# Quasar: Sublinear Accumulation Schemes for Multiple Instances

Tianyu Zheng[1], Shang Gao[1], Yu Guo[2], and Bin Xiao[1]

[1] The Hong Kong Polytechnic University, Hong Kong, China
{tian-yu.zheng}@connect.polyu.hk
{shang-jason.gao, b.xiao}@polyu.edu.hk
[2] SECBIT Labs, Su Zhou, China
yu.guo@secbit.io

**Abstract.** Accumulation is a core technique in state-of-the-art Incrementally Verifiable Computations (IVCs), enabling the avoidance of recursively implementing costly SNARK verification within circuits. However, the recursion overhead in existing IVCs remains significant due to the accumulation verifier complexity, which scales linearly with the number of accumulated instances. In this work, we present a novel accumulation scheme for multiple instances based on polynomial commitment schemes, achieving a theoretical verifier complexity that is sublinear in the number of instances. Technically, our scheme leverages partial evaluation of polynomials to replace random linear combinations, thereby minimizing the costly Commitment Random Linear Combination (CRC) operations on the verifier side. Building on this accumulation scheme, we introduce Quasar, a multi-instance IVC with small recursion overhead in practice. Notably, Quasar reduces the number of costly CRC operations in the recursive circuit from linear to quasi-linear, substantially improving practical performance. By instantiating Quasar with appropriate polynomial commitment schemes, it can achieve linear-time accumulation prover complexity, plausible post-quantum security, and support for parallelizable proving at each step.

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

## 1 Introduction

Succinct Non-interactive ARguments of Knowledge (SNARKs) are powerful cryptographic primitives that enable efficient and succinct verification of computations across a wide range of applications, including zero-knowledge virtual machines [2,36], verifiable databases [45,47], and verifiable inference systems [35,40]. As the complexity of computational tasks continues to grow real-world applications, existing SNARKs face significant scalability challenges. The primary bottleneck arises from the requirement that the prover must explicitly record all intermediate values throughout the computation and generate a proof for the entire computation trace with a general-purpose proof system. Nguyen et

al. in [38] categorized the class of proof systems with this feature as monolithic SNARKs. For instance, consider generating a proof for a typical zkEVM represented as an arithmetic circuit with $2^{26}$ gates: a monolithic SNARK (such as Plonk [28]) requires around 1 hour of proving time [21] and more than 200GB of memory [41]. Despite substantial research efforts aimed at optimizing the prover performance of monolithic SNARKs in terms of both time complexity and memory consumption [11,31,29], the improvements have been incremental and often insufficient for large-scale computations.

*IVC and PCD.* In contrast to monolithic SNARKs, an alternative approach is partitioning a large computation into multiple smaller chunks, allowing the prover to transform each chunk into a *predicate instance* (typically as a SNARK or NARK proof) and aggregate them using specialized techniques such as proof composition or aggregation [9,10]. This strategy is typically referred to as Incrementally Verifiable Computation (IVC) [42], a cryptographic primitive that supports sequential computations while enabling efficient verification of the execution at any point. A generalization of IVC to directed acyclic graphs is known as *Proof-Carrying Data* (PCD) [22]. IVC/PCD clearly offers several advantages over monolithic SNARKs: (1) the proving algorithm operates on a much smaller input at any given time, dramatically reducing memory requirements, (2) the prover can generate predicate instances for any finished chunk rather than waiting for the entire computation to finish, and (3) some IVCs based on multi-accumulation/folding (explained later) and PCDs enable parallelization of the prover, thereby decreasing the overall proving time.

*Accumulation/folding schemes.* The traditional approach to constructing IVC employs a general-purpose Succinct Non-interactive Argument of Knowledge (SNARK) at each step $i$ to attest to the correctness of the predicate instance output by step $i-1$ recursively. Such recursive-composition-based schemes [9,8,24] require implementing the entire verification logic of the $(i-1)$-th SNARK within the $i$-th SNARK proving circuit (i.e., the recursive circuit), which incurs significant overhead because the verification involves costly cryptographic operations, such as elliptic curve pairings, as noted by Chiesa in [24]. Therefore, a more practical construction is based on a primitive called an *accumulation or folding scheme* [12,17,16,10,34]. Generally speaking, these schemes enable the prover to efficiently accumulate the predicate instance for each chunk (a NARK proof) into a running "accumulator" and defer their verification to the final step. As a result, the recursive circuit only needs to additionally encode the verification of each accumulation step, and the final accumulator can be checked by a decider in time that is independent of the number of recursive steps.

### 1.1 Problem statements.

Although IVCs/PCDs provide a more memory-efficient approach for proving large-scale computations, the recursion overhead introduced at each step remains

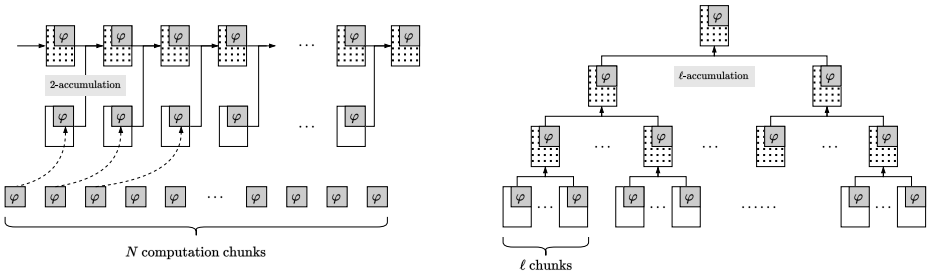


**Fig. 1.** Comparisons between workflows of IVCs and PCDs: the grey block represented the computation chunk represented as predicate $\varphi$, which is integrated into (1) the white block for an empty circuit and (2) the dotted block for a recursive circuit.

significant, resulting in new efficiency challenges. Taking Nova as an example, its recursive circuit consists of two main parts:

- the trace for computing a new chunk, denoted as *computational overhead*;
- the trace for verifying the accumulation between a predicate instance and an accumulator at previous step.

In practical implementations, the recursion overhead for verifying a single-instance accumulation proof consumes around 10,000 gates [32], which restricts the prover from instantiating an IVC with too many recursive steps.

The same issue exists in PCDs [33,25,46,38] based on multi-accumulation schemes. As discussed in ProtoGalaxy [25], by employing a multi-accumulation scheme, a prover can process more than one instances or accumulators at each step, thereby reducing the total number of recursive steps. However, we emphasize that the improvement in recursion overhead is still limited. This is because for each step, the recursive cost is dominated by the linear combination of commitments, which scales linearly with the number of predicate instances accumulated. Specifically, let $\mathsf{CRC}$ (Commitments Random linear Combination) denote the operation of computing $C := r_1 \cdot C_1 + r_2 \cdot C_2$. For the IVC on the left-hand side of Figure 1, running $N$ steps of single-instance accumulation schemes, the total number of $\mathsf{CRC}$ operations in the recursive circuits among all steps is $N \cdot t$, where $t$ is the maximum number of commitments contained in an accumulator. Correspondingly, for the PCD on the right-hand side, the prover runs $N/\ell$ steps of an $\ell$-accumulation scheme, $\ell \geq 2$, and the total number of $\mathsf{CRC}$ operations is $Nt + Nt/\ell + \cdots + 1 \leq 2Nt$. Though a recent work [38] concretely reduces this overhead, its complexity is still linear in $N$. Since $\mathsf{CRC}$ operations typically involve non-native computations, such as *scalar multiplication over group* for curve-based commitments and *Merkle path checking* for code-based commitments, they take a dominated proportion in the recursive circuit.

We observe that the main cause for the above bottleneck is that, for $\ell$ accumulated predicate instances, the accumulation verifier must perform $O(\ell)$ computations (including $O(\ell)$ $\mathsf{CRC}$ operations). As a result, although multi-accumulation schemes reduce the number of recursive steps and thereby decrease some of the

prover's marginal work, such as circuit synthesis, the overall recursion overhead does not improve (and may even worsen). We provide more detailed data in Table 1 to illustrate this point and illustrate our motivation as follows.

*Motivations.* Can we propose a new accumulation scheme with verifier time sublinear in the predicate instances $\ell$? More practically, can we further reduce the ComBatch operations to a constant size?

### 1.2 Our Results

We overcome the aforementioned limitations by proposing an efficient multi-instance accumulation scheme based on polynomial commitment schemes (PCS). In general, we focus on a special case where the prover only takes $\ell$ *predicate instances* and *one accumulator* as inputs. Under this setting, we show that there exists an theoretical construction of an accumulation scheme with verifier complexity sublinear in $\ell$. Based on this, we further built Quasar, a multi-instance IVC with the following features:

- **Small recursion overhead.** The total recursive circuits for all steps of the multi-instance IVC only contains *quasi-linear* $O(\sqrt{N})$ number of CRC operations among all steps. Specifically, the recursive circuit at each step includes $O(\ell)$ field operations[3] and $O(1)$ CRC operations.
- **Linear accumulation prover time.** If the underlying PCS provides linear proving complexity, e.g., [31,43] based on linear-time-encodable codes, and [26] based on elliptic curves, the constructed accumulation scheme can achieve *linear-time prover complexity*. Moreover, if the NARK is instantiated with an linear-time IOP, e.g., Hyperplonk, then the IVC also has linear prover time.
- **Plausible post-quantum security.** The multi-instance IVC can achieve *plausible post-quantum* security by instantiating with PCS based on cryptographic hash functions or lattice-based assumptions.
- **Parallelizable proving algorithm.** Although the multi-instance IVC does not have the tree structure of PCD, which naturally admits parallel proving strategies, we claim that it still allows *parallelism* of the accumulation prover at each step, thereby enhancing prover scalability.

### 1.3 Performance Analysis

We briefly summarize the theoretical performance of our new multi-instance accumulation scheme. We instantiate Quasar with a NARK for (multilinear) plonkish constraint systems and two PCSs including an elliptic-curve-based one [26] and a linear-code-based one [31]. We denote them as Quasar (curve) and Quasar (code) respectively in Table 1. Since the code-based PCS relies on minimal assumptions to achieve plausible post-quantum secure, the constructed accumulation scheme also preserves this property. For the theoretical complexity, the

[3] We do not reduce the $\mathbb{F}$ operations to sublinear is because there exists more efficient implementations with linear cost

**Table 1.** Comparisons between multi-accumulation schemes: let $\mathsf{RO}, \mathbb{G}$ denotes an random oracle query and a group operations respectively, the parameters $\ell$ denotes the input predicate instances, $d$ is the maximum degree of the relation, $m$ is witness length, $\rho$ is the code rate.

| **Schemes** | **Constraints** | **PCS type** | **PQ** | **Verifier cost** |
|---|---|---|---|---|
| ProtoGalaxy [25] | Plonkish | Elliptic Curves | No | $O(1)\mathsf{RO}, O(\ell \cdot d)\mathbb{G}$ |
| KiloNova [46] | CCS | Elliptic Curves | No | $O(\log n)\mathsf{RO}, O(\ell)\mathbb{G}$ |
| **Quasar (curve)** | Plonkish | Elliptic Curves | No | $O(\log \ell)\mathsf{RO}, O(1)\mathbb{G}$ |
| Arc [20] | R1CS | RS codes | Yes | $O(\ell \cdot \frac{\lambda}{\log(1/\rho)} \cdot \log n)\mathsf{RO}$ |
| WARP [15] | PESAT | Linear codes | Yes | $O(\ell \cdot \frac{\lambda}{\log(1/\rho)} \cdot \log n)\mathsf{RO}$ |
| **Quasar (code)** | Plonkish | Linear codes | Yes | $O(\frac{\lambda}{\log(1/\rho)} \cdot (\log n + \log \ell))\mathsf{RO}$ |

key point is that both our two constructions have accumulation verifier cost sublinear in $\ell$, while all other solutions scale linearly in $\ell$. For curve-based schemes, Quasar (curve) achieves a notable performance gain by requiring only a constant number of group operations in the verification process, thereby substantially reducing recursion overhead in practical deployments.

## 2 Technical Overview

We provide a technical overview of our work from a top-down perspective to facilitate understanding, which differs from the order presented in Sections 4–6.

### 2.1 Multi-Instance IVC

We first introduce the new notion of multi-instance IVC. As mentioned in the introduction, a multi-accumulation scheme allows the prover to accumulate arbitrary $\ell \in \mathbb{N}$ predicate instances and $m \in \mathbb{N}$ accumulators at each step, thereby restricting the number of recursive steps to an acceptable range. However, when considering practical applications, such as building a zkVM, it is sufficient to consider a simplified model: the prover still performs sequential computations as in standard IVCs, but can accumulate *multiple instances* and *one accumulator* at each step[4]. We illustrate this new notion as *multi-instance IVC* in Figure 2, where the prover updates the running accumulator until $\ell$ chunk instances have been generated. Since the multi-instance IVC is an extension of the existing IVC

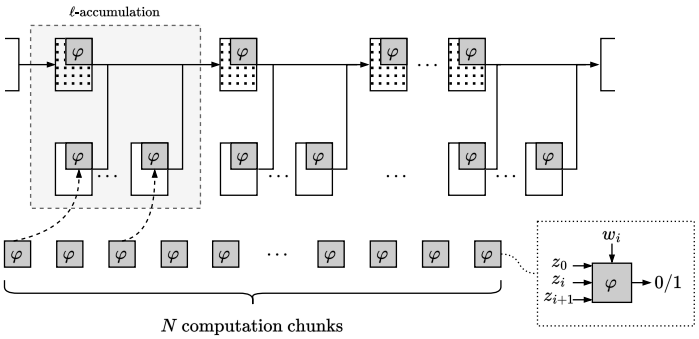


**Fig. 2.** Overview of multi-instance IVC: $\varphi$ denotes the predicate function representing a computation chunk, s.t., $\varphi(z_0, z_i, z_{i+1}, w_i) = 1$ indicates the compliance of inputs.

model, it is able to capture zkVM and other IVC applications. Additionally, the ability to handle multiple instances further allows the prover to achieve a trade-off between the number of recursive steps and the recursion overhead, which enables the implementation of parallel proving algorithms. We present the definition of multi-instance IVC as a new cryptographic primitive in Definition 1.

