function $u \colon [n] \to \Sigma$.

Definition 9 (Zero-evaders). *A* zero-evader *over* $\mathbb{F}$ *with error* ϵ_{zero} *is a function* $\mathsf{ZE} \colon D_{\mathsf{ZE}} \to \mathbb{F}^m$ *such that*

$$\forall \boldsymbol{v} \in \mathbb{F}^m \setminus \{\mathbf{0}\}, \quad \Pr_{\rho \leftarrow D_{\mathsf{ZE}}} \left[\langle \mathsf{ZE}(\rho), \boldsymbol{v} \rangle = 0 \right] \leq \epsilon_{\mathsf{zero}}. \tag{13}$$

Moreover, let $G \in \mathbb{F}^{n \times k}$ *be an injective* $\mathbb{F}$*-linear map. Let* $\tilde{G}$ *denote the multilinear extension of* G*. Define* $\mathsf{ZE} \colon \mathbb{F}^{\log n} \to \mathbb{F}^k$ *by*

$$\mathsf{ZE}(\boldsymbol{\alpha}) \coloneqq \left(\tilde{G}(\boldsymbol{\alpha}, \boldsymbol{b}) \right)_{\boldsymbol{b} \in \{0,1\}^{\log k}}.$$

Then ZE *is a zero-evader with error at most* $\frac{\log n}{|\mathbb{F}|}$.

Definition 10 (Out-of-domain samples). *Let* $\mathcal{C} \colon \mathbb{F}^k \to \mathbb{F}^n$ *be a linear code. Let* $f \colon [n] \to \mathbb{F}$ *be a received word. Let* $s \in \mathbb{N}$ *be a repetition parameter and let* $\delta \in [0, 1]$ *be a distance parameter. Let* $\mathsf{ZE} \colon D_{\mathsf{ZE}} \to \mathbb{F}^k$ *be a zero-evader with error* ϵ_{zero}*. Let* $\Lambda(\mathcal{C}, f, \delta)$ *denote the list of codewords in* $\mathrm{Im}(\mathcal{C})$ *within distance* δ *of* f*. Let* $L(\mathcal{C}, \delta) \coloneqq \max_f |\Lambda(\mathcal{C}, f, \delta)|$*. Then*

$$\Pr_{\rho_1, \ldots, \rho_s \leftarrow\$ D_{\mathsf{ZE}}} \left[\begin{array}{c} \exists \text{ distinct } u, v \in \Lambda(\mathcal{C}, f, \delta) \text{ s.t.} \\ \forall i \in [s],\ \langle \mathsf{ZE}(\rho_i), \mathcal{C}^{-1}(u) \rangle = \langle \mathsf{ZE}(\rho_i), \mathcal{C}^{-1}(v) \rangle \end{array} \right] \leq \frac{L(\mathcal{C}, \delta)^2}{2} \cdot \epsilon_{\mathsf{zero}}^s.$$

D.2 Sum-check Protocol

We write $\langle \mathcal{P}(\tilde{f}), \mathcal{V}(r) \rangle(m, d, \mathsf{sum}) \to b$ for the interactive sum-check protocol between prover $\mathcal{P}$ and verifier $\mathcal{V}$. The protocol outputs $b = 1$ if the claim $\sum_{\boldsymbol{x} \in \{0,1\}^m} \tilde{f}(\boldsymbol{x}) = \mathsf{sum}$ holds. Here $\tilde{f}$ is an m-variate polynomial with individual degree at most d.

Definition 11 (Completeness). *If* $\mathsf{sum} = \sum_{\boldsymbol{x} \in \{0,1\}^m} \tilde{f}(\boldsymbol{x})$*, then for an honest prover* $\mathcal{P}$ *and for all verifier randomness strings* $r \in \{0, 1\}^*$*,*

$$\Pr \left[\langle \mathcal{P}(\tilde{f}), \mathcal{V}(r) \rangle(m, d, \mathsf{sum}) = 1 \right] = 1.$$

Definition 12 (Soundness). *If* $\mathsf{sum} \neq \sum_{\boldsymbol{x} \in \{0,1\}^m} \tilde{f}(\boldsymbol{x})$*, then for any (unbounded) prover* $\mathcal{P}^*$ *and for all verifier randomness strings* $r \in \{0, 1\}^*$*,*

$$\Pr \left[\langle \mathcal{P}^*(\tilde{f}), \mathcal{V}(r) \rangle(m, d, \mathsf{sum}) = 1 \right] \leq m \cdot d / |\mathbb{F}|.$$

D.3 Polynomial Commitment Scheme

Definition 13 (Multilinear polynomial commitment scheme [57]). *A multilinear polynomial commitment scheme* PCS *consists of a tuple of efficient algorithms* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Eval}, \mathsf{Verify})$*:*

- $\mathsf{Setup}(1^\lambda, n) \to \mathsf{ck}$ *takes security parameter* λ *and variable count* $n \in \mathbb{N}$*, and outputs a commitment key for polynomials in* $\mathcal{F}_n^{(<2)}$*.*
- $\mathsf{Commit}(\mathsf{ck}, p(\boldsymbol{X})) \to C$ *takes* $p(\boldsymbol{X}) \in \mathcal{F}_n^{(<2)}$ *and outputs a commitment* C*.*
- $\mathsf{Eval}(\mathcal{P}(p(\boldsymbol{X})), \mathcal{V}(\mathsf{ck}, C, \boldsymbol{z}, y))$ *is a protocol between* $\mathcal{P}$ *and* $\mathcal{V}$ *with public inputs* $(\mathsf{ck}, C, \boldsymbol{z}, y)$*, where* $\boldsymbol{z} \in \mathbb{F}^n$ *and* $y \in \mathbb{F}$*. The prover additionally knows* $p(\boldsymbol{X}) \in \mathcal{F}_n^{(<2)}$ *and aims to convince the verifier that* C *commits to* p *and that* $p(\boldsymbol{z}) = y$*. The verifier outputs a bit* b*.*
- $\mathsf{Verify}(\mathsf{ck}, C, \boldsymbol{z}, y, \pi) \to b$ *denotes the (possibly implicit) verifier algorithm for the protocol* Eval*, where* π *denotes the transcript provided by the prover in the non-interactive (Fiat–Shamir) form.*

We require PCS to satisfy completeness, binding, and knowledge soundness.

Definition 14 (Completeness). *The scheme* PCS *satisfies* completeness *if for any* $n \in \mathbb{N}$*, any* $p(\boldsymbol{X}) \in \mathcal{F}_n^{(<2)}$*, and any* $\boldsymbol{z} \in \mathbb{F}^n$ *with* $y \coloneqq p(\boldsymbol{z})$*,*

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{ck}, C, \boldsymbol{z}, y, \pi) \\ = 1\end{array}\middle|\begin{array}{r}\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, n) \\ C \leftarrow \mathsf{Commit}(\mathsf{ck}, p(\boldsymbol{X})) \\ \pi \leftarrow \textit{Honest transcript of } \mathsf{Eval}(\mathcal{P}(p(\boldsymbol{X})), \mathcal{V}(\mathsf{ck}, C, \boldsymbol{z}, y))\end{array}\right] = 1.$$

Definition 15 (Binding). *The scheme* PCS *is (computationally)* binding *if for any* $n \in \mathbb{N}$ *and any PPT adversary* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{c}y_0 \neq y_1 \\ \wedge\ \mathsf{Verify}(\mathsf{ck}, C, \boldsymbol{z}, y_0, \pi_0) = 1 \\ \wedge\ \mathsf{Verify}(\mathsf{ck}, C, \boldsymbol{z}, y_1, \pi_1) = 1\end{array}\middle|\begin{array}{r}\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, n) \\ (C, \boldsymbol{z}, y_0, y_1, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{ck})\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Definition 16 (Knowledge soundness). PCS *satisfies* knowledge soundness *if* Eval *is an argument of knowledge for the relation*

$$\mathcal{R}_{\mathsf{eval}}(\mathsf{ck}) \coloneqq \{(C, \boldsymbol{z}, y; p(\boldsymbol{X})) \colon p(\boldsymbol{z}) = y \wedge C = \mathsf{Commit}(\mathsf{ck}, p(\boldsymbol{X}))\},$$

where $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, n)$*. That is, for any* $n \in \mathbb{N}$ *and any PPT adversaries* $\mathcal{A}$ *and* $\tilde{\mathcal{P}}$*, there exists an expected polynomial-time extractor* $\mathcal{E}$ *such that for any auxiliary randomness* r*,*

$$\Pr\left[\begin{array}{c}\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, n) \\ (C, \boldsymbol{z}, y) \leftarrow \mathcal{A}(\mathsf{ck}, r) \\ \mathsf{Verify}(\mathsf{ck}, C, \boldsymbol{z}, y, \pi) = 1 \textit{ for } \pi \leftarrow \tilde{\mathcal{P}}(r)\end{array}\right] \approx \Pr\left[\begin{array}{c}\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, n) \\ (C, \boldsymbol{z}, y) \leftarrow \mathcal{A}(\mathsf{ck}, r) \\ p(\boldsymbol{X}) \leftarrow \mathcal{E}(\mathsf{ck}, r) \\ (C, \boldsymbol{z}, y; p(\boldsymbol{X})) \in \mathcal{R}_{\mathsf{eval}}(\mathsf{ck})\end{array}\right].$$

D.4 Multi-Instance Accumulation Schemes

We adopt the definition of accumulation schemes for NARK from BCLMS21 [18]. We consider a non-interactive argument of knowledge $\mathsf{NARK} := (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ in the ROM for an indexed relation $\mathcal{R}$. We assume proofs have a canonical partition $\pi := (\pi.\mathbb{x}, \pi.\mathbb{w})$.