**Definition 1 (Multi-instance IVC).** *A* multi-instance IVC *is a tuple consisting of a prover and a verifier algorithm with the following interfaces:*

[4] We find it difficult to achieve an accumulation for multiple accumulators, i.e., PCD, with sublinear verification time, which is left as an open problem

- $\mathsf{IVC}.\mathcal{P}(\boldsymbol{z}_0, \boldsymbol{z}_i, \boldsymbol{w}_i, \boldsymbol{z}_{i+1}, \Pi_i) \to \Pi_{i+1}$*: Takes as input the initial vector* $\boldsymbol{z}_0 \in \mathbb{F}^\ell$*, a vector* $\boldsymbol{z}_i \in \mathbb{F}^\ell$ *output by* $\ell$ *predicate functions at step* $i-1$*, witness* $\boldsymbol{w}_i \in \mathbb{F}^\ell$*, the expected outputs* $\boldsymbol{z}_{i+1} \in \mathbb{F}^\ell$ *at step* $i$*, and an IVC proof* $\Pi_i$*. The prover outputs a new proof* $\Pi_{i+1}$*.*
- $\mathsf{IVC}.\mathcal{V}(\boldsymbol{z}_0, \boldsymbol{z}_{i+1}, \Pi_{i+1}) \to b$*: Takes as input the initial vector* $\boldsymbol{z}_0 \in \mathbb{F}^\ell$*, a vector* $\boldsymbol{z}_{i+1} \in \mathbb{F}^\ell$ *output at step* $i$*, and an IVC proof* $\Pi_{i+1}$*. The verifier outputs* 1 *for 'accept' and* 0 *otherwise.*

The $\mathsf{IVC}$ scheme is expected to preserve completeness and knowledge soundness, as formally defined in Appendix A.6. We draw a conclusion by adopting from [17] for the construction of multi-instance IVC in Theorem 1, reducing our task to constructing an appropriate *multi-instance accumulation scheme* that accumulates $\ell$ instances and one accumulator into a new accumulator.

**Theorem 1 (Multi-instance IVC from accumulation (informal)).** *In the standard model, given a NARK for NP relations and a multi-instance accumulation scheme for the NARK, with verifier complexity sublinear in its input length, there exists an efficient transformation that outputs a multi-instance IVC scheme for constant-depth compliance predicates. Moreover, if the accumulation verifier complexity is sublinear in the accumulated instance number* $\ell$*, then the recursive cost of the IVC scheme is sublinear in the total instance number* $N$*.*

*Remark 1.* We clarify that the IVC constructed from a multi-accumulation scheme for $\ell$ chunks is distinct from the one constructed using a single-instance accumulation scheme for a single chunk of $\ell$ times larger size. The key difference is that the final proof of the latter is also $\ell$ times larger.

### 2.2 New Construction for Accumulation Scheme

We first present the definition of a multi-instance accumulation scheme.

**Definition 2 (Multi-instance accumulation scheme).** *A multi-instance accumulation scheme* $\mathsf{ACC}$ *for a* $\mathsf{NARK} := (\mathcal{P}, \mathcal{V})$ *consists of three algorithms: a prover* $\mathcal{P}$*, a verifier* $\mathcal{V}$*, and a decider* $\mathcal{D}$*, with the following interfaces:*

- $\mathsf{ACC}.\mathcal{P}(\{\mathbb{x}_k\}_{k\in[\ell]}, \pi, \mathsf{acc}) \to (\mathsf{acc}', \mathsf{pf})$*: The accumulation prover takes as input a* ***multi-predicate tuple*** $(\{\mathbb{x}_k\}_{k\in[\ell]}, \pi)$*, where* $\pi$ *is a NARK proof* $\pi$*, and an old accumulator* $\mathsf{acc}$*, outputs a new accumulator* $\mathsf{acc}'$ *together with a proof* $\mathsf{pf}$*.*
- $\mathsf{ACC}.\mathcal{V}(\{\mathbb{x}_k\}_{k\in[\ell]}, \pi.\mathbb{x}, \mathsf{acc}.\mathbb{x}) \to b$*: The accumulation verifier takes as input a multi-predicate instance* $(\{\mathbb{x}_k\}_{k\in[\ell]}, \pi.\mathbb{x})$*, and an accumulator instance* $\mathsf{acc}.\mathbb{x}$*, outputs* 1 *for 'accept' and* 0 *otherwise.*
- $\mathsf{ACC}.\mathcal{D}(\mathsf{acc}) \to b$*: The decider takes as input an accumulator* $\mathsf{acc}$ *and outputs* 1 *for 'accept' and* 0 *otherwise.*

We highlight the notion of a *multi-predicate tuple* as the instant-witness pair $(\mathbb{x} := (\{\mathbb{x}_k\}_{k\in[\ell]}, \pi.\mathbb{x}), \mathbb{w} := \pi.\mathbb{w})$. By accumulating $\{\mathbb{x}_k\}_{k\in[\ell]}$ into one $\mathbb{x}$, the verifier can obtain a tuple $((\mathbb{x}, \pi.\mathbb{x}), \pi.\mathbb{w})$ with the same form as $(\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$. A multi-instance accumulation scheme $\mathsf{ACC}$ should provide completeness and

knowledge soundness formally defined in Appendix A.3. For the construction, Bünz et al. in previous works [16,20] introduced a general framework for building accumulation schemes from the following components:

- A cast reduction $\mathsf{NIR}_{\mathsf{cast}}$ from an NP relation $\mathcal{R}$ to a committed relation $\mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}}$.
- A many-to-one reduction $\mathsf{NIR}_{\mathsf{fold}}$ from $(\mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}})^*$ to $\mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}}$, where $(\mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}})^*$ is a multi-instance relation $\{(\{\mathbb{x}_k\}_{k\in[\ell]}, \{\mathbb{w}_k\}_{k\in[\ell]}) : \forall k \in [\ell], (\mathbb{x}_k, \mathbb{w}_k) \in \mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}}\}$.

where $\mathsf{cm}$ can be any applicable commitment scheme, $[\ell]$ denotes the range $[0, \ell - 1]$. However, this framework can not fulfill our requirements on efficiency. Note that $\mathsf{NIR}_{\mathsf{fold}}$ takes inputs as $\ell$ predicate instances as $\{\mathbb{x}^{(k)}, \pi^{(k)}.\mathbb{x}\}_{k\in[\ell]}$, which is different from our multi-predicate instance $(\{\mathbb{x}_k\}_{k\in[\ell]}, \pi.\mathbb{x})$ in Definition 2. Since each $\pi^{(k)}$ generated by $\mathsf{NIR}_{\mathsf{cast}}$ is a NARK proof consists of multiple commitments, accumulating them into one $\pi$ incurs $O(\ell)$ $\mathsf{CRC}$ operations on the verifier side. Alternatively, we propose a new construction as formalized in Theorem 2, and provide a formal proof of its correctness in Appendix B.3.

**Theorem 2 (Multi-instance accumulation construction (informal)).** *Given the following non-interactive reductions in the random oracle model:*

- *a multi-cast reduction* $\mathsf{NIR}_{\mathsf{multicast}}$ *from the multi-instance relation* $\mathcal{R}^{\ell}$ *to a committed relation* $\mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}}$,
- *a 2-to-1 reduction* $\mathsf{NIR}_{\mathsf{fold}}$ *from* $(\mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}})^2$ *to* $\mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}}$,

*there exists a transformation* $T[\mathsf{NIR}_{\mathsf{multicast}}, \mathsf{NIR}_{\mathsf{fold}}, \mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}}] = (\mathsf{NARK}, \mathsf{ACC})$*, where* $\mathsf{NARK}$ *is a non-interactive argument for* $\mathcal{R}$ *and* $\mathsf{ACC}$ *is an accumulation scheme for* $\mathsf{NARK}$*, both in the random oracle model.*

Unlike the previous framework, our new construction requires a primitive called *mult-cast reduction* and a relatively simple 2-to-1 reduction, as depicted in Figure 2. The general idea is to shift the task of combining $\ell$ committed predicate instances from $\mathsf{NIR}_{\mathsf{fold}}$ to $\mathsf{NIR}_{\mathsf{cast}}$. Since the cast reduction takes as inputs *non-committed* instances in $\mathcal{L}(\mathcal{R})$, it will perform far less $\mathsf{CRC}$ operations. In particular, the reduction $\mathsf{NIR}_{\mathsf{multicast}}$ enables the prover to accumulate and cast multiple non-committed predicate tuple $\{(\mathbb{x}^{(k)}, \mathbb{w}^{(k)})\}_{k\in[\ell]}$ into *one* committed instance-witness tuple $((\mathbb{x}, \pi.\mathbb{x}), \pi.\mathbb{w})$ (with the same form as an accumulator) in $\mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}}$. Additionally, $\mathsf{NIR}_{\mathsf{fold}}$ accumulated the output committed tuple with another accumulator into a new accumulator for the next step.

*Remark 2.* Note that the multi-instance accumulation scheme in Definition 2 can not achieve a verification complexity sublinear in $\ell$ because the verifier still needs to perform the accumulation for $\{\mathbb{x}^{(k)}\}_{k\in[\ell]}$ over $\mathbb{F}$. This cost can be further optimized by extending the multi-cast reduction $\mathsf{NIR}_{\mathsf{multicast}}$ for both $\ell$ instances and witnesses as discussed in Section 5.3. A more practical–although linear–solution in existing work [34,14] is hashing them into one value, which is concretely better.

### 2.3 Building Blocks

To build the multi-instance accumulation schemes, we (1) propose a novel multi-cast reduction in Section 4.2 and (2) derive a 2-to-1 reduction in Section 5.2 based

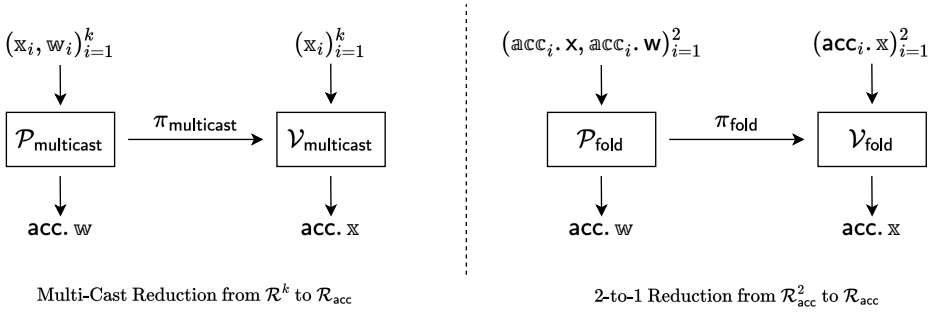


**Fig. 3.** Two components $\mathsf{NIR}_{\mathsf{multicast}}, \mathsf{NIR}_{\mathsf{fold}}$ for multi-instance accumulation schemes.

on the previous techniques in [25,15]. For simplicity, we follow the Interactive Oracle Reduction (IOR) paradigm and present the "compiled" version of these two primitives in this part.

**Multi-cast reduction.** Let $\mathsf{cm}$ be a polynomial commitment scheme, a multi-cast reduction $\mathsf{IOR}^{\mathsf{cm}}_{\mathsf{multicast}}$ compiled with $\mathsf{cm}$ takes as inputs $\ell$ tuples of $\{(\boldsymbol{x}_k, \boldsymbol{w}_k)\}_{k\in[\ell]}$ in $\mathcal{R}^\ell$ and outputs one tuple $(\boldsymbol{x}, \boldsymbol{w})$ in $\mathcal{R}^{\mathsf{cm}}_{\mathsf{acc}}$, where $\boldsymbol{x}_k \in \mathbb{F}^m, \boldsymbol{w}_k \in \mathbb{F}^n,\ \forall k \in [\ell]$. Instead of sending $\ell$ commitments $\{C_k = \mathsf{Commit}(\boldsymbol{w}_k)\}_{k\in[\ell]}$ as in previous solutions [14], the prover of $\mathsf{IOR}^{\mathsf{cm}}_{\mathsf{multicast}}$ first sends a union polynomial commitment $C_\cup$ of the multilinear extension of all witness vectors $\{\boldsymbol{w}_k\}_{k\in[\ell]}$ as

$$\tilde{w}_\cup(\boldsymbol{Y}, \boldsymbol{X}) := \sum_{k\in[\ell]} \tilde{eq}(\mathsf{Bits}(k), \boldsymbol{Y}) \cdot \sum_{i\in[n]} \tilde{eq}(\mathsf{Bits}(i), \boldsymbol{X}) \cdot \boldsymbol{w}_k[i] \in \mathbb{F}^{(<2)}[\boldsymbol{Y}||\boldsymbol{X}].$$

Clearly, the partial evaluation $\tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{X})$ at $\boldsymbol{Y} = \boldsymbol{\tau} \in \mathbb{F}^{\log \ell}$ equals to a multilinear extension $\tilde{w}(\boldsymbol{X}) \in \mathbb{F}^{(<2)}[\boldsymbol{X}]$ of the accumulated witness vector

$$\boldsymbol{w} := \sum_{k\in[\ell]} \tilde{eq}(\mathsf{Bits}(k), \boldsymbol{\tau}) \cdot \boldsymbol{w}_k. \tag{1}$$

Again, the prover sends a polynomial commitment $C$ of $\tilde{w}(\boldsymbol{X})$. To ensure that the vector $\boldsymbol{w}$ is indeed the linear combination of all $\{\boldsymbol{w}_k\}_{k\in[\ell]}$ in terms of $\boldsymbol{\tau}$ as in Equation 1, the verifier only needs to check that $\tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{r}_x) = \tilde{w}(\boldsymbol{r}_x)$ at a random $\boldsymbol{r}_x \in \mathbb{F}^{\log n}$. The soundness error is upper-bounded by $\log n/|\mathbb{F}|$, which is the probability that a malicious prover forges an invalid $\tilde{w}' \in \mathbb{F}^{(<2)}[\boldsymbol{X}]$ happens to equal to $\tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{r}_x)$ at $\boldsymbol{r}_x$. We sketch the protocol as follows:

- The prover sends a commitment $C_\cup$ of $\tilde{w}_\cup(\boldsymbol{Y}, \boldsymbol{X})$.
- The verifier replies a challenge vector $\boldsymbol{\tau} \leftarrow\!\$\ \mathbb{F}^{\log \ell}$.
- The prover sends a commitment $C$ of $\tilde{w}(\boldsymbol{X}) := \tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{r}_x)$.
- The verifier samples $\boldsymbol{r}_x \leftarrow\!\$\ \mathbb{F}^{\log n}$ and queries for evaluations $\tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{r}_x)$ and $\tilde{w}(\boldsymbol{r}_x)$; outputs 'accept' if they are equal.

The reduced relation $\mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}}$ is given below, which has a constant number of commitments (independent of $\ell$):

$$\mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}}(\mathsf{pp}) = \left\{ (C_\cup, C, \boldsymbol{\tau}, \boldsymbol{r}_x; \tilde{w}_\cup(\boldsymbol{Y}, \boldsymbol{X}), \tilde{w}(\boldsymbol{X})) \; : \; \begin{array}{c} C_\cup = \mathsf{Commit}(\mathsf{pp}, \tilde{w}_\cup(\boldsymbol{Y}, \boldsymbol{X})) \\ \wedge C = \mathsf{Commit}(\mathsf{pp}, \tilde{w}(\boldsymbol{X})) \\ \wedge \tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{r}_x) = \tilde{w}(\boldsymbol{r}_x) \end{array} \right\}.$$

*Sublinear complexity.* We can apply the same technique on multiple instances $\{\boldsymbol{x}^{(k)}\}_{k\in\ell}$ to check that $\tilde{x}_\cup(\boldsymbol{\tau}, \boldsymbol{r}_x) = \tilde{x}(\boldsymbol{r}_x)$, where $\tilde{x}$ is the multilinear extension of the accumulated instance vector. As a result, the obtained multi-cast reduction can achieve sublinear-$\ell$ verifier cost.

**2-to-1 reduction.** The construction of 2-to-1 reduction $\mathsf{IOR}_{\mathsf{fold}}^{\mathsf{cm}}$ is relatively trivial, which inputs two instance-witness tuples and outputs one all in the same $\mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}}$. Note that the relation $\mathcal{R}_{\mathsf{acc}}^{\mathsf{cm}}$ can be written as two polynomial evaluation claims $\mathcal{R}_{\mathsf{eval}}^{\mathsf{cm}} \times \mathcal{R}_{\mathsf{eval}}^{\mathsf{cm}}$ where

$$\mathcal{R}_{\mathsf{eval}}^{\mathsf{cm}}(\mathsf{pp}) := \left\{ (C, \{\boldsymbol{x}_i, v_i\}_i; \tilde{p}(\boldsymbol{X})) \; : \; C = \mathsf{Commit}(\mathsf{pp}, \tilde{p}(\boldsymbol{X})) \wedge \tilde{p}(\boldsymbol{x}_i) = v_i) \; \forall i \right\}.$$

Then, we can adopt a similar construction in [15]: a 2-to-1 reduction can be built from an oracle batching reduction $\mathsf{IOR}_{\mathsf{batch}}$ with the following definition:

**Definition 3 (Oracle batching reduction [15]).** *A (committed)* ***oracle batching protocol*** $\mathsf{IOR}_{\mathsf{batch}}^{\mathsf{cm}}$ *is a reduction from* $\mathcal{R}_0^{\mathsf{cm}}$ *to* $\mathcal{R}_{\mathsf{eval}}^{\mathsf{cm}}$*, where*

$$\mathcal{R}_0^{\mathsf{cm}} := \left\{ \{C_i, \boldsymbol{x}_i, r_i\}_{i=1}^2, v; \{\tilde{p}_i(\boldsymbol{X})\}_{i=1}^2 \; : \; \begin{array}{c} C_i = \mathsf{Commit}(\mathsf{pp}, \tilde{p}_i(\boldsymbol{X})) \; \forall i = 1, 2 \\ \wedge r_1 \cdot \tilde{p}_1(\boldsymbol{X}) + r_2 \cdot \tilde{p}_2(\boldsymbol{X}) = v \end{array} \right\}.$$

We sketch the 2-to-1 reduction as follows:

- Run a sum-check protocol to reduce the claims $\tilde{w}_{\cup,i}(\boldsymbol{\tau}, \boldsymbol{r}_x) = v_i \wedge \tilde{w}_i(\boldsymbol{r}_x) = v_i, i = 1, 2$ to

$$\sum_{i=1}^{2} r_i \cdot \tilde{w}_{\cup,i}(\boldsymbol{\tau}, \boldsymbol{r}_x) = v \tag{2}$$

$$\sum_{i=1}^{2} r_i \cdot \tilde{w}_i(\boldsymbol{r}_x) = v \tag{3}$$

- Run $\mathsf{IOR}_{\mathsf{batch}}^{\mathsf{cm}}$ protocols separately for the statements satisfying Equation 2, 3.

We additionally require the $\mathsf{IOR}_{\mathsf{batch}}^{\mathsf{cm}}$ to satisfy a *succinctness* property such that the proof it generates is sublinear to the length of $\tilde{p}_i(\boldsymbol{X})$'s. This property ensures that the constructed accumulation scheme has a verifier complexity sublinear in $\ell$. In Section 5.3, we investigate state-of-the-art PCS and analyze their theoretical performance in $\mathsf{NIR}_{\mathsf{batch}}^{\mathsf{cm}}$. Then in Section 6, we provide a concrete instantiation of $\mathsf{IOR}_{\mathsf{fold}}^{\mathsf{cm}}$ based on linear error correcting codes [15], which derives an efficient accumulation scheme with linear-time prover and plausible post quantum security.

*Remark 3.* For the two primitives above, we omit the additional constraint involved in $\mathcal{R}$ for simplicity. In the full definitions, we assume the instance and witness $\mathcal{R}$ also satisfies a constraint $F(\boldsymbol{x}, \boldsymbol{w}) = 0$, where $F : \mathbb{F}^{m+n}$ $rightarrow\mathbb{F}$ is a $d$-degree algebraic map. We show that this assumption is already sufficient to capture the Special-sound protocol in Section 5.1, which is a fundamental primitive with many applications [14].

# 3 Preliminaries

## 3.1 Notations

In this paper, we use $\lambda$ to denote the security parameter. Accordingly, $\mathsf{negl}(\lambda)$ denotes a function that is negligible in $\lambda$. Let $\mathbb{F}$ denote a finite field, e.g., $\mathbb{F}$ is a prime field for a large prime $p$. The vector is denoted as $\boldsymbol{a} \in \mathbb{F}^n$ with $a_1, ..., a_n \in \mathbb{F}$, where the $i$-th element of $\boldsymbol{a}$ is referred to as $a_i$ or $\boldsymbol{a}[i]$ when the element is not specified with a concrete value. For notations of sets, let $[n]$ indicate the set $\{0, ..., n-1\} \subseteq \mathbb{N}$, and also let $\{a_i\}_{i\in[n]}$ be a short-hand for $\{a_0, ..., a_{n-1}\}$. For the algorithm notations, we use "PPT algorithms" to refer to Probabilistic Polynomial Time Algorithms. For a finite set $S$, let $x \leftarrow\!\$\, S$ denote sampling $x$ from $S$ uniformly at random.

*Relations and oracles.* For a non-deterministic polynomial time (NP) indexed relation $\mathcal{R}$ parameterized over public parameters $\mathsf{pp}$ (e.g., including the field $\mathbb{F}$), it consists of a triple of index $\mathtt{i}$, instance $\mathtt{x}$, and witness $\mathtt{w}$ (the secret inputs). We denote $\mathcal{R}$ as an oracle relation if it has an additional oracle string $\mathtt{y}$ that consists of several oracles. For an indexed oracle relation $\mathcal{R}(\mathsf{pp}) = \{(\mathtt{i}, \mathtt{x}, \mathtt{y}, \mathtt{w})\}$, let $\mathcal{L}(\mathcal{R}) = \{(\mathtt{i}, \mathtt{x}, \mathtt{y}) : \exists \mathtt{w}, s.t., (\mathtt{i}, \mathtt{x}, \mathtt{y}, \mathtt{w}) \in \mathcal{R}\}$ be the language induced by $\mathcal{R}$ ($\mathsf{pp}$ is sometimes written implicitly). To represent multi-instance relations, we define for $\ell \in \mathbb{N}$ instances in a relation $\mathcal{R}$ as:

$$\mathcal{R}^\ell := \{\mathtt{i}, (\mathtt{x}_1, \ldots, \mathtt{x}_\ell), (\mathtt{y}_1, \ldots, \mathtt{y}_\ell), (\mathtt{w}_1, \ldots, \mathtt{w}_\ell) : \ \forall i \in [\ell], (\mathtt{i}, \mathtt{x}_i, \mathtt{y}_i, \mathtt{w}_i) \in \mathcal{R}\}.$$

Let $\mathbb{O}(\mathtt{i}, \mathtt{x}) := \{\mathtt{y} : (\mathtt{i}, \mathtt{x}, \mathtt{y}) \in \mathcal{L}(\mathcal{R})\}$ be the set of oracles in the relation induced by the instance. To cover the code-based polynomial oracles, we define the distance of a statement $(\mathtt{i}, \mathtt{x}, \mathtt{y})$ from the relation $\mathcal{R}$ to be $\Delta_{\mathcal{R}}(\mathtt{i}, \mathtt{x}, \mathtt{y}) := \mathsf{min}_{c\in\mathbb{O}(\mathtt{i},\mathtt{x})}\Delta(\mathtt{y}, c)$, where $\Delta$ is the relative Hamming distance. More details about the code-based setting can be found in Appendix A.1.

## 3.2 Multilinear Extensions and Sum-check Protocols

*Multilinear extensions.* Let $\tilde{f}(\cdot) : \mathbb{F}^n \to \mathbb{F}$ be a *multivariate polynomial* with $n$ input elements over $\mathbb{F}$, its total degree $\mathsf{deg}(\tilde{f}) \in \mathbb{N}$ is defined as the maximum degree over all monomials. A multivariate polynomial is a *multilinear* polynomial if the degree of each variable is at most one. Next, we denote the multilinear extension of a vector $\boldsymbol{f}$ as a unique multilinear $n$-variate polynomial $\tilde{f}(\cdot) : \mathbb{F}^n \to \mathbb{F}$ such that $\tilde{f}(\mathsf{Bits}(i)) = \boldsymbol{f}[i]$ for all $i \in n$, where the binary representation $\mathsf{Bits}(i)$

belongs to the boolean hypercube $B_{\log n} = \{0,1\}^{\log n}$. The multilinear extension can be computed as $\tilde{f}(\boldsymbol{X}) = \sum_{i\in[n]} \boldsymbol{f}[i] \cdot \widetilde{eq}(\boldsymbol{X}, \mathsf{Bits}(i))$, where $\widetilde{eq}(\boldsymbol{X}, \mathsf{Bits}(i))$ is an extension of the equality function $eq(\cdot)$. For simplicity, we sometimes use the integer index instead of a binary vector to represent the equality function as $\widetilde{eq}_i(\boldsymbol{X}) := \widetilde{eq}(\boldsymbol{X}, \mathsf{Bits}(i)) = \prod_{j=0}^{\log n-1}(\mathsf{Bits}(i)[j] \cdot X_j + (1-\mathsf{Bits}(i)[j])(1-X_j)), i \in [n]$. Denote $\mathbb{F}^{(<d)}[\boldsymbol{X}]$ where $\boldsymbol{X} \in \mathbb{F}^n$ as $\mathcal{F}_n^{(<d)}$ for short.

*Sum-check protocol.* The sumcheck protocol is an interactive proof proposed by Lund et al. [37]. Assume $\tilde{f}(\boldsymbol{X})$ as an $n$-variate polynomial with the maximum individual degree of $d$. The prover aims to convince the verifier that $\mathsf{sum} = \sum_{\boldsymbol{x}\in B_n} \tilde{f}(\boldsymbol{X})$. To achieve this, the prover and verifier participate in an $n$-round interactive proof. At each $i \in [1,n]$ round, the prover computes an intermediate univariate polynomial as

$$f_i(X) = \sum_{x_{i+1},\ldots,x_n \in B_{n-i}} \tilde{f}(r_1,\ldots,r_{i-1},X,x_{i+1},\ldots,x_n), \tag{4}$$

the verifier checks $\mathsf{sum} = f_1(0) + f_1(1)$ for $i = 1$ and checks $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$ else. At the final step, the verifier checks if $f_n(r_n) = \tilde{f}(r_1,\ldots,r_n)$. We mention that the interactive sum-check protocol satisfies both completeness and soundness according to [21].

**Lemma 1 (Ore-DeMillo-Lipton-Schwartz-Zippel lemma).** *Assume an $n$-variate non-zero polynomial $\tilde{f} \in \mathcal{F}_{\log n}^{(<d)}$, then it holds that*

$$\mathsf{Pr}_{\boldsymbol{r}\leftarrow\$\mathbb{F}^n}[\tilde{f}(\boldsymbol{r}) = 0] \leq \frac{n\cdot(d-1)}{|\mathbb{F}|} \tag{5}$$

### 3.3 Interactive Oracle Reductions

Interactive Oracle Reduction (IOR) [6,20] is an information-theoretic proof system that generalizes the Interactive Oracle Proofs (IOPs) to the language of reduction. The key difference is that an IOR verifier may output claims about proof strings (oracles) without fully reading them. In this paper, we consider public-coin polynomial IORs from relation $\mathcal{R}$ to $\mathcal{R}'$ with $\mu$ rounds of interactions between a prover and a verifier. In the $i$-th round, $i \in [\mu]$, the verifier sends a challenge to the prover, and the prover replies either a full message or a polynomial oracle that the verifier can query.

**Definition 4 (Interactive oracle reductions [20]).** *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 a deterministic algorithm that receives as input an index $\mathbb{i}$ for relation $\mathcal{R}$. It outputs a short index $\iota$, index string $\mathbb{I}$ and a new index $\mathbb{i}'$ for $\mathcal{R}'$.*

- *The prover $\mathcal{P}$ is an interactive algorithm that receives as input the index $\mathbb{i}$, instance $\mathbb{x}$, oracle string $\mathbb{y}$ and witness $\mathbb{w}$. It engages in $\mu$ rounds of interaction. In the $i$-th round, it either sends a proof string $\pi_i$ as an oracle or a non-oracle string $m$, then receives a challenge $r_i$.*
- *The verifier $\mathcal{V}$ is an interactive algorithm that receives as input the short index $\iota$, the instance $\mathbb{x}$ and query access to $\mathbb{y}, \mathbb{I}$. It engages in $\mu$ rounds of interactions. In each round, it receives oracle access to a proof string $\pi_i$ or a non-oracle string $m$, then sends a uniformly random challenge $r_i$ sampled from challenge space $\{0,1\}^{\mathsf{r}_i}, \mathsf{r}_i \in \mathbb{N}$. After the interaction, the verifier queries the oracles including $\mathbb{I}, \mathbb{y}, \boldsymbol{\pi}$.*
- *At the end of the protocol, the verifier either rejects or outputs a new statement $(\mathbb{x}', \mathbb{y}')$. The prover outputs a new witness $\mathbb{w}'$ such that $(\mathbb{i}', \mathbb{x}', \mathbb{y}', \mathbb{w}') \in \mathcal{R}'$.*

An IOR is assumed to satisfy perfect completeness and soundness with proximity radius $\delta^* \in [0,1]$ (to cover proximity proof) as defined in Definitions 15 and 16 in Appendix A.4. To ensure Fiat–Shamir security of the corresponding non-interactive reduction in the ROM (i.e., against state-restoration attack), we highlight a stronger security notion for IORs as the round-by-round (RBR) knowledge soundness proposed in [7]. The formal definitions of RBR knowledge soundness can be referred to Definition 18 in Appendix A.4.

## 4 Technique: Multi-Cast Reductions

In this section, we introduce the concept of multi-cast reduction, which serves as the core building block for our multi-instance accumulation scheme described in Section 5. To begin, we first review existing multi-cast solutions in Section 4.1, where the verifier is required to compute a random linear combination over $O(\ell)$ homomorphic commitments, i.e., $\mathsf{CRC}$. Subsequently, in Section 4.2, we present a new approach that reduces the complexity from $O(\ell)$ to $O(1)$ by leveraging PCS, thereby achieving significant performance improvements over previous works when applied to accumulation schemes.