Completeness. NARK is complete if for every (unbounded) adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} (\mathbb{i}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_k) \in \mathcal{R}^*(\mathsf{pp}) \\ \Rightarrow\ \mathcal{V}(\mathsf{vk}, \{\mathbb{x}^{(k)}\}_k, \pi) = 1 \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (\mathbb{i}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_k) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{I}(\mathsf{pp}, \mathbb{i}) \\ \pi \leftarrow \mathcal{P}(\mathsf{pk}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_k) \end{array} \right] = 1.$$

Knowledge soundness. NARK is knowledge sound with respect to auxiliary input distribution χ if for every (non-uniform) polynomial-time malicious prover $\tilde{\mathcal{P}}$ there exists a deterministic polynomial-time extractor $\mathcal{E}$ such that

$$\Pr\left[\begin{array}{c} \mathcal{V}(\mathsf{vk}, \{\mathbb{x}^{(k)}\}_k, \pi) = 1 \\ \wedge\ (\mathbb{i}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_k) \notin \mathcal{R}^*(\mathsf{pp}) \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \chi(1^\lambda) \\ (\mathbb{i}, \{\mathbb{x}^{(k)}\}_k, \pi), \mathsf{tr} \leftarrow \tilde{\mathcal{P}}(\mathsf{pp}, \mathsf{ai}) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{I}(\mathsf{pp}, \mathbb{i}) \\ \{\mathbb{w}^{(k)}\}_k \leftarrow \mathcal{E}(\mathsf{pp}, \mathbb{i}, \{\mathbb{x}^{(k)}\}_k, \pi, \mathsf{ai}, \mathsf{tr}) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

Definition 17 (Multi-instance accumulation schemes). *A multi-instance accumulation scheme for* NARK *is a tuple* $\mathsf{ACC} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V}, \mathcal{D})$ *with* $\mathcal{G}$ *the same as* NARK.

- $\mathsf{ACC}.\mathcal{G}(1^\lambda) \to \mathsf{pp}$ *samples and outputs public parameters* pp.
- $\mathsf{ACC}.\mathcal{I}(\mathsf{pp}, \mathbb{i}) \to (\mathsf{apk}, \mathsf{avk}, \mathsf{adk})$ *deterministically outputs accumulator keys and a decision key.*
- $\mathsf{ACC}.\mathcal{P}(\mathsf{apk}, \{\mathbb{x}^{(k)}\}_k, \pi, \mathsf{acc}) \to (\mathsf{acc}', \mathsf{pf})$ *takes multiple instances* $\{\mathbb{x}^{(k)}\}_k$*, a proof* $\pi = (\pi.\mathbb{x}, \pi.\mathbb{w})$*, and an old accumulator* $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$*, and outputs a new accumulator* $\mathsf{acc}' = (\mathsf{acc}'.\mathbb{x}, \mathsf{acc}'.\mathbb{w})$ *and an accumulation proof* pf.
- $\mathsf{ACC}.\mathcal{V}(\mathsf{avk}, \{\mathbb{x}^{(k)}\}_k, \pi.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}'.\mathbb{x}, \mathsf{pf}) \to b$ *outputs accept/reject.*
- $\mathsf{ACC}.\mathcal{D}(\mathsf{adk}, \mathsf{acc}) \to b$ *takes* adk *and an accumulator* acc *and outputs whether* acc *is valid for the corresponding index.*

Completeness. For every (unbounded) adversary $\mathcal{A}$, the probability $\Pr[E_1 \mid E_2]$ equals 1, where E_1, E_2 are events defined as:

$$E_1 := \left\{ \begin{array}{c} \mathsf{NARK}.\mathcal{V}(\mathsf{vk}, \{\mathbb{x}^{(k)}\}_k, \pi) = 1 \ \wedge\ \mathsf{ACC}.\mathcal{D}(\mathsf{adk}, \mathsf{acc}) = 1 \\ \Rightarrow\ \mathsf{ACC}.\mathcal{V}(\mathsf{avk}, \{\mathbb{x}^{(k)}\}_k, \pi.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}'.\mathbb{x}, \mathsf{pf}) = 1 \ \wedge\ \mathsf{ACC}.\mathcal{D}(\mathsf{adk}, \mathsf{acc}') = 1 \end{array} \right\},$$

$$E_2 := \left\{ \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{ACC}.\mathcal{G}(1^\lambda) \\ (\mathbb{i}, \{\mathbb{x}^{(k)}\}_k, \pi, \mathsf{acc}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{NARK}.\mathcal{I}(\mathsf{pp}, \mathbb{i}) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{adk}) \leftarrow \mathsf{ACC}.\mathcal{I}(\mathsf{pp}, \mathbb{i}) \\ (\mathsf{acc}', \mathsf{pf}) \leftarrow \mathsf{ACC}.\mathcal{P}(\mathsf{apk}, \{\mathbb{x}^{(k)}\}_k, \pi, \mathsf{acc}) \end{array} \right\}.$$

Knowledge soundness. ACC is knowledge sound with respect to auxiliary input distribution χ if for every non-uniform polynomial-time malicious prover $\tilde{\mathcal{P}}$ there exists a polynomial-time extractor $\mathcal{E}$ such that the probability $\Pr[E_3 \mid E_4]$ is negligibly close to 1, where E_3, E_4 are events defined as:

$$E_3 := \left\{ \begin{array}{r} \mathsf{ACC}.\mathcal{V}(\mathsf{avk}, \{\mathbb{x}^{(k)}\}_k, \pi.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}'.\mathbb{x}, \mathsf{pf}) = 1 \ \wedge \ \mathsf{ACC}.\mathcal{D}(\mathsf{adk}, \mathsf{acc}') = 1 \\ \Rightarrow \ \mathcal{E} \text{ outputs witnesses consistent with } \pi \text{ and } \mathsf{acc} \end{array} \right\},$$

$$E_4 := \left\{ \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{ACC}.\mathcal{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \chi(1^\lambda) \\ (\mathbb{i}, \{\mathbb{x}^{(k)}\}_k, \pi.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}', \mathsf{pf}; \mathsf{tr}) \leftarrow \tilde{\mathcal{P}}(\mathsf{apk}, \mathsf{ai}) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{NARK}.\mathcal{I}(\mathsf{pp}, \mathbb{i}) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{adk}) \leftarrow \mathsf{ACC}.\mathcal{I}(\mathsf{pp}, \mathbb{i}) \\ (\pi.\mathbb{w}, \mathsf{acc}.\mathbb{w}) \leftarrow \mathcal{E}(\mathsf{pp}, \mathbb{i}, \{\mathbb{x}^{(k)}\}_k, \pi.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}', \mathsf{pf}; \mathsf{ai}, \mathsf{tr}) \end{array} \right\}.$$

D.5 Definitions for Interactive Oracle Reductions

Definition 18 (Completeness). *An IOR is complete if for any* $(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathbb{w}) \in \mathcal{R}$*,*

$$\Pr_{\{r_i\}_{i\in[\mu]}} \left[(\mathbb{i}', \mathbb{x}', \mathbb{y}', \mathbb{w}') \in \mathcal{R}' \middle| \begin{array}{r} (\iota, \mathbb{I}, \mathbb{i}') \leftarrow \mathcal{I}(\mathbb{i}) \\ (\mathbb{x}', \mathbb{y}'; \mathbb{w}') \leftarrow \langle \mathcal{P}(\mathbb{x}, \mathbb{y}, \mathbb{w}), \mathcal{V}^{\mathbb{I}, \mathbb{y}}(\mathbb{x}) \rangle \end{array} \right] = 1.$$

Definition 19 (Soundness). *For every proximity bound* $\delta \in (0, \delta^*)$*, every statement* $(\mathbb{i}, \mathbb{x}, \mathbb{y}) \notin \mathcal{L}(\mathcal{R})$ *with* $\Delta_{\mathcal{R}}(\mathbb{i}, \mathbb{x}, \mathbb{y}) > \delta$*, and every (unbounded) prover* $\mathcal{P}^*$*,*

$$\Pr_{\{r_i\}_{i\in[\mu]}} \left[\Delta_{\mathcal{R}'}(\mathbb{i}', \mathbb{x}', \mathbb{y}') \leq \delta \middle| \begin{array}{r} (\iota, \mathbb{I}, \mathbb{i}') \leftarrow \mathcal{I}(\mathbb{i}) \\ (\mathbb{x}', \mathbb{y}'; \mathbb{w}') \leftarrow \langle \mathcal{P}^*, \mathcal{V}^{\mathbb{I}, \mathbb{y}}(\mathbb{x}) \rangle \end{array} \right] \leq \epsilon(\mathbb{i}, \mathbb{x}, \mathbb{y}, \delta).$$

We omit δ when there is no proximity gap, *i.e.*, when $\delta = \delta^* = 0$.