### 4.1 Existing Solutions

A many-to-one reduction is a key component in accumulation schemes [20,15]. Informally, it enables the "batching" (we avoid using "accumulation" due to the repetition in Section 5) of $\ell$ input statement-witness tuples into a single new one that satisfies the same relation, allowing the verifier to check the validity of the original $\ell$ instances at the cost of verifying only one. In this section, we generalize the many-to-one reduction to what we call a *multi-cast reduction*, which allows the prover to convince the verifier that a reduction from a multi-instance relation $\mathcal{R}_1^\ell$ to a different relation $\mathcal{R}_2$ is correct. In other words, this reduction enables the prover to batch multiple instances and, at the same time, cast them to an instance satisfying a different relation.

To illustrate, we first consider a tuple $\{\mathbb{i}, (\mathbb{x}^{(i)}, \mathbb{w}^{(i)})\}_{i\in[\ell]}$ satisfying the multi-instance relation $\mathcal{R}_L^\ell$ with respect to a linear map $L : \mathbb{F}^n \to \mathbb{F}$, where each

$\mathbb{x}^{(i)} = (\boldsymbol{x}^{(i)} \in \mathbb{F}^{m_i}, y^{(i)})$, and each $\mathbb{w}^{(i)} = \boldsymbol{w}^{(i)} \in \mathbb{F}^n$, where $y^{(i)} = L(\boldsymbol{x}^{(i)}, \boldsymbol{w}^{(i)})$. The naive approach would require the prover to publish all $\{\boldsymbol{w}^{(i)}\}_{i\in[\ell]}$ in plaintext, with the verifier computing the random linear combination among them. This results in an inefficient protocol with $O(\ell \cdot n)$ proof size and $O(\ell \cdot n)$ verification cost. A more practical approach, used in compressed Sigma protocols [3,30] (known as amortization), is to first commit to the witnesses and then open them together. We describe the reduction in a 3-move protocol as follows, where $\mathsf{Commit}$ is an algorithm from the homomorphic commitment scheme ($\mathsf{Setup}$, $\mathsf{Commit}, \mathsf{Open}$), such as the Pedersen commitment scheme. For simplicity, we omit the randomness used for hiding.

- The prover computes $C^{(i)} = \mathsf{Commit}(\mathsf{ck}, \boldsymbol{w}^{(i)})$ for each $i$ and sends $\{C^{(i)}\}_{i\in[\ell]}$ to the verifier.
- The verifier randomly samples a challenge $\alpha \leftarrow\!\!\$\ \mathbb{F}$ as a response.
- The prover outputs the new witness $\mathbb{w} := \boldsymbol{w} = \sum_{i=0}^{\ell-1} \alpha^i \cdot \boldsymbol{w}^{(i)}$. The verifier outputs the same index $\mathbb{i}$, the new instance $\mathbb{x} := (\boldsymbol{x}, y, C)$ where $\boldsymbol{x} = \sum_{i=0}^{\ell-1} \alpha^i \cdot \boldsymbol{x}^{(i)}$, $y = \sum_{i=0}^{\ell-1} y^{(i)}$, $C = \sum_{i=0}^{\ell-1} \alpha^i \cdot C^{(i)}$.

To demonstrate the validity of the new instance $\mathbb{x}$, the verifier checks (1) $\mathsf{Open}(\mathsf{ck}, \boldsymbol{w}) = C$, and (2) $L(\boldsymbol{w}) = y = \sum_{i=0}^{\ell-1} \alpha^i \cdot y^{(i)}$. Since $L$ is a linear map, it holds that $L(\sum_{i=0}^{\ell-1} \alpha^i \cdot \boldsymbol{x}^{(i)}, \sum_{i=0}^{\ell-1} \alpha^i \cdot \boldsymbol{w}^{(i)}) = \sum_{i=0}^{\ell-1} \alpha^i \cdot L(\boldsymbol{x}^{(i)}, \boldsymbol{w}^{(i)})$ as long as $(\mathbb{x}^{(i)}, \mathbb{w}^{(i)})$ are valid. We highlight the security of this approach in Lemma 2. The verifier only needs to compute an $O(\ell)$-sized linear combination for all $C^{(i)}$'s, where the commitment is assumed to be succinct.

**Lemma 2.** *Given that the underlying homomorphic commitment scheme is binding, the above 3-move interactive protocol is a reduction of knowledge from the multi-instance relation $(\mathcal{R}_L)^\ell$ to an indexed committed relation $\mathcal{R}_L^{\mathsf{cm}}$ as below:*

$$\mathcal{R}_L^{\mathsf{cm}} = \{(\mathbb{i}, \mathbb{x} = (\boldsymbol{x}, y, C), \mathbb{y} = \bot, \mathbb{w} = \boldsymbol{w}) : L(\boldsymbol{w}) = y \wedge \mathsf{Open}(\mathsf{ck}, \boldsymbol{w}) = C\}\,.$$

*Efficiency Problem.* The above approach remains impractical for constructing recursive SNARKs due to two main issues: First, the verifier needs to compute the linear combination of commitments, i.e., $O(\ell)$-sized $\mathsf{CRC}$ operations, which is circuit-unfriendly as we discussed in Section 1. Second, the target relation $\mathcal{R}_L$ only considers a linear map, which is too simple for practical use.

### 4.2 New Solution based on Polynomial Evaluations

To address the above issue, we propose a new protocol for multi-cast reductions that checks the witness batching with polynomial evaluations, which significantly reduces the cost for $\mathsf{CRC}$ on the verifier side. For simplicity and generality, we present our scheme under the IOR model; therefore, only the denotations of polynomial oracles are used in the protocol rather than polynomial commitments. Later in Section 5.3, we discuss how to compile the oracles with different PCS candidates. We start with a more complicated high-degree relation as $\mathcal{R}_F$ below,

$$\mathcal{R}_F(\mathsf{pp}) = \{(\mathbb{i}, \mathbb{x} = \boldsymbol{x}, \mathbb{w} = \boldsymbol{w}) : F(\boldsymbol{x}, \boldsymbol{w}) = 0\}\,,$$

where $F(\boldsymbol{x}, \boldsymbol{w}) := \sum_{j=0}^{d} f_j^F(\boldsymbol{x}, \boldsymbol{w})$ is a function with each $f_j^F$ as a homogeneous degree-$d$ algebraic map that outputs a zero element in $\mathbb{F}$, $e \in \mathbb{F}$. Our target is to propose an IOR from the multi-instance relation $\mathcal{R}_F^\ell$ to a new oracle relation $\mathcal{R}_{\mathsf{acc}}$ defined later.

**Multi-cast IOR.** Consider inputs as a tuple $\{\mathtt{i}, (\mathtt{x}^{(i)}, \mathtt{w}^{(i)})\}_{i\in[\ell]}$ satisfying $\mathcal{R}_F^\ell$, the IOR outputs a new tuple $(\mathtt{i}, \mathtt{x}, \mathtt{y}, \mathtt{w})$ satisfying an oracle indexed relation $\mathcal{R}_{\mathsf{acc}}$, where $\mathtt{y}$ is the set of oracles. We first interpret each witness vector $\boldsymbol{w}^{(i)} \in \mathbb{F}^n$ as a function mapping a boolean hypercube to the field $\mathbb{F}$. For simplicity, we assume $\ell, n$ are both powers of 2. Then each witness vector can be represented as a multilinear polynomial as $\tilde{w}^{(i)}(\boldsymbol{X})$ by an extension over $\boldsymbol{X} \in B_{\log n}$, i.e., $\tilde{w}^{(i)}(\mathsf{Bits}[j]) = \boldsymbol{w}^{(i)}[j]$ for all $j \in [n]$. We first present a step-by-step description of our multi-cast reduction as an IOR in Figure 4 with prover inputs as an index $\mathtt{i}$, $\ell$ instances $\{\mathtt{x}^{(i)}\}_{i\in\ell}$, and $\ell$ witnesses $\{\mathtt{w}^{(i)}\}_{i\in\ell}$ for the prover. The verifier takes inputs as a short index $\iota$, an oracle $\mathbb{I}$, and $\ell$ instances $\{\mathtt{x}^{(i)}\}_{i\in\ell}$, where $(\iota, \mathbb{I}, \mathtt{i}') \leftarrow \mathcal{I}(\mathtt{i})$, $\mathtt{i}' = \mathtt{i}$. We explain the technical details afterward.

Generally speaking, the protocols have two main tasks: (1) showing that $\{\mathtt{w}^{(i)}\}_{i\in[\ell]}$ are correctly batched into one $\mathtt{w} = \boldsymbol{w}$, (2) deriving a correct $e$ as the image of the function $F$ with input $\boldsymbol{w}$. For task 1, the prover first computes the union multilinear polynomial $\tilde{w}_\cup(\boldsymbol{Y}, \boldsymbol{X})$, where $\tilde{w}_\cup(\mathsf{Bits}(i), \boldsymbol{X}) = \tilde{w}^{(i)}(\boldsymbol{X})$ for all $i \in [\ell]$, and sends its oracle $[\![\tilde{w}_\cup]\!]$ at step 1,2. Then he proves that $\tilde{w}$ *partially evaluates* to $\tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{X})$ at $\boldsymbol{Y} = \boldsymbol{\tau}$, where $\tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{X})$ is exactly the batched witness under a random linear combination with $\{\tilde{eq}_{i-1}(\boldsymbol{\tau})\}_{i\in[\ell]}$. However, there is no construction of polynomial oracles for answering such "partial evaluation". Alternatively, the prover sends another oracle $[\![\tilde{w}]\!]$ at step 9 and claims that $\tilde{w}(\boldsymbol{X})$ is honestly computed from $\tilde{w}_\cup(\boldsymbol{Y}, \boldsymbol{X})$. To check this claim, the verifier samples another random vector $\boldsymbol{r}_x$ at step 10 and queries for $\tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{r}_x)$ and $\tilde{w}(\boldsymbol{r}_x)$. Clearly, when the two evaluations agree at a randomly sampled $\boldsymbol{r}_x$, then with a high probability $\tilde{w}(\boldsymbol{X}) = \tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{X})$ as per Schwartz-Zippel lemma [39].

For task 2, we first establish the following equality about the constraint $F$ on the new instance-witness tuple $(\mathtt{x}, \mathtt{w})$, which relies on the following corollary:

**Corollary 1 (adopted from [25]).** *A multilinear equality polynomial $\tilde{eq}_i(\boldsymbol{b})$ defined over the Boolean hypercube $B_{\log \ell}$ satisfies the following facts:*

- $(\tilde{eq}_i(\boldsymbol{b}))^d = \tilde{eq}_i(\boldsymbol{b})$ *holds for all* $d \in \mathbb{N}$ *and* $\boldsymbol{b} \in B_{\log \ell}$,
- *for all* $\boldsymbol{b} \in B_{\log \ell}$, $\tilde{eq}_i(\boldsymbol{b}) \cdot \tilde{eq}_j(\boldsymbol{b}) = 0$ *holds if and only if* $i \neq j \in [\ell]$.

*Proof.* Note that $\tilde{eq}_i(\boldsymbol{X}) := \prod_{j=1}^{\ell}(\mathsf{Bits}(i)[j] \cdot X_j + (1 - \mathsf{Bits}(i)[j])(1 - X_j))$. Hence, for a multivariate polynomial $g(\boldsymbol{X}) = \tilde{eq}_i(\boldsymbol{X})^d - \tilde{eq}_i(\boldsymbol{X})$, it vanishes to zero on all $\boldsymbol{b} \in B_{\log \ell}$, which proves fact 1. For the second fact, since $\tilde{eq}_i(\boldsymbol{b}) = 1$ when $\mathsf{Bits}(i) = \boldsymbol{b} \in B_{\log \ell}$, then $\tilde{eq}_i(\boldsymbol{b}) = \tilde{eq}_j(\boldsymbol{b})$ if and only if $i = j$ for all $\boldsymbol{b} \in B_{\log \ell}$. □

Given above, we observe that the polynomial $F(\tilde{\boldsymbol{x}}(\boldsymbol{Y}), \tilde{\boldsymbol{w}}(\boldsymbol{Y}))$ derived from constraint $F$ evaluates to zero for all $\boldsymbol{y} \in B_{\log \ell}$, where the two vectors of poly-

$\mathcal{P}$'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: $\tilde{w}_{\cup}(\boldsymbol{Y}, \boldsymbol{X}) := \sum_{i\in[\ell]} \tilde{eq}_{i-1}(\boldsymbol{Y}) \cdot \tilde{w}^{(i)}(\boldsymbol{X})$

2 : $\mathcal{P} \to \mathcal{V}$ : oracle message $[\![\tilde{w}_{\cup}]\!]$

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 per Equation 6

5 : $\mathcal{P}$ computes $G(\boldsymbol{Y})$ as per Equation 8

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

7 : $\mathcal{V}$ checks: $G_1(0) + G_1(1) = 0 \wedge G_{i+1}(0) + G_{i+1}(1) = G_i(\tau_i),\ \forall i \in [\log \ell - 1]$

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

9 : $\mathcal{P} \to \mathcal{V}$ : oracle message $[\![\tilde{w}]\!]$

10 : $\mathcal{V}$ samples: $\boldsymbol{r}_x \leftarrow\$ \mathbb{F}^{\log n}$

**Output phase.**

11 : Define $\boldsymbol{x} = \sum_{i\in[\ell]} \tilde{eq}_{i-1}(\boldsymbol{\tau}) \cdot \boldsymbol{x}^{(i)}$

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

13 : $\mathcal{V}$ outputs: $\mathbb{x} = (\boldsymbol{x}, \boldsymbol{\tau}, \boldsymbol{r}_x, e), \mathbb{y} = ([\![\tilde{w}_{\cup}]\!], [\![\tilde{w}]\!])$

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

nomials are defined as (step 4):

$$\tilde{\boldsymbol{x}}(\boldsymbol{Y}) := \sum_{i\in[\ell]} \tilde{eq}_{i-1}(\boldsymbol{Y}) \cdot \boldsymbol{x}^{(i)} \in (\mathcal{F}_{\log n}^{(<2)})^m, \tag{6}$$

$$\tilde{\boldsymbol{w}}(\boldsymbol{Y}) := \sum_{i\in[\ell]} \tilde{eq}_{i-1}(\boldsymbol{Y}) \cdot \boldsymbol{w}^{(i)} \in (\mathcal{F}_{\log n}^{(<2)})^n \tag{7}$$

Accordingly, we can apply a common trick [21] to reduce this zero-check problem to a sum-check problem. Simply, given a randomly sampled vector $\boldsymbol{r}_y$, the following polynomial with maximum individual degree $d+1$ satisfies the sum-check relation $\sum_{\boldsymbol{y}\in B_{\log \ell}} G(\boldsymbol{y}) = 0$ (step 5), where

$$G(\boldsymbol{Y}) := F(\tilde{\boldsymbol{x}}(\boldsymbol{Y}), \tilde{\boldsymbol{w}}(\boldsymbol{Y})) \cdot \tilde{eq}(\boldsymbol{Y}, \boldsymbol{r}_y). \tag{8}$$

The prover and verifier runs a $\log \ell$-round sum-check protocol at step 6. At the final step of the sum-check reduction, the verifier outputs an evaluation claim

$G_{\log \ell}(\tau_{\log \ell}) = G(\boldsymbol{\tau}) = F(\tilde{\boldsymbol{x}}(\boldsymbol{\tau}), \tilde{\boldsymbol{w}}(\boldsymbol{\tau})) \cdot \tilde{eq}(\boldsymbol{\tau}, \boldsymbol{r}_y)$, which is equivalent to the following constraint consistent with $\mathcal{R}_F$:

$$F(\boldsymbol{x}, \boldsymbol{w}) = e \tag{9}$$

where $\boldsymbol{x} = \tilde{\boldsymbol{x}}(\boldsymbol{\tau}), \boldsymbol{w} = \tilde{\boldsymbol{w}}(\boldsymbol{\tau})$ are batched vector, and $e = G_{\log \ell}(\tau_{\log \ell}) \cdot \tilde{eq}^{-1}(\boldsymbol{\tau}, \boldsymbol{r}_y)$. Since the virtual oracle $[\![G]\!]$ can be derived from $[\![\tilde{\mathbb{w}}]\!]$, the reduced oracle string $\mathbb{y}$ does not include $[\![G]\!]$.

At the output phase, the verifier outputs the reduced instance at step 14, which satisfies the $\mathcal{R}_{\mathsf{acc}}$ relation as below

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

For easier discussion, we introduce an intermediate value $v = \tilde{w}_\cup(\boldsymbol{\tau}, \boldsymbol{r}_x) = \tilde{w}(\boldsymbol{r}_x)$, and rewrite the relation $\mathcal{R}_{\mathsf{acc}}$ into a ternary one as $\mathcal{R}_{\mathsf{eval}} \times \mathcal{R}_{\mathsf{eval}} \times \mathcal{R}_F$ where

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{\tau}, \boldsymbol{r}_x, v), \mathbb{y} = [\![\tilde{\mathbb{w}}_\cup]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}} \tag{10}$$

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{r}_x, v), \mathbb{y} = [\![\tilde{\mathbb{w}}]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}} \tag{11}$$

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{x}, \boldsymbol{r}_x, v), \mathbb{y} = \bot, \mathbb{w} = \boldsymbol{w}) \in \mathcal{R}_F \tag{12}$$

Here, we define $\mathcal{R}_{\mathsf{eval}}$ as an oracle relation for a list of multilinear evaluation claims for generality

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

**Lemma 3.** *The protocol* $\mathsf{IOR}_{\mathsf{cast}}$ *is an interactive oracle reduction from the multi-instance relation* $\mathcal{R}_F^\ell$ *to an indexed oracle relation* $\mathcal{R}_{\mathsf{acc}}$*. It satisfies completeness and RBR knowledge soundness. The complexity is measured as follows:*

- *The protocol has* $\log \ell + 2$ *rounds.*
- *The proof length includes* $d \log \ell$ *field elements for the sum-check protocol. The oracle proof length is* $\ell n + n$ *field elements for* $[\![\tilde{w}_\cup]\!]$ *and* $[\![\tilde{w}]\!]$*.*
- *The verifier makes no polynomial queries.*
- *The verifier time is dominated by the* $O(\ell \cdot n)$ *Field operations for computing* $\mathbb{x}$*.*
- *The reduced relation* $\mathcal{R}_{\mathsf{acc}}$ *only contains* ***two polynomial oracles****.*

*Suppose that the polynomial commitment scheme* $\mathsf{PC}$ *satisfies extractability, the Fiat-Shamir transformed* $\mathsf{FS}[\mathsf{IOR}_{\mathsf{acc}}^{\mathsf{pc}}]$ *is a non-interactive reduction of knowledge under the random oracle model.*

The proof of Lemma 3 is given in Appendix C.1.

## 5 Multi-Instance Accumulation Scheme

In this section, we propose a novel accumulation scheme specialized for multiple predicate instances. We first propose a multi-cast reduction for special sound protocols in Section 5.1, which captures a wide range of relations as per [14,38]. Then in Section 5.2, we construct a multi-instance accumulation scheme by combining the multi-cast reduction with the 2-to-1 reduction. Finally, we discuss the concrete instantiations of our accumulation scheme on different PCS. We also expand the recipe for candidate PCS by covering code-based ones [44,1] as long as they provide a so-called *oracle batching succinctness*.

### 5.1 Multi-Cast Reductions for SPS

**Recap of Special-Sound Protocols.** We adopt the denotations in Protostar and Protogalaxy [14,25] as follows. Define a $(2\mu-1)$-move special-sound protocol $\Pi_{\mathsf{sps}}$ for relation $\mathcal{R}$ according to Definition 20, with verifier degree $d$, output length $m$ and two PPT algorithms $\mathsf{P}_{\mathsf{sps}}, \mathsf{V}_{\mathsf{sps}}$. Let public parameters $\mathsf{pp}$ contain $\mu, m, d$, public input instance be $\mathbb{x} = \boldsymbol{x}$, private input witness be $\mathbb{w} = \boldsymbol{w}$. At each round $i \in [\mu - 1]$, the prover runs $\mathsf{P}_{\mathsf{sps}}$ to compute the next message $\boldsymbol{m}_i \in \mathbb{F}^\mu$ based on previous information including instance vector $\boldsymbol{x}$, witness vector $\boldsymbol{w}$ and transcripts $\{\boldsymbol{m}_j, r_j\}_{j\in[i]}$ in previous $i$ rounds (set as empty if $i = 0$). After receiving the $i$-th round message from the prover, the verifier randomly samples a challenge $r_i$ as a response. Then the two parties repeat the above procedure until $i = \mu$. After receiving the final message $\boldsymbol{m}_\mu$, the verifier runs the algorithm $\mathsf{V}_{\mathsf{sps}}$ for checking the validity of the witness $\mathsf{w}$. Since $\mathsf{V}_{\mathsf{sps}}$ is an algebraic map with maximum degree $d$, it can be further written as a sum of $d$ homogeneous algebraic maps $f_j^{\mathsf{V}_{\mathsf{sps}}}$, each with degree of $j$:

$$\mathsf{V}_{\mathsf{sps}}(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i\in[\mu]}, \boldsymbol{r}) := \sum_{j=0}^{d} f_j^{\mathsf{V}_{\mathsf{sps}}}(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i\in[\mu]}, \boldsymbol{r}) = \boldsymbol{0}^\nu, \tag{13}$$

where $\boldsymbol{r} = \{r_i\}_{i\in[\mu-1]}$. To further reduce the $\nu$ degree-$d$ checks in the verification into one equation[5], the verifier additionally samples a challenge $\alpha \leftarrow\!\!\$\ \mathbb{F}$ and the prover responds with a vector $\boldsymbol{\alpha} = (\alpha, \alpha^2, \alpha^4, \ldots, \alpha^{2^{\log\nu-1}})$ (assume $\nu$ is a power of 2). For $j \in [\nu]$, let $S \subset \{0, \ldots, \log\nu - 1\}$ be the set such that $j = \sum_{k\in S} 2^k + 1$. The verifier derives $\alpha^j = \mathsf{pow}_j(\boldsymbol{\alpha})$, where $\mathsf{pow}_j$ is a $t$-variate polynomial $\mathsf{pow}_j(\boldsymbol{X}) = \prod_{k\in S} X_k$. Accordingly, the prover and verifier agree on a linearly combined map with degree $D = d + \log\nu$ as follows,

$$F(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i\in[\mu]}, \boldsymbol{r}, \boldsymbol{\alpha}) = \sum_{j\in[\nu]} \mathsf{pow}_j(\boldsymbol{\alpha}) \cdot \mathsf{V}_{\mathsf{sps}}(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i=1}^{\mu}, \boldsymbol{r})[j] = 0, \tag{14}$$

where $\mathsf{V}_{\mathsf{sps}}(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i\in[\mu]}, \boldsymbol{r})[j]$ is the $j$-th equation checked by $\mathsf{V}_{\mathsf{sps}}$. We denote the transformed $(2\mu + 1)$-move protocol in terms of $F$ as $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ and provide the step-by-step description as follows.

[5] Note that this reduction does not improve the performance for $\Pi_{\mathsf{sps}}$, but optimizes accumulation schemes based on it.

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

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

1 : For $i \in [\mu]$

$a$ : $\boldsymbol{m}_i \leftarrow \mathsf{P}_{\mathsf{sps}}(\boldsymbol{x}, \boldsymbol{w}, \{\boldsymbol{m}_j, r_j\}_{j=1}^{i-1})$

$b$ : $\mathcal{P} \rightarrow \mathcal{V} : \boldsymbol{m}_i$

$c$ : $\mathcal{V} \rightarrow \mathcal{P} : r_i \leftarrow\!\!\$\ \mathbb{F}$

2 : Set $\alpha := r_\mu$

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

4 : $\mathcal{P} \rightarrow \mathcal{V} : \boldsymbol{\alpha}$

5 : $\mathcal{V}$ checks that

$a$ : $F(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i\in[\mu]}, \boldsymbol{r}, \boldsymbol{\alpha}) = 0$

$b$ : $\alpha_0 = \alpha$

$c$ : $\alpha_{i+1} = \alpha_i^2 \ \ \forall i \in [\log m - 1]$

**Fig. 5.** Transformed Special-Sound Protocol $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ for relation $\mathcal{R}$

**Lemma 4.** *Given that the* $(2\mu+1)$*-move interactive protocol* $\Pi_{\mathsf{sps}}$ *in terms of polynomial time algorithms* $(\mathsf{P}_{\mathsf{sps}}, \mathsf{V}_{\mathsf{sps}})$ *is* $(k_0, \ldots, k_{\mu-1})$*-special-sound for relation* $\mathcal{R}$*, then the* $(2\mu+3)$*-move transformed protocol* $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ *in terms of* $(\mathsf{P}_{\mathsf{sps}}, F)$ *is* $(k_0, \ldots, k_{\mu-1}, \nu+1)$*-special-sound for relation* $\mathcal{R}$*.*

*Proof Sketch.* Given an extractor $\mathcal{E}_{\mathsf{sps}}$ for $\Pi_{\mathsf{sps}}$ and $(k_0, \ldots, k_{\mu-1}, \nu+1)$-tree of transcripts, we can build an extractor $\mathcal{E}_{\mathsf{CV}}$ for $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ under the same relation. Typically, the extractor $\mathcal{E}_{\mathsf{CV}}$ can invoke $\mathcal{E}_{\mathsf{sps}}$ to extract the witness from the depth-$(\mu)$ transcript sub-tree. To demonstrate the validity of the extracted witness, we notice that $F(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i\in[\mu]}, \boldsymbol{r}, \boldsymbol{\alpha})$ with degree $\nu$ equals zero at $\nu+1$ distinct points. Therefore, the original $\mathsf{V}_{\mathsf{sps}}$ must output a zero vector[6].

**Applying multi-cast reductions.** Although the SPS protocol $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ is not an IOR, we can encapsulate its transcript as a tuple $(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathbb{w})$ satisfying the following relation.

$$\mathcal{R}_{\mathsf{sps}} = \left\{ \begin{array}{c} \mathbb{i}, \mathbb{x} = (\boldsymbol{x}, \boldsymbol{r}, \boldsymbol{\alpha}), \\ \mathbb{y} = \bot, \mathbb{w} = \{\boldsymbol{m}_i\}_{i\in[\mu]} \end{array} : \begin{array}{c} F(\boldsymbol{x}, \{\boldsymbol{m}_i\}_{i\in[\mu]}, \boldsymbol{r}, \boldsymbol{\alpha}) = 0 \wedge \\ \alpha_0 = \alpha \wedge \alpha_{i+1} = \alpha_i^2 \ \ \forall i \in [\log \nu - 1] \end{array} \right\}.$$

where $\mathbb{i}$ includes the description of finite field $\mathbb{F}$ and the function $F(\cdot)$ and maximum degree $d$. Next, we consider the situation for multiple predicate instances, i.e., given a tuple $(\mathbb{i}, \{\mathbb{x}^{(k)}, \mathbb{y}^{(k)}, \mathbb{w}^{(k)}\}_{k\in[\ell]})$ satisfying the multi-instance relation $\mathcal{R}^\ell$. To design a multi-cast reduction from $\mathcal{R}^\ell$ to a single relation $\mathcal{R}_{\mathsf{acc}}$, the prover and verifier can straightforwardly run the following process:

[6] The formal proof can be referred to [14].

- execute $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ for each tuple $(\mathbb{i}, \mathbb{x}^{(k)}, \mathbb{y}^{(k)}, \mathbb{w}^{(k)}), \forall k \in [\ell]$,
- apply the multi-cast reduction $\mathsf{IOR}_{\mathsf{cast}}$ from $\mathcal{R}_{\mathsf{sps}}^{\ell}$ to $\mathcal{R}_{\mathsf{acc}}$.

Although the above process fulfills our requirement, its practical efficiency is not satisfactory because the random oracles (later instantiated as hash functions) take inputs as $O(\ell n)$-sized messages $\{\boldsymbol{m}_i^{(k)}\}_{k\in[\ell]}$ when applying Fiat-Shamir transform in step 1.c and 2 of Figure 5.

To solve this problem, we slightly modify this process by interleaving the SPS protocol and the multi-cast reduction. The general idea is first to execute the original SPS protocol for each $i$, and send the oracle of the union polynomial (i.e., $\tilde{w}_{\cup}$ in steps 1,2 of $\mathsf{IOR}_{\mathsf{cast}}$) computed with all round-$i$ messages from different instances. We present the detailed protocol of $\mathsf{IOR}_{\mathsf{cast}}$ in Figure 6. Specifically, the indexer outputs $(\iota, \mathbb{I}, \mathbb{i}') \leftarrow \mathcal{I}(\mathbb{i})$, where $\mathbb{I} = \bot, \mathbb{i}' = \mathbb{i}$. The prover takes inputs $(\mathbb{i}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_{k\in[\ell]})$, and the verifier takes inputs $\iota, \{\mathbb{x}^{(k)}\}_{k\in[\ell]})$. For each round $i \in [\mu]$, the prover runs $\boldsymbol{m}_i^{(k)} \leftarrow \mathsf{P}_{\mathsf{sps}}(\boldsymbol{x}^{(k)}, \boldsymbol{w}^{(k)}, \{\boldsymbol{m}_j^{(k)}, r_j\}_{j\in[i-1]})$ at step 1.a. Then he collects the messages together as $\{\boldsymbol{m}_i^{(k)}\}_{k\in[\ell]}$ and computes $\tilde{m}_{\cup,i}(\boldsymbol{Y}, \boldsymbol{X})$ at step 1.b and sends its oracle at step 1.c. The verifier replies with a challenge $r_i$ at step 1.d. When applying the Fiat-Shamir transform here, the random oracles only need to take inputs as $O(n)$-sized messages $\boldsymbol{m}_{\cup,i}\ \forall i \in [\mu]$ instead of $O(\ell n)$. After the $\mu$ rounds interaction, the prover and verifier negotiate a random vector $\boldsymbol{\alpha}$ with the $\mu$-round challenge $r_\mu$ at step 2-5. Given that vectors $\boldsymbol{r}, \boldsymbol{\alpha}$ are properly generated, we have the following equation for each $k \in [\ell]$:

$$F(\boldsymbol{x}^{(k)}, \{\boldsymbol{m}_i^{(k)}\}_{i\in[\mu]}, \boldsymbol{r}_F) = 0 \tag{15}$$

where $\boldsymbol{r}_F := \boldsymbol{r}||\boldsymbol{\alpha}$ for simplicity.