Definition 20 (Knowledge state function for IOR [17]). *Let* $\mathsf{IOR} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ *be a* μ*-round* IOR *from* $\mathcal{R}$ *to* $\mathcal{R}'$ *with proximity radius* δ^**. A* knowledge state function *for* IOR *is a (possibly inefficient) function* State*, parameterized by* $\delta \in [0, \delta^*)$*, that on input* $(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbb{w})$ *outputs a bit and satisfies:*

- ***Empty transcript:*** *If* $\mathsf{tr} = \emptyset$*, then* $\mathsf{State}_\delta(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbb{w}) = 1$ *iff there exists* $\mathbb{y}_*$ *such that* $(\mathbb{i}, \mathbb{x}, \mathbb{y}_*, \mathbb{w}) \in \mathcal{R}$ *and* $\Delta(\mathbb{y}, \mathbb{y}_*) \leq \delta$*.*
- ***Prover moves:*** *If the prover is about to move and* $\mathsf{State}_\delta(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbb{w}) = 0$*, then for every prover message* π *we have* $\mathsf{State}_\delta(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \| \pi, \mathbb{w}) = 0$*.*
- ***Full transcript:*** *If* $\mathsf{tr} = (\pi_1, r_1, \ldots, \pi_\mu, r_\mu)$ *is full, then* $\mathsf{State}_\delta(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbb{w}) = 1$ *iff the verifier outputs* $(\mathbb{i}', \mathbb{x}', \mathbb{y}')$ *such that there exists* $\mathbb{y}'_*$ *with* $(\mathbb{i}', \mathbb{x}', \mathbb{y}'_*, \mathbb{w}) \in \mathcal{R}'$ *and* $\Delta(\mathbb{y}', \mathbb{y}'_*) \leq \delta$*.*

Definition 21 (A variant of round-by-round knowledge soundness [17]). *A μ-round* $\mathsf{IOR} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ *from* $\mathcal{R}$ *to* $\mathcal{R}'$ *with proximity radius* δ^* *has RBR knowledge soundness errors* $(\epsilon_1, \ldots, \epsilon_\mu)$ *and extraction times* $(\mathsf{et}_1, \ldots, \mathsf{et}_\mu)$ *if there exist a knowledge state function* State *and a deterministic extractor* $\mathcal{E}$ *such that for every* $\delta \in (0, \delta^*)$*, every* $(\mathbb{i}, \mathbb{x}, \mathbb{y})$*, every round index* $i \in [\mu]$*, and every partial transcript* $\mathsf{tr} = (\pi_1, r_1, \ldots, \pi_{i-1}, r_{i-1}, \pi_i)$ *where the verifier is about to sample* r_i*, the extractor runs in time at most* et_i *and*

$$\Pr\left[\exists \mathbb{w} \colon \mathsf{State}_\delta(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathcal{E}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \| r_i, \mathbb{w})) = 0 \ \wedge \ \mathsf{State}_\delta(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \| r_i, \mathbb{w}) = 1\right] \leq \epsilon_i(\mathbb{i}, \mathbb{x}, \mathbb{y}, \delta).$$

Bünz *et al.* [17] show that RBR knowledge soundness implies straightline state-restoration knowledge soundness, which suffices for Fiat–Shamir security.

Theorem 5 (Polynomial IOR compilation adopted from [24]). *There is a transformation* T *from a public-coin polynomial interactive oracle reduction* IOR *to a non-interactive reduction* NIR *with compilation in two steps:*

- *replace every oracle in* IOR *with a polynomial commitment* PCS *to obtain* $\mathsf{IOR}^{\mathsf{PCS}}$*;*
- *apply the Fiat–Shamir transform to* $\mathsf{IOR}^{\mathsf{PCS}}$ *to obtain* $\mathsf{NIR} \coloneqq \mathsf{FS}[\mathsf{IOR}^{\mathsf{PCS}}]$.

If IOR *is RBR knowledge sound and* PCS *is extractable, then* NIR *is RBR knowledge sound in the ROM. The prover time, verifier time, and proof size of* NIR *are obtained by summing the corresponding costs from* IOR *and from the PCS compilation (commitments and openings) according to the message and query complexity of* IOR*.*

D.6 Definitions for Special-Sound Protocols

For an $\mathcal{NP}$ relation $\mathcal{R} \coloneqq \{(\mathbb{x}; \mathbb{w})\}$, an interactive proof for $\mathcal{R}$ allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that $\mathbb{x}$ admits a witness $\mathbb{w}$. In a $(2\mu - 1)$-move protocol, the prover and verifier exchange μ prover messages and $\mu - 1$ verifier challenges. If $\mathcal{V}$ samples each challenge uniformly at random and independently of the prover's messages, the protocol is *public coin*. Public-coin protocols can be transformed into non-interactive protocols via Fiat–Shamir [32].

Definition 22 (Tree of transcripts). *Let* $k \in \mathbb{N}$ *and* $(a_1, \ldots, a_k) \in \mathbb{N}^k$*. An* $(a_1, \ldots, a_k)$*-tree of transcripts is a set of* $\prod_{i=1}^{k} a_i$ *transcripts organized as a rooted tree. Edges correspond to verifier challenges. Vertices correspond to prover messages (possibly empty). Each node at depth* i *has* a_i *children, corresponding to* a_i *distinct challenges. Each transcript corresponds to a unique root-to-leaf path.*

Definition 23 ($(a_1, \ldots, a_k)$-special soundness). *A protocol* Π *provides* $(a_1, \ldots, a_k)$*-special soundness if there exists a PPT extractor* $\mathcal{E}$ *that, given* $(\mathbb{x}, T)$ *for any* $(a_1, \ldots, a_k)$*-tree of accepting transcripts* T*, outputs* $\mathbb{w}$ *such that* $(\mathbb{x}; \mathbb{w}) \in \mathcal{R}$*.*

Lemma 6 ($(a_1, \ldots, a_k)$-special soundness vs. knowledge soundness [3]). *Let $k, a_1, \ldots, a_k \in \mathbb{N}$ and let $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ be a $(a_1, \ldots, a_k)$-special sound $(2k+1)$-move public-coin protocol for relation $\mathcal{R}$, where each challenge is sampled uniformly from $\mathbb{F}$. Then $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is knowledge sound with knowledge error*

$$\kappa \leq \frac{\sum_{i=1}^{k}(a_i - 1)}{|\mathbb{F}|}.$$

E Complementary Protocols

E.1 Non-Interactive Multi-Cast Reduction for SPS

Let $\mathsf{pp} \leftarrow \mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{G}(1^\lambda)$ and $(\mathsf{pk}, \mathsf{vk}, \mathbb{i}') \leftarrow \mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{I}(\mathsf{pp}, \mathbb{i})$. We define the non-interactive prover and verifier algorithms in Figure 13 and 14 for the non-interactive reduction $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}} \coloneqq (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ defined in Section 4.1, where the hash function Hash will be modeled by the RO in the security analysis. We denote $\boldsymbol{r}_F \coloneqq (\boldsymbol{r} \| \boldsymbol{\alpha})$ for short.