As a result, the prover and verifier run the remaining process of the multi-cast reduction for the $\ell$ number of oracles $\{[\![\tilde{m}_{\cup,i}]\!]\}_{i\in[\ell]}$ with the function $F$. The reduced instance should satisfy the language $\mathcal{L}(\mathcal{R}_{\mathsf{acc}})$, where $\mathcal{R}_{\mathsf{acc}} = \mathcal{R}_{\mathsf{eval}}^{\mu} \times \mathcal{R}_{\mathsf{eval}}^{\mu} \times \mathcal{R}_F$ such that:

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{\tau}, \boldsymbol{r}_x, v_i), \mathbb{y} = [\![\tilde{m}_{\cup,i}]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}}\ \forall i \in [\mu] \tag{16}$$

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{r}_x, v_i), \mathbb{y} = [\![\tilde{m}_i]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}}\ \forall i \in [\mu] \tag{17}$$

$$(\mathbb{i}, \mathbb{x} = (\boldsymbol{x}, \boldsymbol{r}_F, e), \mathbb{y} = \bot, \mathbb{w} = \{\boldsymbol{m}_i\}_{i\in[\mu]}) \in \mathcal{R}_F \tag{18}$$

where $\boldsymbol{x} = \tilde{\boldsymbol{x}}(\boldsymbol{\tau}), \boldsymbol{m}_i = \tilde{\boldsymbol{m}}_i(\boldsymbol{\tau})\ \forall i \in [\mu]$.

According to Theorem 4, the protocol $\mathsf{IOR}_{\mathsf{cast}}$ can be further transformed into a non-interactive oracle proof $\mathsf{NIR}_{\mathsf{cast}}$, with a detailed description given in Appendix B.1. Lemma 5 highlights both the security of $\mathsf{IOR}_{\mathsf{cast}}$ in Figure 6 and $\mathsf{NIR}_{\mathsf{cast}}$, of which the security proof is omitted since it can be trivially derived from the security proofs of Lemma 3 and 4.

**Lemma 5.** *Let* $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ *be* $(k_0, \ldots, k_{\mu-1}, m+1)$*-special-sound for relation* $\mathcal{R}$ *for the relation* $\mathcal{R}$ *in terms of* $(\mathsf{P}_{\mathsf{sps}}, F)$*. Let* $\mathsf{IOR}_{\mathsf{cast}}$ *be an interactive oracle reduction from the multi-instance relation* $\mathcal{R}_F^{\ell}$ *to an indexed oracle relation* $\mathcal{R}_{\mathsf{acc}}$*. The above protocol* $\mathsf{IOR}_{\mathsf{cast}}$ *is an interactive oracle reduction from the multi-instance relation* $\mathcal{R}^{\ell}$ *to an oracle relation* $\mathcal{R}_{\mathsf{acc}}$*. It satisfies completeness and*

$\mathcal{P}$'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 $i \in [\mu]$ :

$a$ : $\forall k \in [\ell], \boldsymbol{m}_i^{(k)} \leftarrow \mathsf{P}_{\mathsf{sps}}(\boldsymbol{x}^{(k)}, \boldsymbol{w}^{(k)}, \{\boldsymbol{m}_j^{(k)}, r_j\}_{j=0}^{i-1})$

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

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

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

2 : Set $\alpha := r_\mu$

3 : $\mathcal{P}$ computes $\boldsymbol{\alpha} := (\alpha, \alpha^2, \alpha^4, \ldots, \alpha^{2^{\log m - 1}})$

4 : $\mathcal{P} \to \mathcal{V} : \boldsymbol{\alpha}$

5 : $\mathcal{V}$ checks that

$a$ : $\alpha_0 = \alpha$

$b$ : $\alpha_{i+1} = \alpha_i^2 \ \ \forall i \in [\log \nu - 1]$

6 : $\mathcal{P}$ and $\mathcal{V}$ run the remaining process of $\mathsf{IOR}_{\mathsf{cast}}$ (step 3-9).

**Output phase.**

7 : Define $\boldsymbol{x} = \sum_{k\in[\ell]} \tilde{eq}_{k-1}(\boldsymbol{\tau}) \cdot \boldsymbol{x}^{(k)}$

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

9 : $\mathcal{V}$ outputs $\mathbb{x} = (\boldsymbol{x}, \boldsymbol{r}_F, \boldsymbol{\tau}, \boldsymbol{r}_x, e), \mathbb{y} = (\{[\![\tilde{m}_{\cup,i}]\!]\}_{i\in[\mu]}, \{[\![\tilde{m}_i]\!]\}_{i\in[\mu]})$

**Fig. 6.** Interactive Oracle Reduction $\mathsf{IOR}_{\mathsf{cast}}$ from $\mathcal{R}^\ell$ to $\mathcal{R}_{\mathsf{acc}}$

*RBR knowledge soundness. By applying the compilation with an extractable PCS and the Fiat-Shamir transform, we obtain a non-interactive reduction* $\mathsf{NIR}_{\mathsf{cast}} = \mathsf{FS}[\mathsf{IOR}_{\mathsf{cast}}^{\mathsf{PC}}]$ *under the random oracle model, with complexity measured as follows:*

- *The proof length includes* $2\mu$ *elements for the commitments, and* $O(\log \ell)$ *field elements for the sum-check protocol.*
- *The random oracle query complexity includes* $\mu + \log \ell$ $O(1)$*-sized query,* 1 $O(\log m)$ *query,* 1 $O(\log m)$ *query,* 1 $O(\mu + \log \ell)$ *query and* 1 $O(\ell n)$ *query.*
- *The verifier time is dominated by the* $O(\ell n)$ *field computation for* $\mathbb{x}$*.*
- *The reduced relation* $\mathcal{R}_{\mathsf{acc}}$ *only contains* $2\mu$ ***polynomial oracles****.*

### 5.2 2-to-1 Reduction

We introduce the second component for building an accumulation scheme, i.e., the 2-to-1 reduction, which aims to reduce $\mathcal{R}_{\mathsf{acc}}^2$ to $\mathcal{R}_{\mathsf{acc}}$. A key technique necessary in this construction is to batch multiple oracles into one, which corresponds

to a polynomial aggregation scheme as described in Halo Infinite [10]. In this part, we only present a definition for this oracle batching protocol while leaving the concrete instantiations in Section 5.3.

**Definition 5 (adopted from [15]).** *An **oracle batching protocol** $\mathsf{IOR}_{\mathsf{batch}}$ with proximity radius $\delta^*$ is an interactive oracle reduction from indexed oracle relation $\mathcal{R}_0$ to indexed oracle relation $\mathcal{R}_1$:*

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

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

*Concretely, the verifier gets as input randomness $r \in \mathbb{F}$, a multilinear evaluation claim on $\sum_{i\in[1]} \tilde{eq}_i(r) \cdot \tilde{f}_i(\boldsymbol{x}) = v$ and oracle access to $\tilde{f}_0, \tilde{f}_1 \in \mathcal{F}^{(<2)}_{\log n}$. The prover additionally gets the witness $\boldsymbol{f}_0, \boldsymbol{f}_1$. After the interaction, the verifier outputs a list of multilinear evaluation claims on $\tilde{f}(\boldsymbol{x}_j) = v_j$ and an oracle $[\![\tilde{f}]\!]$. We highlight the protocol's property:*

- *Completeness: if $\sum_{i\in[1]} \tilde{eq}_i(r) \cdot \tilde{f}_i(\boldsymbol{x}) = v$, then $\tilde{f}(\boldsymbol{x}_j) = v_j$*
- *Soundness: for every $\delta \in (0, \delta^*)$, if $\Delta_{\mathcal{R}_0}(\mathbb{x}, \mathbb{y}) > \delta$ then, except for a small error probability $\epsilon_0$, $\Delta_{\mathcal{R}_1}(\mathbb{x}, \mathbb{y}) > \delta$.*
- *Succinctness: the proof $\pi_{\mathsf{batch}}$ of $\mathsf{IOR}_{\mathsf{batch}}$ (excluding the oracles) should be sub-linear of the polynomial size $|\tilde{f}|$.*

We now describe the 2-to-1 Reduction $\mathsf{IOR}_{\mathsf{fold}}$ from $(\mathcal{R}_{\mathsf{acc}})^2$ to $\mathcal{R}_{\mathsf{acc}}$ as follows. Lemma 6 highlights the security and theoretical performance of $\mathsf{IOR}_{\mathsf{fold}}$, of which the proof is given in the following.

**Construction 1** (2-to-1 Reduction $\mathsf{IOR}_{\mathsf{fold}}$)**.** Given inputs as two instance-wintness tuples $(\mathbb{i}_{\mathsf{acc}}, \mathbb{x}^{(k)}_{\mathsf{acc}}, \mathbb{y}^{(k)}_{\mathsf{acc}}, \mathbb{w}^{(k)}_{\mathsf{acc}}) \in \mathcal{R}^{\mu}_{\mathsf{eval}} \times \mathcal{R}^{\mu}_{\mathsf{eval}} \times \mathcal{R}_F$ into one, where

$$\mathbb{x}^{(k)}_{\mathsf{acc}} = (\boldsymbol{x}^{(k)}, \boldsymbol{\tau}^{(k)}, \boldsymbol{r}^{(k)}_x, \boldsymbol{r}^{(k)}_F, e^{(k)}, \{v^{(k)}_i\}_{i\in[\mu]}), \tag{19}$$

$$\mathbb{y}^{(k)}_{\mathsf{acc}} = (\{[\![\tilde{m}^{(k)}_{\cup,i}]\!], [\![\tilde{m}^{(k)}_i]\!]\}_{i\in[\mu]}, \tag{20}$$

$$\mathbb{w}^{(k)}_{\mathsf{acc}} = \{\boldsymbol{m}^{(k)}_i\}_{i\in[\mu]}. \tag{21}$$

- The prover first computes the following univariate polynomials:

$$\tilde{\boldsymbol{x}}(Z) = \sum_{k\in[1]} \tilde{eq}_k(Z) \cdot \boldsymbol{x}^{(k)} \in (\mathcal{F}^{(<2)}_1)^m, \tag{22}$$

$$\tilde{\boldsymbol{r}}_F(Z) = \sum_{k\in[1]} \tilde{eq}_k(Z) \cdot \boldsymbol{r}^{(k)}_F \in (\mathcal{F}^{(<2)}_1)^{\mu+\log\nu}, \tag{23}$$

$$\tilde{\boldsymbol{\tau}}(Z) = \sum_{k\in[1]} \tilde{eq}_k(Z) \cdot \boldsymbol{\tau}^{(k)} \in (\mathcal{F}^{(<2)}_1)^{\log\ell}, \tag{24}$$

$$\tilde{\boldsymbol{r}}_x(Z) = \sum_{k\in[1]} \tilde{eq}_k(Z)\cdot \boldsymbol{r}_x^{(k)} \in (\mathcal{F}_1^{(<2)})^{\log n}, \tag{25}$$

$$\tilde{\boldsymbol{m}}_{\cup,i}(Z) = \sum_{k\in[1]} \tilde{eq}_k(Z)\cdot \boldsymbol{m}_{\cup}^{(k)} \in (\mathcal{F}_1^{(<2)})^{\ell n}\ \forall i\in[\mu], \tag{26}$$

$$\tilde{\boldsymbol{m}}_{i}(Z) = \sum_{k\in[1]} \tilde{eq}_k(Z)\cdot \boldsymbol{m}_{i}^{(k)} \in (\mathcal{F}_1^{(<2)})^{n}\ \forall i\in[\mu]. \tag{27}$$

Note that the above polynomials $\tilde{\boldsymbol{x}}(Z)$, $\boldsymbol{r}_F(Z)$, $\boldsymbol{\tau}$, $\boldsymbol{r}_x(Z)$, $\{\boldsymbol{m}_{\cup,i}(Z)\}_{i\in[\mu]}$, $\{\boldsymbol{m}_i(Z)\}_{i\in[\mu]}$ should satisfy the following equations for all $z\in\{0,1\}$ according to Lemma 1:

$$F(\tilde{\boldsymbol{x}}(z), \{\boldsymbol{m}_i(z)\}_{i\in[\mu]}, \boldsymbol{r}_F(z)) = \sum_{k\in[1]} \tilde{eq}_k(z)\cdot e^{(k)} = \tilde{e}(z), \tag{28}$$

$$\tilde{m}_i(z, \boldsymbol{r}_x(z)) = \sum_{k\in[1]} \tilde{eq}_k(z)\cdot v_i^{(k)} = \tilde{v}_i(z)\ \forall i\in[\mu], \tag{29}$$

$$\tilde{m}_{\cup,i}(z, \boldsymbol{\tau}(z), \boldsymbol{r}_x(z)) = \sum_{k\in[1]} \tilde{eq}_k(z)\cdot v_i^{(k)} = \tilde{v}_i(z)\ \forall i\in[\mu]. \tag{30}$$

- Given challenges $\boldsymbol{\gamma}\leftarrow\$\ \mathbb{F}^{\log(\mu+1)}, r_z\leftarrow\$\ \mathbb{F}$, the prover first combines above $2\cdot\mu+1$ equations in Equation 28-30 into one by $\boldsymbol{\gamma}$. Then, similar to Section 4.2, the prover computes the following polynomial with degree $\max(d+1, \log\ell+\log n)$ with $r_z$:

$$\begin{aligned} G(Z) = \tilde{eq}(r_z, Z)\cdot\big[&(F(\tilde{\boldsymbol{x}}(Z), \{\boldsymbol{m}_i(Z)\}_{i\in[\mu]}, \boldsymbol{r}_F(Z)) - \tilde{e}(Z)) \\ &+ \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{\gamma})\cdot(\tilde{m}_i(Z, \boldsymbol{r}_x(Z))) - \tilde{v}_i(Z)) \\ &+ \sum_{i\in[\mu]} \mathsf{pow}_{\mu+i}(\boldsymbol{\gamma})\cdot(\tilde{m}_{\cup,i}(Z, \boldsymbol{\tau}(Z), \boldsymbol{r}_x(Z)) - \tilde{v}_i(Z))\big] \end{aligned} \tag{31}$$