E.2 Non-Interactive 2-to-1 Reduction

Let $\mathsf{pp} \leftarrow \mathsf{NIR}_{\mathsf{fold\text{-}SPS}}.\mathcal{G}(1^\lambda)$ and $(\mathsf{pk}, \mathsf{vk}, \mathbb{i}') \leftarrow \mathsf{NIR}_{\mathsf{fold\text{-}SPS}}.\mathcal{I}(\mathsf{pp}, \mathbb{i})$. We define the non-interactive prover and verifier algorithms in Figure 15 and 16 for the non-interactive reduction $\mathsf{NIR}_{\mathsf{fold\text{-}SPS}} \coloneqq (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ defined in Section 5.3.

F Security Proofs

F.1 Proof of Lemma 5

Proof. To conduct the proof, we follow the roadmap of Arc and WARP [1,22]. The completeness holds trivially. For the RBR knowledge soundness, the protocol $\mathsf{IOR}_{\mathsf{fold}}$ has soundness error $\epsilon^{\mathsf{zc}}, \epsilon^{\mathsf{sc}}, \epsilon^{\mathsf{batch}}$ with extraction time $O(\ell^2 \cdot n^2)$, where:

$$\epsilon^{\mathsf{zc}} = (\mu+1)/|\mathbb{F}|, \epsilon^{\mathsf{sc}} = \max(d+1, \log(\ell n))/|\mathbb{F}|, \epsilon^{\mathsf{eval}} = \mu/|\mathbb{F}|, \epsilon^{\mathsf{batch}} = 2\mu \cdot \epsilon_0.$$

where ϵ_0 is the soundness error for the oracle batching protocol $\mathsf{IOR}_{\mathsf{batch}}$. We define the state function and prove bounds on the RBR knowledge soundness error as follows:

- State function for empty transcript. Given an input $\mathbb{x} = \{\mathbb{x}^{(k)}\}_{k\in[k]}, \mathbb{y} = \{\mathbb{y}^{(k)}\}_{k\in[k]}$, we set $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \emptyset) = 1$ if and only if $(\mathbb{x}^{(k)}, \mathbb{y}^{(k)}) \in \mathcal{L}(\mathcal{R}_F)$ for all $k \in [\ell]$.
- Bounding ϵ^{zc}. At this stage, the partial transcript is empty as $\mathsf{tr} = \emptyset$ and the verifier sends $\boldsymbol{\gamma} \in \mathbb{F}^{\log(\mu+1)}, r_z \in \mathbb{F}$. We set the state function $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}\|\boldsymbol{\gamma}, r_z) = 1$ if and only if the following hold:
  - $\sum_{z\in\{0,1\}} G(z) = 0$.

The extractor outputs the witness $\mathbb{w} = (\{\boldsymbol{m}^{(k)}_{\cup,i}\}_{i\in[\mu],k\in[1]}, \{\boldsymbol{m}^{(k)}_{i}\}_{i\in[\mu],k\in[1]})$ in time $O(\mu{\cdot}n^2)$, where $\boldsymbol{m}^{(k)}_{\cup,i}[j] = \tilde{m}^{(k)}_{\cup,i}(\mathsf{Bits}(j)), j \in [\ell n]$ queried from $[\![\tilde{m}_{\cup,i}]\!]$, and $\boldsymbol{m}^{(k)}_{i}[j] = \tilde{m}_i(\mathsf{Bits}(j)), j \in [n]$ queried from $[\![\tilde{m}_i]\!]$. For the knowledge error, we show that if the extracted witness is invalid, *i.e.*, $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathbb{w}, \mathsf{tr}) = 0$, then $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}\|\boldsymbol{\gamma}, \boldsymbol{r}_z) = 1$ holds with a negligible probability ϵ^{zc}. Concretely, we define $\{G^{(k)}(\boldsymbol{Z}_\gamma, Z_r)\}_{k\in[1]}$ as a multivariate polynomial below

$$\begin{aligned} G^{(k)}(\boldsymbol{Z}_\gamma, Z_r) = \tilde{eq}(Z_r, k) \cdot \big[&(F(\tilde{\boldsymbol{x}}(k), \{\boldsymbol{m}_i(k)\}_{i\in[\mu]}, \boldsymbol{r}_F(k)) - e(k)) \\ &+ \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{Z}_\gamma) \cdot (\tilde{m}_i(k, \boldsymbol{r}_x(k))) - v_i(k)) \\ &+ \sum_{i\in[\mu]} \mathsf{pow}_{i+\mu}(\boldsymbol{Z}_\gamma) \cdot (\tilde{m}\cup, i(k, \boldsymbol{\tau}(k), \boldsymbol{r}_x(k)) - v_i(k))\big]. \end{aligned} \tag{14}$$

Since $G^{(k)}(\boldsymbol{Z}_\gamma, Z_r)$ has total degree at most $\mu + 1$, then $\sum_{k\in[1]} G^{(k)}(\boldsymbol{Z}_\gamma, Z_r)$ equals to zero with probability $\epsilon^{\mathsf{zc}} \leq (\mu + 1)/|\mathbb{F}|$.

- Bounding ϵ^{sc}. For the sum-check protocol, the partial transcript has the form $\mathsf{tr} = (\boldsymbol{\gamma}, r_z, G(Z))$. We set the state function $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}\|\sigma) = 1$ if and only if the following hold:

$$\begin{aligned} G(\sigma) = \tilde{eq}(r_z, \sigma) \cdot \big[&(F(\tilde{\boldsymbol{x}}(\sigma), \{\boldsymbol{m}_i(\sigma)\}_{i\in[\mu]}, \boldsymbol{r}_F(\sigma)) - e(\sigma)) \\ &+ \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{\gamma}) \cdot (\tilde{m}_i(Z, \boldsymbol{r}_x(Z))) - v_i(Z)) \\ &+ \sum_{i\in[\mu]} \mathsf{pow}_{\mu+i}(\boldsymbol{\gamma}) \cdot (\tilde{m}_{\cup,i}(Z, \boldsymbol{\tau}(Z), \boldsymbol{r}_x(Z)) - v_i(Z))\big] \end{aligned} \tag{15}$$

  To bound the error, we prove that if $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}) = 0$, *i.e.*, $\sum_{z\in\{0,1\}} G(z) \neq 0$, then $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}\|\sigma) = 1$ holds with a negligible probability ϵ^{sc}. The only way the prover can convince the verifier is by sending a univariate polynomial $G'(Z)$ with maximum degree $\max(d+1, \log(\ell n))$ that does not equal the prescribed polynomial $G(Z)$ and yet $G'(\sigma) = G(\sigma)$, of which the probability is bounded by $\max(d + 1, \log(\ell n))/|\mathbb{F}|$.
- Bounding ϵ^{eval}. At this stage, the partial transcript is $\mathsf{tr} = (\boldsymbol{\gamma}, r_z, G(Z), \sigma, G(\sigma))$. The prover sends the new evaluation claims as $\boldsymbol{\eta} \coloneqq (\eta, \{\eta_i, \eta_{\cup,i}\}_{i\in[\mu]})$. We set the state function $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}\|\boldsymbol{\eta}) = 1$ if and only if the following holds:
  - $\eta = F(\tilde{\boldsymbol{x}}(\sigma), \{\boldsymbol{m}_i(\sigma)\}, \boldsymbol{r}_F(\sigma)) - e(\sigma)$
  - $\eta_i = \tilde{m}_i(\sigma, \boldsymbol{r}_x(\sigma)) - v_i(\sigma)$
  - $\eta_{\cup,i} = \tilde{m}\cup, i(\sigma, \boldsymbol{\tau}(\sigma), \boldsymbol{r}_x(\sigma)) - v_i(\sigma)$

  For the knowledge error, we show that if the claim $G(\sigma)$ is invalid, *i.e.*, $\mathsf{State}(\mathbb{x}, \mathbb{y}, \mathbb{w}, \mathsf{tr}) = 0$, then $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}\|\boldsymbol{\eta}) = 1$ holds with a negligible probability ϵ_{eval}. Clearly, given a randomly sampled $\boldsymbol{\gamma} \in \mathbb{F}^{\log\mu+1}$, the prover can forges a vector $\boldsymbol{\eta}'$ satisfying the following equation with probability at most $\mu/|\mathbb{F}|$:

$$G(\sigma) = \tilde{eq}(r_z, \sigma) \cdot [\eta' + \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{\gamma}) \cdot \eta'_i + \sum_{i\in[\mu]} \mathsf{pow}_{i+\mu}(\boldsymbol{\gamma}) \cdot \eta'_{\cup,i}]. \tag{16}$$

- Bounding $\epsilon^{\mathsf{batch}}$. At this stage, the partial transcript has the form $\mathsf{tr} = (\boldsymbol{\gamma}, r_z, G(Z), \sigma, G(\sigma), \boldsymbol{\eta})$ and the verifier sends $\boldsymbol{c} = (\{(\boldsymbol{y}_j, \boldsymbol{x}_j)\}_j, \{\boldsymbol{x}_t\}_t)$. We set the state function $\mathsf{State}(\mathtt{i}, \mathtt{x}, \mathtt{y}, \mathsf{tr} \| \boldsymbol{c}) = 1$ if and only if the following hold:
  - $(\mathtt{x} = \{\boldsymbol{y}_j, \boldsymbol{x}_j, v_j\}_j, \mathtt{y} = [\![\tilde{m}_{\cup,i}]\!], \mathtt{w} = \bot) \in \mathcal{R}_{\mathsf{eval}}\ \forall j \in [\mu]$
  - $(\mathtt{x} = \{\boldsymbol{x}_t, v_t\}_t, \mathtt{y} = [\![\tilde{m}_i]\!], \mathtt{w} = \bot) \in \mathcal{R}_{\mathsf{eval}}\ \forall t \in [\mu]$
  - $(\mathtt{x} = (\boldsymbol{x}, \boldsymbol{r}_F, e), \mathtt{y} = \bot, \mathtt{w} = \{\boldsymbol{m}_i\}_{i \in [\mu]}) \in \mathcal{R}_F$

  where $\tilde{m}_i = \tilde{\boldsymbol{m}}_i(\sigma)$ and $\tilde{m}_{\cup,i} = \tilde{\boldsymbol{m}}_{\cup,i}(\sigma)$ for all $i \in [\mu]$. For the knowledge error, we show that if the batched witnesses are invalid, *i.e.*, $\mathsf{State}(\mathtt{i}, \mathtt{x}, \mathtt{y}, \mathtt{w}, \mathsf{tr}) = 0$, then $\mathsf{State}(\mathtt{i}, \mathtt{x}, \mathtt{y}, \mathsf{tr} \| \boldsymbol{c}) = 1$ holds with a negligible probability $\epsilon_{\mathsf{batch}}$. Since each protocol $\mathsf{IOR}_{\mathsf{batch}}$ has soundness error ϵ_0, the overall probability is restricted by a union bound as $2\mu \cdot \epsilon_0$. □

$\mathcal{P}$'s inputs: $(\mathbb{i}_{\mathsf{acc}}, \{\mathbb{x}_{\mathsf{acc}}^{(k)}, \mathbb{w}_{\mathsf{acc}}^{(k)}\}_{k=0}^{1})$

$\mathcal{V}$'s inputs: $(\mathbb{i}_{\mathsf{acc}}, \{\mathbb{x}_{\mathsf{acc}}^{(k)}\}_{k=0}^{1})$

Interaction phase.

1: For each $k \in \{0,1\}$, define $\mathsf{vec}^{(k)}$ as

$(\boldsymbol{x}^{(k)} \,\|\, \boldsymbol{r}^{(k)} \,\|\, \boldsymbol{\alpha}^{(k)} \,\|\, \boldsymbol{\tau}^{(k)} \,\|\, \boldsymbol{r}_x^{(k)} \,\|\, \boldsymbol{r}_w^{(k)} \,\|\, (e^{(k)}, v_x^{(k)}, v_1^{(k)}, \ldots, v_\mu^{(k)}) \,\|\, \boldsymbol{m}_1^{(k)} \,\|\, \cdots \,\|\, \boldsymbol{m}_\mu^{(k)})$.

2: $\mathcal{P}$ computes

$$\tilde{\mathsf{vec}}(Z) \coloneqq \textstyle\sum_{k=0}^{1} \tilde{eq}_k(Z) \cdot \mathsf{vec}^{(k)},$$
$$\tilde{x}_\cup(Z, \boldsymbol{Y}, \boldsymbol{X}) \coloneqq \textstyle\sum_{k=0}^{1} \tilde{eq}_k(Z) \cdot \tilde{x}_\cup^{(k)}(\boldsymbol{Y}, \boldsymbol{X}),$$
$$\tilde{m}_i(Z, \boldsymbol{X}) \coloneqq \textstyle\sum_{k=0}^{1} \tilde{eq}_k(Z) \cdot \tilde{m}_i^{(k)}(\boldsymbol{X}) \ \forall i \in [\mu],$$
$$\tilde{m}_{\cup,i}(Z, \boldsymbol{Y}, \boldsymbol{X}) \coloneqq \textstyle\sum_{k=0}^{1} \tilde{eq}_k(Z) \cdot \tilde{m}_{\cup,i}^{(k)}(\boldsymbol{Y}, \boldsymbol{X}) \ \forall i \in [\mu].$$

3: $\mathcal{P}$ parses $\tilde{\mathsf{vec}}(Z)$ as

$$\tilde{\boldsymbol{x}}(Z), \tilde{\boldsymbol{r}}(Z), \tilde{\boldsymbol{\alpha}}(Z), \tilde{\boldsymbol{\tau}}(Z), \tilde{\boldsymbol{r}}_x(Z), \tilde{\boldsymbol{r}}_w(Z), \tilde{e}(Z), \tilde{v}_x(Z),$$
$$\tilde{v}_1(Z), \ldots, \tilde{v}_\mu(Z), \tilde{\boldsymbol{m}}_1(Z), \ldots, \tilde{\boldsymbol{m}}_\mu(Z).$$

4: $\mathcal{V} \to \mathcal{P}$: $\boldsymbol{\gamma} \leftarrow\!\!\$\ \mathbb{F}^{\log(2\mu+2)}$, $r_z \leftarrow\!\!\$\ \mathbb{F}$.

5: $\mathcal{P}$ computes $G(Z) \coloneqq \tilde{eq}(r_z, Z) \cdot \big(p_1(Z) + p_2(Z) + p_3(Z) + p_4(Z)\big)$, where

$$p_1(Z) \coloneqq \tilde{x}_\cup\big(Z, \tilde{\boldsymbol{\tau}}(Z), \tilde{\boldsymbol{r}}_x(Z)\big) - \tilde{v}_x(Z),$$
$$p_2(Z) \coloneqq \textstyle\sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{\gamma}) \cdot \big(\tilde{m}_i\big(Z, \tilde{\boldsymbol{r}}_w(Z)\big) - \tilde{v}_i(Z)\big),$$
$$p_3(Z) \coloneqq \textstyle\sum_{i\in[\mu]} \mathsf{pow}_{\mu+i}(\boldsymbol{\gamma}) \cdot \big(\tilde{m}_{\cup,i}\big(Z, \tilde{\boldsymbol{\tau}}(Z), \tilde{\boldsymbol{r}}_w(Z)\big) - \tilde{v}_i(Z)\big),$$
$$p_4(Z) \coloneqq \mathsf{pow}_{2\mu+1}(\boldsymbol{\gamma}) \cdot \big(F(\tilde{\boldsymbol{x}}(Z), \{\tilde{\boldsymbol{m}}_i(Z)\}_{i\in[\mu]}, \tilde{\boldsymbol{r}}(Z), \tilde{\boldsymbol{\alpha}}(Z)) - \tilde{e}(Z)\big).$$

6: $\mathcal{P}$ and $\mathcal{V}$ engage in a 1-round sum-check protocol for $\sum_{z\in\{0,1\}} G(z) = 0$, and output a claim on $G(\sigma)$.

7: $\mathcal{P} \to \mathcal{V}$: $\eta_x, \{\eta_i, \eta_{\cup,i}\}_{i\in[\mu]}, \eta_F$, where

$$\eta_x \coloneqq \tilde{x}_\cup\big(\sigma, \tilde{\boldsymbol{\tau}}(\sigma), \tilde{\boldsymbol{r}}_x(\sigma)\big) - \tilde{v}_x(\sigma),$$
$$\eta_i \coloneqq \tilde{m}_i\big(\sigma, \tilde{\boldsymbol{r}}_w(\sigma)\big) - \tilde{v}_i(\sigma) \ \forall i \in [\mu],$$
$$\eta_{\cup,i} \coloneqq \tilde{m}_{\cup,i}\big(\sigma, \tilde{\boldsymbol{\tau}}(\sigma), \tilde{\boldsymbol{r}}_w(\sigma)\big) - \tilde{v}_i(\sigma) \ \forall i \in [\mu],$$
$$\eta_F \coloneqq F(\tilde{\boldsymbol{x}}(\sigma), \{\tilde{\boldsymbol{m}}_i(\sigma)\}_{i\in[\mu]}, \tilde{\boldsymbol{r}}(\sigma), \tilde{\boldsymbol{\alpha}}(\sigma)) - \tilde{e}(\sigma).$$

8: $\mathcal{V}$ checks whether $G(\sigma)$ equals

$\tilde{eq}(r_z, \sigma) \cdot \Big(\eta_x + \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{\gamma}) \cdot \eta_i + \sum_{i\in[\mu]} \mathsf{pow}_{\mu+i}(\boldsymbol{\gamma}) \cdot \eta_{\cup,i} + \mathsf{pow}_{2\mu+1}(\boldsymbol{\gamma}) \cdot \eta_F\Big)$.