- The prover and verifier engage in a 1-round sum-check protocol for the statement $\sum_{z\in\{0,1\}} G(z) = 0$:
  - the prover directly sends the polynomial $G(Z)$
  - the verifier checks that $G(0)+G(1)=0$
  - the verifier outputs an evaluation claim $G(\sigma)=v_G$ in terms of a virtual oracle $[\![G]\!]$.
- Similarly, the verifier can define the virtual oracle $[\![G]\!]$ based on all batched oracles $\{[\![\tilde{m}_{\cup,i}]\!]\}_{i\in[\mu]}$, $\{[\![\tilde{m}_i]\!]\}_{i\in[\mu]} \in \mathbb{y}_{\mathsf{sps}}$ that are multilinear extended from $\{\tilde{\boldsymbol{m}}_{\cup,i}(r_z)\}_{i\in[\mu]}$, $\{\tilde{\boldsymbol{m}}_i(r_z)\}_{i\in[\mu]}$. Hence the prover sends

$$\eta = F(\tilde{\boldsymbol{x}}(\sigma), \{\boldsymbol{m}_i(\sigma)\}, \boldsymbol{r}_F(\sigma) - e(\sigma)) \tag{32}$$

$$\eta_i = \tilde{m}_i(\sigma, \boldsymbol{r}_x(\sigma)) - v_i(\sigma) \tag{33}$$

$$\eta_{\cup,i} = \tilde{m}_{\cup,i}(\sigma, \boldsymbol{\tau}(\sigma), \boldsymbol{r}_x(\sigma)) - v_i(\sigma) \tag{34}$$

- The verifier checks if

$$G(\sigma) = \tilde{eq}(r_z, \sigma)\cdot(\eta + \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{\gamma})\cdot\eta_i + \sum_{i\in[\mu]} \mathsf{pow}_{\mu+i}(\boldsymbol{\gamma})\cdot\eta_{\cup,i}). \tag{35}$$

- Finally, the prover and verifier engage in $2\mu$ oracle batching protocols in parallel $\mathsf{IOR}_{\mathsf{batch}}$ with the random $\sigma \in \mathbb{F}$. This is to batch the oracles in $\mathbb{y}_{\mathsf{sps}}$ above from those in $\mathbb{y}_{\mathsf{sps}}^{(k)}$ for all $k \in [1]$. The derived tuple is $(\mathbb{i}_{\mathsf{acc}}, \mathbb{x}_{\mathsf{acc}}, \mathbb{y}_{\mathsf{acc}}, \mathbb{w}_{\mathsf{acc}})$ satisfying $\mathcal{R}_{\mathsf{acc}} = \mathcal{R}_{\mathsf{eval}}^{\mu} \times \mathcal{R}_{\mathsf{eval}}^{\mu} \times \mathcal{R}_F$

$$(\mathbb{i}_{\mathsf{acc}}, \mathbb{x} = \{(\boldsymbol{y}_j, \boldsymbol{x}_j, v_j)\}_j, \mathbb{y} = [\![\tilde{m}_{\cup,i}]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}} \ \forall i \in [\mu] \tag{36}$$

$$(\mathbb{i}_{\mathsf{acc}}, \mathbb{x} = \{(\boldsymbol{x}_t, v_t)\}_t, \mathbb{y} = [\![\tilde{m}_i]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}} \ \forall i \in [\mu] \tag{37}$$

$$(\mathbb{i}_{\mathsf{acc}}, \mathbb{x} = (\boldsymbol{x}, \boldsymbol{r}_F, e), \mathbb{y} = \bot, \mathbb{w} = \{\boldsymbol{m}_i\}_{i\in[\mu]}) \in \mathcal{R}_F \tag{38}$$

  where $\tilde{m}_{\cup,i} = \tilde{\boldsymbol{m}}_{\cup,i}(\sigma), \tilde{m}_i = \tilde{\boldsymbol{m}}_i(\sigma)$ for all $i \in [\mu]$, and $\{(\boldsymbol{y}_j, \boldsymbol{x}_j, v_j)\}_j$ and $\{(\boldsymbol{x}_t, v_t)\}_t$ are newly generated evaluation claims in the oracle batching protocols $\mathsf{IOR}_{\mathsf{batch}}$.

*Remark 4.* Construction 1 is a 2-to-1 reduction from $(\mathcal{R}_{\mathsf{acc}})^2$ to $\mathcal{R}_{\mathsf{acc}}$, where $\mathcal{R}_{\mathsf{acc}} = \mathcal{R}_{\mathsf{eval}}^{\mu} \times \mathcal{R}_{\mathsf{eval}}^{\mu} \times \mathcal{R}_F$. While the newly generated relation $\mathcal{R}_{\mathsf{eval}}$ contains a list of evaluation claims instead of one. As mentioned in [20,15], the mismatch can be resolved by adding extra evaluation claims in $\mathcal{R}_{\mathsf{acc}}$ at an arbitrary point, e.g., 0, which does not have to be chosen at random.

*Optimization.* In the above IOR protocol, the prover and verifier batch two oracle strings $\mathbb{y}^{(k)}, k \in [1]$, where each string consists of $2\mu$ oracles. In fact, the number can be reduced to $\mu + 1$ by running a sub-protocol for preprocessing. For an instance-wintness tuple $(\mathbb{x}, \mathbb{y}, \mathbb{w}) \in \mathcal{R}_{\mathsf{eval}}^{\mu}$ with $(\mathbb{x} = (\boldsymbol{\tau}, \boldsymbol{r}_x, v_i), \mathbb{y} = [\![\tilde{m}_{\cup,i}]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}} \ \forall i \in [\mu]$, the prover and verifier run an oracle batching protocol to obtain a new instance-witness tuple

$$(\mathbb{i}_{\mathsf{acc}}, \mathbb{x} = ((\boldsymbol{\tau}, \boldsymbol{r}_x, v), \{(\boldsymbol{y}_j, \boldsymbol{x}_j, v_j)\}_j), \mathbb{y} = [\![\tilde{m}_{\cup}]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}}, \tag{39}$$

where $v = \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{\beta}) \cdot v_i$ is checked by the verifier, and $\tilde{m}_{\cup}$ is computed as

$$\tilde{m}_{\cup}(\boldsymbol{Y}, \boldsymbol{X}) = \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{\beta}) \cdot \tilde{m}_{\cup,i}(\boldsymbol{Y}, \boldsymbol{X}), \tag{40}$$

Finally, the reduced new relation is $\mathcal{R}_{\mathsf{eval}} \times \mathcal{R}_{\mathsf{eval}}^{\mu} \times \mathcal{R}_F$.

According to Theorem 4, the protocol $\mathsf{IOR}_{\mathsf{fold}}$ can be further transformed into a non-interactive oracle proof $\mathsf{NIR}_{\mathsf{fold}}$, with a detailed description given in Appendix B.2. The proof of Lemma 6 is given in Appendix C.2.

**Lemma 6.** *Let $\mathsf{IOR}_{\mathsf{batch}}$ be an interactive oracle reduction from an indexed oracle relation $\mathcal{R}_1$ to an indexed oracle relation $\mathcal{R}_2$. The above protocol $\mathsf{IOR}_{\mathsf{fold}}$ is an interactive oracle reduction from the multi-instance relation $(\mathcal{R}'_{\mathsf{acc}})^2$ to an indexed oracle relation $\mathcal{R}'_{\mathsf{acc}}$. It satisfies completeness and RBR knowledge soundness. By applying the compilation with an extractable PCS and the Fiat-Shamir transform, we obtain a non-interactive reduction $\mathsf{NIR}_{\mathsf{fold}} = \mathsf{FS}[\mathsf{IOR}_{\mathsf{fold}}^{\mathsf{PC}}]$ under the random oracle model, with complexity measured as follows:*

- *The protocol has 3 rounds.*

- *The proof length includes $O(d+\mu)$ field elements for the sum-check protocol and the proof elements for $\mathsf{NIR}_{\mathsf{batch}}$. The oracle proof length is $O(\mu\ell n)$ field elements (optimized to $O(\mu n)$) for $[\![\tilde{w}_{\cup,i}]\!]$'s and $[\![\tilde{w}_i]\!]$'s.*
- *The verifier makes no queries.*
- *The verifier time is dominated by the $O(\ell n)$ field computation for computing $\mathbb{x}$ and the verification times of $\mathsf{NIR}_{\mathsf{batch}}$.*
- *The reduced relation $\mathcal{R}_{\mathsf{acc}}$ only contains $2\cdot\mu$ **polynomial oracles** (optimized to $\mu+1$).*

### 5.3 Putting them together

In this part, we discuss how to construct an accumulation scheme based on different PCS candidate and analyze their theoretical performance. We first introduce a conclusion in Theorem 3 similar to [20] for the general construction of the multi-instance accumulation scheme from the two components presented previously. We defer the proof of Theorem 3 to the Appendix B.3. For simplicity, we only consider distance-preserving accumulation scheme in this paper. Nevertheless, we claim it is straightforward to extend the above theorem to promise relations by following the proof in [20].

**Theorem 3.** *There exists a polynomial-time transformation $T$ such that the following holds. Let $\mathcal{R}$ be an indexed relation parameterized by public parameters $\mathsf{pp}$. Let $\mathcal{R}_{\mathsf{acc}}$ be an indexed oracle relation parameterized by the same $\mathsf{pp}$ and be implicitly relative to a random oracle $\mathsf{RO}$. Given the following non-interactive reductions in the random oracle model:*

- *$\mathsf{NIR}_{\mathsf{cast}} = (\mathcal{G}_{\mathsf{cast}}, \mathcal{P}_{\mathsf{cast}}, \mathcal{V}_{\mathsf{cast}})$, a multi-cast reduction from $(\mathcal{R}_F^{\mathsf{PC}})^\ell$ to $\mathcal{R}_{\mathsf{acc}}^{\mathsf{PC}}$.*
- *$\mathsf{NIR}_{\mathsf{fold}} = (\mathcal{G}_{\mathsf{fold}}, \mathcal{P}_{\mathsf{fold}}, \mathcal{V}_{\mathsf{fold}})$, a 2-to-1 reduction from $(\mathcal{R}_{\mathsf{acc}}^{\mathsf{PC}})^2$ to $\mathcal{R}_{\mathsf{acc}}^{\mathsf{PC}}$ with the same generator algorithm as $\mathsf{NIR}_{\mathsf{cast}}$,*

*there exists $T[\mathsf{NIR}_{\mathsf{cast}}, \mathsf{NIR}_{\mathsf{fold}}, \mathcal{R}_{\mathsf{acc}}] = (\mathsf{NARK}, \mathsf{ACC})$, where $\mathsf{NARK}$ is a non-interactive argument for $\mathcal{R}$ and $\mathsf{ACC}$ is an accumulation scheme for $\mathsf{NARK}$, both in the random oracle model. Denote by $|m|$, $|x|$, $|r|$ the number of elements in the prover messages $[\boldsymbol{m}_i]_{i\in[\mu]}$, instances $[\mathbb{x}^{(k)}]_{k\in[\ell]}$, and random challenges $\boldsymbol{r}, \boldsymbol{\alpha}$. Let $n = \max_{i\in[\mu]} |\boldsymbol{m}_i|$ be the maximum length among all messages. We claim that the accumulation scheme can achieve the following theoretical performance.*

- *The accumulation prover cost is dominated by $2\mu$ $\mathbb{G}$ operations, $O(\ell\cdot m+\mu\cdot n)$ $\mathbb{F}$ operations, and $\mu+\log\ell$ RO queries, and the cost of $2\mu$ $\mathsf{NIR}_{\mathsf{batch}}.\mathcal{P}$ algorithm.*
- *The accumulation verifier cost is dominated by $2\mu$ $\mathbb{G}$ operations, $O(\ell\cdot m)$ $\mathbb{F}$ operations for accumulating $\{\mathbb{x}^{(k)}\}_{k\in[\ell]}$, and $\mu+\log\ell$ RO queries, and the cost of $2\mu$ $\mathsf{NIR}_{\mathsf{batch}}.\mathcal{V}$ algorithm.*
- *The decider cost is dominated by checking the $O(\mu)$ evaluation claims with respect to the oracles, which is dependent on the PCS evaluation algorithm complexity.*

Next, we review a list of practical PCS candidates in real-world implementations and analyze their performances. Note that throughout the accumulation

**Table 2.** Comparisons between Multi-Accumulation Schemes: denote $n$ as the polynomial size, the last column compares the proof size of the oracle batching protocols instantiated on the corresponding PCS

| Schemes | Assumptions | Prover | Proof size | Batch proof size |
|---|---|---|---|---|
| Bulletproofs [13] | DL | $O(n \log n)$ | $O(\log n)$ | $O(\log n)$ |
| Mercury [26] | DL | $O(n \log n)$ | $O(1)$ | $O(1)$ |
| RS codes [20] | Hash | $O(n \log n)$ | $O(\log n)$ | $O(\frac{\lambda}{\log(1/\rho)} \cdot \log n)$ |
| Linear codes [15] | Hash | $O(n)$ | $O(\log n)$ | $O(\frac{\lambda}{\log(1/\rho)} \cdot \log n)$ |

scheme, only the oracle batching protocols contains the oracle queries. Hence, we mainly focus on this part. To make our improvement approach distinguishable, the key point is that the batching cost should be as small as possible. Or, theoretically speaking, the cost should be independent of the oracle size, i.e., the succinctness property as defined in Definition 5. Thankfully, most PCS satisfy our requirements as shown in Table 2. Typically, for polynomial commitments providing homomorphic property, the oracle batching protocol is trivial to instantiate because, the verifier can simply checks if $C = \sum_{i\in[1]} \tilde{eq}_i(r) \cdot C_i$ where $C := \mathsf{Commit}(\mathsf{ck}, \sum_{i\in[1]} \tilde{eq}_i(r) \cdot \tilde{f}_i(\boldsymbol{x}))$. For non-homomorphic polynomial commitments such as the code-based ones [5,31,43,44], a recent line of work [19,20,15] conducted a comprehensive research on this problem. We present a more detailed description on the code-based instantiations in Section 6.