9: $\mathcal{P}$ and $\mathcal{V}$ engage $2\mu+1$ oracle batching protocols $\mathsf{IOR}_{\mathsf{batch}}$ in parallel. **Output phase.**

10: Define $\{(\boldsymbol{y}_\alpha, \boldsymbol{x}_\alpha, v_\alpha)\}_\alpha, \{(\boldsymbol{y}_\beta, \boldsymbol{x}_\beta, v_\beta)\}_\beta, \{(\boldsymbol{x}_\gamma, v_\gamma)\}_\gamma$ as the output of $\mathsf{IOR}_{\mathsf{batch}}$.

11: Define $\boldsymbol{x} \coloneqq \tilde{\boldsymbol{x}}(\sigma)$, $\boldsymbol{r} \coloneqq \tilde{\boldsymbol{r}}(\sigma)$, $\boldsymbol{\alpha} \coloneqq \tilde{\boldsymbol{\alpha}}(\sigma)$, $e \coloneqq \tilde{e}(\sigma)$.

12: $\mathcal{V}$ outputs

$$\mathbb{x} = (\{(\boldsymbol{y}_\alpha, \boldsymbol{x}_\alpha, v_\alpha)\}_\alpha, \{(\boldsymbol{y}_\beta, \boldsymbol{x}_\beta, v_\beta)\}_\beta, \{(\boldsymbol{x}_\gamma, v_\gamma)\}_\gamma, \boldsymbol{x}, \boldsymbol{r}, \boldsymbol{\alpha}, e),$$
$$\mathbb{y} = (\llbracket \tilde{x}_\cup \rrbracket, \{\llbracket \tilde{m}_{\cup,i} \rrbracket\}_{i\in[\mu]}, \{\llbracket \tilde{m}_i \rrbracket\}_{i\in[\mu]}).$$

Fig. 7: Interactive oracle reduction $\mathsf{IOR}_{\mathsf{fold\text{-}SPS}}$ from $\mathcal{R}_{\mathsf{acc}}^2$ to $\mathcal{R}_{\mathsf{acc}}$.

$\mathcal{P}$'s inputs: $f, \tilde{q}(\boldsymbol{X}), \tilde{w}(\boldsymbol{X})$

$\mathcal{V}$'s inputs: $f, \tilde{q}(\boldsymbol{X})$

1 : $\mathcal{P} \to \mathcal{V} : \tilde{w}(\boldsymbol{X})$

2 : $\mathcal{V}$ checks if $f(\{\tilde{q}(\mathsf{Bits}(j), \boldsymbol{X})\}_{j=0}^{s-1}, \{\tilde{w}(\mathsf{Bits}(j), \boldsymbol{X})\}_{j=0}^{n-1}) = 0$.

Fig. 8: Special-sound protocol Π_{gate} for the gate identity

$\mathcal{P}$'s inputs: $\sigma, \tilde{w}(\boldsymbol{X})$

$\mathcal{V}$'s inputs: σ

1 : $\mathcal{P} \to \mathcal{V} : \tilde{w}(\boldsymbol{X})$

2 : $\mathcal{V}$ checks if $\tilde{w}(\boldsymbol{x}) - \tilde{w}(\sigma(\boldsymbol{x})) = 0$ for all $\boldsymbol{x} \in \{0,1\}^{\log \mu}$.

Fig. 9: Special-sound protocol Π_{wire} for the wiring identity

$\mathcal{P}$'s inputs: $\tilde{p}(\boldsymbol{x}), \tilde{w}(\boldsymbol{X})$

$\mathcal{V}$'s inputs: $\tilde{p}(\boldsymbol{x})$

1 : $\mathcal{P} \to \mathcal{V} : \tilde{w}(\boldsymbol{X})$

2 : $\mathcal{V}$ checks if $\tilde{p}(\boldsymbol{x}) = \tilde{w}(\mathbf{0}^{\log \mu + \log n - \log m}, \boldsymbol{x})$ for all $\boldsymbol{x} \in \{0,1\}^{\log m}$.

Fig. 10: Special-sound protocol Π_{pi} for the instance consistency

$\mathcal{P}$'s inputs: $\mathsf{pp}, \mathtt{i}, \mathbb{x}, \mathbb{y}; \mathbb{w}$

$\mathcal{V}$'s inputs: $\mathsf{pp}, \mathtt{i}, \mathbb{x}, \mathbb{y}$

Interaction phase.

1 : Parse $\mathbb{x} = (\boldsymbol{\alpha}, v, r)$ and $\mathbb{y} = (\llbracket \tilde{u}_0 \rrbracket, \llbracket \tilde{u}_1 \rrbracket)$.

2 : $\mathcal{P}$ computes $\boldsymbol{u} \coloneqq \sum_{i=0}^{1} \tilde{eq}_i(r) \cdot \boldsymbol{u}_i$ and its oracle $\llbracket \tilde{u} \rrbracket$.

3 : $\mathcal{P} \to \mathcal{V} : \llbracket \tilde{u} \rrbracket$.

4 : $\mathcal{V} \to \mathcal{P} : \boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_s \leftarrow\!\$\ \mathbb{F}^{\log(1/\rho)+\log n}$. // out-of-domain samples

5 : $\mathcal{P}$ computes $\mu_1, \ldots, \mu_s \in \mathbb{F}$ where $\mu_j \coloneqq \tilde{u}(\boldsymbol{\alpha}_j)$.

6 : $\mathcal{P} \to \mathcal{V} : \mu_1, \ldots, \mu_s$.

7 : $\mathcal{V}$ samples $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_t \leftarrow\!\$\ \{0,1\}^{\log(1/\rho)+\log n}$

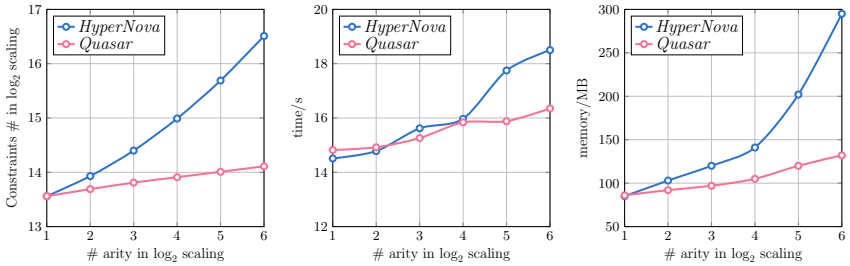
8 : $\mathcal{V}$ queries $\tilde{u}(\boldsymbol{b}_j),\ \tilde{u}_0(\boldsymbol{b}_j),\ \tilde{u}_1(\boldsymbol{b}_j)$ for all $j \in [t]$.

9 : $\mathcal{V}$ checks that $\tilde{u}(\boldsymbol{b}_j) = \sum_{i=0}^{1} \tilde{eq}_i(r) \cdot \tilde{u}_i(\boldsymbol{b}_j)$ for all $j \in [t]$.

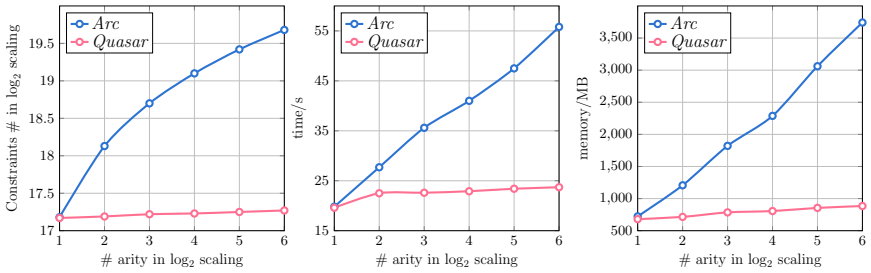
Output phase.

10 : $\mathcal{V}$ outputs the new instance-oracle pair
$\mathbb{x}' \coloneqq (\{(\boldsymbol{\alpha}_j, \mu_j)\}_{j\in[s]}, \{(\boldsymbol{b}_j, \tilde{u}(\boldsymbol{b}_j))\}_{j\in[t]}),\ \mathbb{y}' \coloneqq \llbracket \tilde{u} \rrbracket$.

Fig. 11: Oracle batching protocol $\mathsf{NIR}_{\mathsf{batch}}$ based on linear codes



(a) Curve-based instantiation (Mercury on BN256): Left: constraint count for accumulation verification. Middle: accumulation of proving time. Right: peak accumulation prover memory. We compare Quasar against HyperNova while varying the arity ℓ.



(b) Code-based instantiation (FRI with Poseidon-Merkle): Left: constraint count for accumulation verification. Middle: accumulation prover time. Right: peak accumulation prover memory. We compare Quasar against Arc while varying the arity ℓ.

Fig. 12: Comparison of curve-based and code-based instantiations.

Inputs: $(\mathsf{pk}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_{k\in[\ell]})$.

1 : For each $k \in [\ell]$, set $\boldsymbol{m}_1^{(k)} \leftarrow \mathsf{P}_{\mathsf{sps}}(\mathbb{x}^{(k)}, \mathbb{w}^{(k)})$.

2 : Compute $\tilde{x}_\cup(\boldsymbol{Y}, \boldsymbol{X}) \coloneqq \sum_{k\in[\ell]} \tilde{eq}_{k-1}(\boldsymbol{Y}) \cdot \tilde{x}^{(k)}(\boldsymbol{X})$.

3 : Compute $\tilde{m}_{\cup,1}(\boldsymbol{Y}, \boldsymbol{X}) \coloneqq \sum_{k\in[\ell]} \tilde{eq}_{k-1}(\boldsymbol{Y}) \cdot \tilde{m}_1^{(k)}(\boldsymbol{X})$.

4 : Set $C_\cup \coloneqq \mathsf{Commit}(\mathsf{ck}, \tilde{x}_\cup(\boldsymbol{Y}, \boldsymbol{X}))$, $\quad C_{\cup,1} \coloneqq \mathsf{Commit}(\mathsf{ck}, \tilde{m}_{\cup,1}(\boldsymbol{Y}, \boldsymbol{X}))$.

5 : $r_1 \leftarrow \mathsf{Hash}(C_\cup, C_{\cup,1})$.

6 : For $i \in [2, \mu]$:

- a : For each $k \in [\ell]$, set $\boldsymbol{m}_i^{(k)} \leftarrow \mathsf{P}_{\mathsf{sps}}(\mathbb{x}^{(k)}, \mathbb{w}^{(k)}, \{\boldsymbol{m}_j^{(k)}, r_j\}_{j=1}^{i-1})$.
- b : Set $\tilde{m}_{\cup,i}(\boldsymbol{Y}, \boldsymbol{X}) \coloneqq \sum_{k\in[\ell]} \tilde{eq}_{k-1}(\boldsymbol{Y}) \cdot \tilde{m}_i^{(k)}(\boldsymbol{X})$.
- c : Set $C_{\cup,i} \coloneqq \mathsf{Commit}(\mathsf{ck}, \tilde{m}_{\cup,i}(\boldsymbol{Y}, \boldsymbol{X}))$.
- d : $r_i \leftarrow \mathsf{Hash}(r_{i-1}, C_{\cup,i})$.

7 : $\alpha \coloneqq r_\mu, \quad \boldsymbol{\alpha} \coloneqq (\alpha, \alpha^2, \ldots, \alpha^{2^{\log m}-1})$.

8 : $\boldsymbol{r}_{sc} \leftarrow \mathsf{Hash}(\boldsymbol{\alpha})$.

9 : **NI sum-check (over $\boldsymbol{Y}$):**

10 : Define $\bar{x}(\boldsymbol{Y}) \coloneqq \sum_{k\in[\ell]} \tilde{eq}_{k-1}(\boldsymbol{Y}) \cdot \mathbb{x}^{(k)}$.

11 : For each $i \in [\mu]$, define $\bar{m}_i(\boldsymbol{Y}) \coloneqq \sum_{k\in[\ell]} \tilde{eq}_{k-1}(\boldsymbol{Y}) \cdot \boldsymbol{m}_i^{(k)}$.

12 : Define $G(\boldsymbol{Y}) \coloneqq F(\bar{x}(\boldsymbol{Y}), \{\bar{m}_i(\boldsymbol{Y})\}_{i\in[\mu]}, \boldsymbol{r}_F) \cdot \tilde{eq}(\boldsymbol{Y}, \boldsymbol{r}_{sc})$.

13 : Run a $(\log \ell)$-round non-interactive sum-check for $\sum_{\boldsymbol{y}\in\{0,1\}^{\log \ell}} G(\boldsymbol{y}) = 0$,

obtaining sum-check polynomials $\{G_j(Y)\}_{j\in[\log \ell]}$ and challenges $\boldsymbol{\tau} \in \mathbb{F}^{\log \ell}$.

14 : Set $e \coloneqq G_{\log \ell}(\tau_{\log \ell})$.

15 : Set $\tilde{x}(\boldsymbol{X}) \coloneqq \tilde{x}_\cup(\boldsymbol{\tau}, \boldsymbol{X})$.

16 : For all $i \in [\mu]$, set $\tilde{m}_i(\boldsymbol{X}) \coloneqq \tilde{m}_{\cup,i}(\boldsymbol{\tau}, \boldsymbol{X})$ and $C_i \coloneqq \mathsf{Commit}(\mathsf{ck}, \tilde{m}_i(\boldsymbol{X}))$.

17 : $(\boldsymbol{r}_x, \boldsymbol{r}_w) \leftarrow \mathsf{Hash}(\boldsymbol{r}_{sc}, \{C_i\}_{i\in[\mu]})$.

18 : Output proof π consisting of:

message $\tilde{x}$; commitments $C_\cup, \{C_{\cup,i}\}_{i\in[\mu]}, \{C_i\}_{i\in[\mu]}$;

challenges $\boldsymbol{\alpha}, \boldsymbol{\tau}, \boldsymbol{r}_{sc}, \boldsymbol{r}_x, \boldsymbol{r}_w$; sum-check $\{G_j(Y)\}_{j\in[\log \ell]}, e$.

Fig. 13: The prover algorithm $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{P}$ for the non-interactive reduction.

Inputs: $(\mathsf{vk}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi)$.

1 : Parse π as $\tilde{x}, C_\cup, \{C_{\cup,i}\}_{i\in[\mu]}, \{C_i\}_{i\in[\mu]}, \boldsymbol{\alpha}, \boldsymbol{\tau}, \boldsymbol{r}_{sc}, \boldsymbol{r}_x, \boldsymbol{r}_w, \{G_j(Y)\}_{j\in[\log \ell]}, e$.

2 : **Recompute SPS challenges:**

3 : $r_1 \leftarrow \mathsf{Hash}(C_\cup, C_{\cup,1})$.

4 : $r_i \leftarrow \mathsf{Hash}(r_{i-1}, C_{\cup,i})$ for all $i \in [2, \mu]$.

5 : **Recompute sum-check challenges:**

6 : $\boldsymbol{r}'_{sc} \leftarrow \mathsf{Hash}(\boldsymbol{\alpha})$; reject if $\boldsymbol{r}'_{sc} \neq \boldsymbol{r}_{sc}$.

7 : $\tau'_j \leftarrow \mathsf{Hash}(G_j(Y))$ for all $j \in [\log \ell]$; reject if $\boldsymbol{\tau}' \neq \boldsymbol{\tau}$.

8 : **Check sum-check consistency:**

9 : Reject if $G_1(0) + G_1(1) \neq 0$.

Reject if $G_{j+1}(0) + G_{j+1}(1) \neq G_j(\tau_j)$ for any $j \in [\log \ell - 1]$.

10 : **Check final value:**

11 : Reject if $e \neq G_{\log \ell}(\tau_{\log \ell})$.

12 : $(\boldsymbol{r}'_x, \boldsymbol{r}'_w) \leftarrow \mathsf{Hash}(\boldsymbol{r}_{sc}, \{C_i\}_{i\in[\mu]})$; reject if mismatch.

13 : $\boldsymbol{x}$ is the coefficient vector of $\tilde{x}(\boldsymbol{X})$.

14 : Output instance $\mathbb{x}_{\mathsf{acc}} \coloneqq (\boldsymbol{x}, \{C_{\cup,i}\}_{i\in[\mu]}, C_\cup, \{C_i\}_{i\in[\mu]}, \{r_i\}_{i\in[\mu]}, \boldsymbol{\alpha}, \boldsymbol{\tau}, \boldsymbol{r}_x, \boldsymbol{r}_w, e)$.

Fig. 14: The verifier algorithm $\mathsf{NIR}_{\mathsf{cast\text{-}SPS}}.\mathcal{V}$ for the non-interactive reduction

Inputs: $(\mathsf{pk}, \{(\mathbb{x}_{\mathsf{acc}}^{(k)}, \mathbb{w}_{\mathsf{acc}}^{(k)})\}_{k\in\{0,1\}})$.

1 : For each $k \in \{0,1\}$, define $\mathsf{vec}^{(k)}$ as

$$\mathsf{vec}^{(k)} \coloneqq (\boldsymbol{x}^{(k)} \| \boldsymbol{r}^{(k)} \| \boldsymbol{\alpha}^{(k)} \| \boldsymbol{\tau}^{(k)} \| \boldsymbol{r}_x^{(k)} \| \boldsymbol{r}_w^{(k)} \| (e^{(k)}, v_x^{(k)}, v_1^{(k)}, \ldots, v_\mu^{(k)}) \| \boldsymbol{m}_1^{(k)} \| \cdots \| \boldsymbol{m}_\mu^{(k)}).$$

2 : Compute $\tilde{\mathsf{vec}}(Z) \coloneqq \sum_{k=0}^{1} \tilde{eq}_k(Z) \cdot \mathsf{vec}^{(k)}, \tilde{x}_\cup(Z, \boldsymbol{X}) \coloneqq \sum_{k=0}^{1} \tilde{eq}_k(Z) \cdot \tilde{x}_\cup^{(k)}$.

3 : For all $i \in [\mu]$, compute:

$$\tilde{m}_i(Z, \boldsymbol{X}) \coloneqq \sum_{k=0}^{1} \tilde{eq}_k(Z) \cdot \tilde{m}_i^{(k)}, \tilde{m}_{\cup,i}(Z, \boldsymbol{Y}, \boldsymbol{X}) \coloneqq \sum_{k=0}^{1} \tilde{eq}_k(Z) \cdot \tilde{m}_{\cup,i}^{(k)}.$$

4 : Parse $\tilde{\mathsf{vec}}(Z)$ as $\tilde{\boldsymbol{x}}(Z), \tilde{\boldsymbol{r}}(Z), \tilde{\boldsymbol{\alpha}}(Z), \tilde{\boldsymbol{\tau}}(Z), \tilde{\boldsymbol{r}}_x(Z), \tilde{\boldsymbol{r}}_w(Z), \tilde{e}(Z), \tilde{v}_x(Z), \tilde{v}_1(Z), \ldots, \tilde{v}_\mu(Z), \tilde{\boldsymbol{m}}_1(Z), \ldots, \tilde{\boldsymbol{m}}_\mu(Z)$.

5 : $\boldsymbol{\gamma}, r_z \leftarrow \mathsf{Hash}(\{\mathbb{x}_{\mathsf{acc}}^{(k)}\}_{k\in\{0,1\}})$.

6 : Define:

$$p_1(Z) \coloneqq \tilde{x}_\cup(Z, \tilde{\boldsymbol{r}}_x(Z)) - \tilde{v}_x(Z),$$

$$p_2(Z) \coloneqq \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{\gamma}) \cdot \big(\tilde{m}_i(Z, \tilde{\boldsymbol{r}}_w(Z)) - \tilde{v}_i(Z)\big),$$

$$p_3(Z) \coloneqq \sum_{i\in[\mu]} \mathsf{pow}_{\mu+i}(\boldsymbol{\gamma}) \cdot \big(\tilde{m}_{\cup,i}(Z, \tilde{\boldsymbol{\tau}}(Z), \tilde{\boldsymbol{r}}_w(Z)) - \tilde{v}_i(Z)\big),$$

$$p_4(Z) \coloneqq \mathsf{pow}_{2\mu+1}(\boldsymbol{\gamma}) \cdot \big(F(\tilde{\boldsymbol{x}}(Z), \{\tilde{\boldsymbol{m}}_i(Z)\}_{i\in[\mu]}, \tilde{\boldsymbol{r}}(Z), \tilde{\boldsymbol{\alpha}}(Z)) - \tilde{e}(Z)\big).$$

7 : Set $G(Z) \coloneqq \tilde{eq}(r_z, Z) \cdot (p_1(Z) + p_2(Z) + p_3(Z) + p_4(Z))$.

8 : **1-round NI sum-check over** $Z \in \{0,1\}$**:**

9 : Run NI sum-check for $\sum_{z\in\{0,1\}} G(z) = 0$,

obtaining polynomial $G_1(Z)$, $\sigma \in \mathbb{F}$, and claim $v_G \coloneqq G(\sigma)$.

10 : Set $\eta_x \coloneqq p_1(\sigma)$,

$\eta_i \coloneqq \tilde{m}_i(\sigma, \tilde{\boldsymbol{r}}_w(\sigma)) - \tilde{v}_i(\sigma)$, $\eta_{\cup,i} \coloneqq \tilde{m}_{\cup,i}(\sigma, \tilde{\boldsymbol{\tau}}(\sigma), \tilde{\boldsymbol{r}}_w(\sigma)) - \tilde{v}_i(\sigma)$ for all $i \in [\mu]$.

11 : Run $2\mu + 1$ instances of $\mathsf{IOR}_{\mathsf{batch}}.\mathcal{P}$ in parallel and obtain

$\{\boldsymbol{y}_\alpha, \boldsymbol{x}_\alpha, v_\alpha\}_\alpha$, $\{\boldsymbol{y}_\beta, \boldsymbol{x}_\beta, v_\beta\}_\beta$, $\{\boldsymbol{x}_\gamma, v_\gamma\}_\gamma$.

12 : Output $\mathbb{x} \coloneqq (\{\boldsymbol{y}_\alpha, \boldsymbol{x}_\alpha, v_\alpha\}_\alpha, \{\boldsymbol{y}_\beta, \boldsymbol{x}_\beta, v_\beta\}_\beta, \{\boldsymbol{x}_\gamma, v_\gamma\}_\gamma, \boldsymbol{x}, \boldsymbol{r}, \boldsymbol{\alpha}, e)$,

13 : $\mathbb{y} \coloneqq ([\![\tilde{x}_\cup]\!], \{[\![\tilde{m}_{\cup,i}]\!]\}_{i\in[\mu]}, \{[\![\tilde{m}_i]\!]\}_{i\in[\mu]})$,

14 : π containing (G_1, σ, v_G) and the batching transcripts.

Fig. 15: The prover algorithm $\mathsf{NIR}_{\mathsf{fold}}.\mathcal{P}$ for the non-interactive reduction

Inputs: $(\mathsf{vk}, \{\mathbb{x}_{\mathsf{acc}}^{(k)}\}_{k\in\{0,1\}}, \pi)$.

1 : Parse π to obtain $(G_1(Z), \sigma, v_G)$ and the batching transcripts.

2 : $(\boldsymbol{\gamma}, r_z) \leftarrow \mathsf{Hash}(\{\mathbb{x}_{\mathsf{acc}}^{(k)}\}_{k\in\{0,1\}})$.

3 : **Check 1-round sum-check:**

Reject if $G_1(0) + G_1(1) \neq 0$.

$\sigma' \leftarrow \mathsf{Hash}(G_1(Z))$; reject if $\sigma' \neq \sigma$.

Reject if $v_G \neq G(\sigma)$.

4 : **Check aggregated identity:**

Reject if $v_G \neq \tilde{eq}(r_z, \sigma) \cdot \Big(\eta_x + \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{\gamma})\,\eta_i + \sum_{i\in[\mu]} \mathsf{pow}_{\mu+i}(\boldsymbol{\gamma})\,\eta_{\cup,i} + \mathsf{pow}_{2\mu+1}(\boldsymbol{\gamma})\,\eta_F\Big)$.

5 : Run $2\mu+1$ instances of $\mathsf{IOR}_{\mathsf{batch}}.\mathcal{V}$ in parallel and verify $\{\boldsymbol{y}_\alpha, \boldsymbol{x}_\alpha, v_\alpha\}_\alpha$, $\{\boldsymbol{y}_\beta, \boldsymbol{x}_\beta, v_\beta\}_\beta$, $\{\boldsymbol{x}_\gamma, v_\gamma\}_\gamma$.

6 : Compute $\boldsymbol{x} \coloneqq \tilde{\boldsymbol{x}}(\sigma)$, $\boldsymbol{r} \coloneqq \tilde{\boldsymbol{r}}(\sigma)$, $\boldsymbol{\alpha} \coloneqq \tilde{\boldsymbol{\alpha}}(\sigma)$, $e \coloneqq \tilde{e}(\sigma)$.

7 : Output $\mathbb{x}, \mathbb{y}$ as in Figure 15.

Fig. 16: The verifier algorithm $\mathsf{NIR}_{\mathsf{fold}}.\mathcal{V}$ for the non-interactive reduction.