## 6 Quasar

As stated in Theorem 1, there exist efficient constructions for a multi-instance IVC based on a NARK system and a multi-instance accumulation scheme. We sketch the general process:

- Given a multi-predicate input $(\mathbb{x}^{(k)}_{k\in[\ell]}, \pi)$, where $\pi$ is a NARK proof, and an accumulator $\mathsf{acc}$, the IVC prover first runs the proving algorithm $\mathsf{ACC}.\mathcal{P}$ to obtain a new accumulator $\mathsf{acc}'$ as well as a proof $\mathsf{pf}$.
- Next, the IVC prover constructs $\ell$ recursive circuits each consists of the following parts:
  - the trace for computing the predicate $\varphi$ with $k$-th instance-witness pair.
  - if $k = 0$: the trace for verifying the accumulation between a multi-predicate instance $(\{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi.\mathbb{x})$ and an accumulator $\mathsf{acc}$ at the previous step; else a dummy trace.
  - if $k = 0$: the trace for compressing the accumulator $\mathsf{acc}$ by cryptographic hash functions; else a dummy trace.
- For each in the $\ell$ recursive circuits, the IVC prover first arithmetizes it into a tuple $(\mathbb{i}', \mathbb{x}', \mathbb{w}') \in \mathcal{R}$ under an existing *constraint system*. Then he calls the proving algorithm of multi-cast reduction $\mathsf{NIR}_{\mathsf{multicast}}$ complied with a *polynomial commitment scheme*, which outputs a reduced witness $\mathbb{w}'$ and a NARK proof $\pi'$.

- The IVC prover outputs the multi-predicate tuple $((\{(\mathbb{x}')^{(k)}\}_{k\in[\ell]}, \pi'.\mathbb{x}), \mathbb{w}')$, and a new accumulator $\mathsf{acc}'$.
- The verifier takes as inputs a multi-predicate tuple $((\{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi.\mathbb{x}), \mathbb{w})$ and an accumulator $\mathsf{acc}$, he first runs the verification algorithm of $\mathsf{NIR}_{\mathsf{multicast}}$ to derive $\mathbb{x}$. Then he can check the validity of $\mathbb{x}$, $\mathbb{w}$ and $\mathsf{acc}$ by running the decider algorithm.

In this work, we additionally require Quasar to provide some cutting-edge properties, including the linear-time prover and plausible post-quantum security. Typically, we claim that such an IVC can be constructed from a multilinear plonkish constraint system (hyperplonk) and a linear-code-based PCS. We describe the detailed instantiations in the Section 6.1 and 6.2.

### 6.1 SPS for HyperPlonk Relations

We present special-sound protocols for multilinear plonkish relation (Hyperplonk) as defined below.

**Definition 6 (Multilinear plonkish relation).** *Fix public parameters including the field $\mathbb{F}$, the instance length $m$, the total number of gates $\mu$, the number of possible gate types $s$, i.e., the number of selectors, the number of fan-in/fan-outs of each gate $n$, and an algebraic map $f : \mathbb{F}^{s+n} \to \mathbb{F}$ with degree $d$. The indexed oracle relation $\mathcal{R}_{\mathsf{plonk}}$ is the set of all tuples*

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

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

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

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

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

We present the special sound protocols in Appendix B.4.

### 6.2 Linear Time Accumulation Scheme

Section 5.3 provided a general idea for instantiating the multi-instance accumulation schemes with homomorphic PCS. In this part, we present a concrete instantiations on the code-based PCS to achieve more desirable features. We first recap several notions, which are formally defined in Appendix A.1.

- A linear code is a *linear* injective map $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ with message length $k$ and codeword length $n$.

- The rate of a code $\mathcal{C}$ is $\rho := k/n$.
- For a code $\mathcal{C}$, the list decoding of a codeword $\boldsymbol{u} \in \mathcal{C}$ within distance $\delta$ is denoted as $\Lambda(\mathcal{C}, \boldsymbol{u}, \delta)$.
- Out-of-domain sampling allows a verifier to sample the codeword $\boldsymbol{u}$ at a random point $\boldsymbol{\tau} \in \mathbb{F}^{\log n} \setminus \{0,1\}^{\log n}$ and obtain an evaluation $\tilde{u}(\boldsymbol{\tau})$ of the *multilinear extension* of $\boldsymbol{u}$.

To instantiate the accumulation scheme, the key is to construct an efficient oracle batching protocol $\mathsf{NIR}_{\mathsf{batch}}$ compiled with a commitment scheme. The authors in [15] introduces a practical codeword batching protocol for every linear codes, which can be further compile into Merkle commitments by BCS transform. Typically the protocol reduces two proximity claims: codewords $\boldsymbol{u}_1, \boldsymbol{u}_2$ are $\delta$-close to $\mathcal{C}$, to a new proximity claim: codeword $\boldsymbol{u}$ is $\delta$-close to $\mathcal{C}$, where an extra evaluation constraint holds as $\gamma_1 \cdot \tilde{u}_1(\boldsymbol{x}) + \gamma_2 \cdot \tilde{u}_2(\boldsymbol{x}) = \tilde{u}(\boldsymbol{x})$. Intuitively, we can simply encode the multilinear polynomials in relation $\mathcal{R}_0$ in Definition 5 into codewords, and instantiate the codeword batching protocol accordingly. However, there is a technical gap: the relation $\mathcal{R}_0$ includes of evaluation claims on multilinear polynomials $\tilde{f}_0, \tilde{f}_1$. while the codeword batching protocol requires the evaluation claims on $\boldsymbol{u}_i := \mathcal{C}(\boldsymbol{f}_i), i \in [1]$ instead of $\boldsymbol{f}_i$'s. Thankfully, this mismatch can be easily solved if we assume the coding algorithm is *systematic*, i.e., for all $\boldsymbol{x} \in \mathbb{F}^k$, the first $k$ entries of $\mathcal{C}(\boldsymbol{x})$ are equal to $\boldsymbol{x}$. Assume the reciprocal of the code rate $\rho$ is in $2^{\mathbb{N}}$, the multilinear extension of a codeword $\boldsymbol{u}_i$ can be written as

$$\tilde{u}_i(\boldsymbol{Y}, \boldsymbol{X}) := \tilde{eq}_0(\boldsymbol{Y}) \cdot \tilde{f}_i(\boldsymbol{X}) + \sum_{i=1}^{\log(1/\rho)} \tilde{eq}_i(\boldsymbol{Y}) \sum_{j=0}^{\log k - 1} \tilde{eq}_j(\boldsymbol{X}) \cdot \boldsymbol{u}_i[i \cdot k + j].$$

Simply, for an evaluation claim $\tilde{f}_i(\boldsymbol{x}) = v_i, \boldsymbol{x} \in \mathbb{F}^{\log n}$, the prover can pad it with zero into a vector $(\boldsymbol{0}, \boldsymbol{x}) \in \mathbb{F}^{\log(1/\rho)+\log n}$, where the claim also holds $\tilde{u}_i(\boldsymbol{0}, \boldsymbol{x}) = v_i$. Hence given a relation $\mathcal{R}_0$, the prover and verifier first encode it as the following relation

$$\mathcal{R}_0' := \left\{ \begin{array}{c} \mathbb{i}, \mathbb{x} = (\boldsymbol{\alpha} := (\boldsymbol{0}, \boldsymbol{x}), v, r), \\ \mathbb{y} = (\llbracket \tilde{u}_0 \rrbracket, \llbracket \tilde{u}_0 \rrbracket), \mathbb{w} = (\boldsymbol{u}_0, \boldsymbol{u}_1) \end{array} : \sum_{k \in [1]} \tilde{eq}_k(r) \cdot \tilde{u}_k(\boldsymbol{x}) = v \right\}$$

where $\boldsymbol{u}_0 = \mathcal{C}(\boldsymbol{f}_0), \boldsymbol{u}_1 = \mathcal{C}(\boldsymbol{f}_1)$. We present the oracle batching protocol based on codewords in Figure 14 in Appendix B.4.

## Acknowledgments

We utilized generative AI tools (such as ChatGPT) exclusively for minor phrasing adjustments, grammar improvements, and spelling checks. Their use was limited to supporting the editorial refinement of the manuscript.

## References


1. Arnon, G., Chiesa, A., Fenzi, G., Yogev, E.: Stir: reed-solomon proximity testing with fewer queries. In: Annual International Cryptology Conference. pp. 380–413. Springer (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: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 3–33. Springer (2024), https://doi.org/10.1007/978-3-031-58751-1_1
3. Attema, T., Cramer, R.: Compressed $\Sigma$-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics. In: Proc. of the Annual International Cryptology Conference (CRYPTO). pp. 513–543. Springer (2020), https://doi.org/10.1007/978-3-030-56877-1_18
4. Attema, T., Cramer, R., Kohl, L.: A compressed $\sigma$-protocol theory for lattices. In: Annual International Cryptology Conference. pp. 549–579. Springer (2021), https://doi.org/10.1007/978-3-030-84245-1_19
5. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: 45th international colloquium on automata, languages, and programming (icalp 2018). pp. 14–1. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2018)
6. Ben-Sasson, E., Chiesa, A., Goldberg, L., Gur, T., Riabzev, M., Spooner, N.: Linear-size constant-query iops for delegating computation. In: Theory of Cryptography Conference. pp. 494–521. Springer (2019), https://doi.org/10.1007/978-3-030-36033-7_19
7. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Theory of Cryptography Conference. pp. 31–60. Springer (2016), https://doi.org/10.1007/978-3-662-53644-5_2
8. 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/10.1007/978-3-662-44381-1_16
9. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for snarks and proof-carrying data. In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing. pp. 111–120 (2013), https://doi.org/10.1145/2488608.2488623
10. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo infinite: Proof-carrying data from additive polynomial commitments. In: Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41. pp. 649–680. Springer (2021), https://doi.org/10.1007/978-3-030-84242-0_23
11. Bootle, J., Chiesa, A., Hu, Y., Orru, M.: Gemini: Elastic snarks for diverse environments. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 427–457. Springer (2022), https://doi.org/10.1007/978-3-031-07085-3_15
12. Bowe, S., Grigg, J., Hopwood, D.: Recursive proof composition without a trusted setup. Cryptology ePrint Archive (2019)
13. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE symposium on security and privacy (SP). pp. 315–334. IEEE (2018)
14. Bünz, B., Chen, B.: Protostar: Generic efficient accumulation/folding for special-sound protocols. In: International Conference on the Theory and Application of

Cryptology and Information Security. pp. 77–110. Springer (2023), https://doi.org/10.1007/978-981-99-8724-5_3
15. Bünz, B., Chiesa, A., Fenzi, G., Wang, W.: Linear-time accumulation schemes. Cryptology ePrint Archive (2025)
16. Bünz, B., Chiesa, A., Lin, W., Mishra, P., Spooner, N.: Proof-carrying data without succinct arguments. In: Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41. pp. 681–710. Springer (2021), https://doi.org/10.1007/978-3-030-84242-0_24
17. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Proof-carrying data from accumulation schemes. Cryptology ePrint Archive (2020)
18. Bünz, B., Fisch, B., Szepieniec, A.: Transparent snarks from dark compilers. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 677–706. Springer (2020), https://doi.org/10.1007/978-3-030-45721-1_24
19. Bünz, B., Mishra, P., Nguyen, W., Wang, W.: Accumulation without homomorphism. Cryptology ePrint Archive (2024)
20. Bünz, B., Mishra, P., Nguyen, W., Wang, W.: Arc: Accumulation for reed–solomon codes. Cryptology ePrint Archive (2024)
21. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 499–530. Springer (2023), https://doi.org/10.1007/978-3-031-30617-4_17
22. Chiesa, A.: Proof-carrying data. Ph.D. thesis, Massachusetts Institute of Technology (2010)
23. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zksnarks with universal and updatable srs. In: Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39. pp. 738–768. Springer (2020), https://doi.org/10.1007/978-3-030-45721-1_26
24. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 769–793. Springer (2020), https://doi.org/10.1007/978-3-030-45721-1_27
25. Eagen, L., Gabizon, A.: Protogalaxy: Efficient protostar-style folding of multiple instances. Cryptology ePrint Archive (2023)
26. Eagen, L., Gabizon, A.: Mercury: A multilinear polynomial commitment scheme with constant proof size and no prover ffts. Cryptology ePrint Archive (2025), https://doi.org/10.1007/978-3-031-15985-5_11
27. Fiat, A., Shamir, A.: How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Proc. of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). pp. 186–194. Springer (1986), https://doi.org/10.1007/3-540-47721-7_12
28. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive (2019)
29. Ganesh, C., Patranabis, S., Singh, N.: Samaritan: Linear-time prover snark from new multilinear polynomial commitments. Cryptology ePrint Archive (2025)
30. Gao, S., Qian, C., Zheng, T., Guo, Y., Xiao, B.: Sigma-check: Compressed sigma-protocol theory from sum-check. Cryptology ePrint Archive (2024)

31. Golovnev, A., Lee, J., Setty, S., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic snarks for r1cs. In: Annual International Cryptology Conference. pp. 193–226. Springer (2023), https://doi.org/10.1007/978-3-031-38545-2_7
32. Kothapalli, A., Setty, S.: Cyclefold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. Cryptology ePrint Archive (2023)
33. Kothapalli, A., Setty, S.: Hypernova: Recursive arguments for customizable constraint systems. In: Annual International Cryptology Conference. pp. 345–379. Springer (2024), https://doi.org/10.1007/978-3-031-68403-6_11
34. Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive zero-knowledge arguments from folding schemes. In: Annual International Cryptology Conference. pp. 359–388. Springer (2022), https://doi.org/10.1007/978-3-031-15985-5_13
35. Liu, T., Xie, X., Zhang, Y.: Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 2968–2985 (2021), https://doi.org/10.1145/3460120.3485379
36. Liu, T., Zhang, Z., Zhang, Y., Hu, W., Zhang, Y.: Ceno: Non-uniform, segment and parallel zero-knowledge virtual machine. Journal of Cryptology **38**(2), 17 (2025), https://doi.org/10.1007/s00145-024-09533-2
37. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. Journal of the ACM (JACM) **39**(4), 859–868 (1992), https://doi.org/10.1109/fscs.1990.89518
38. Nguyen, W., Datta, T., Chen, B., Tyagi, N., Boneh, D.: Mangrove: A scalable framework for folding-based snarks. In: Annual International Cryptology Conference. pp. 308–344. Springer (2024), https://doi.org/10.1007/978-3-031-68403-6_10
39. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. Journal of the ACM (JACM) **27**(4), 701–717 (1980), https://doi.org/10.1145/322217.322225
40. Sun, H., Li, J., Zhang, H.: zkllm: Zero knowledge proofs for large language models. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. pp. 4405–4419 (2024)
41. 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
42. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Theory of cryptography conference. pp. 1–18. Springer (2008), https://doi.org/10.1007/978-3-540-78524-8_1
43. Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. In: Annual International Cryptology Conference. pp. 299–328. Springer (2022)
44. Zeilberger, H., Chen, B., Fisch, B.: Basefold: efficient field-agnostic polynomial commitment schemes from foldable codes. In: Annual International Cryptology Conference. pp. 138–169. Springer (2024), https://doi.org/10.1007/978-3-031-68403-6_5
45. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vsql: Verifying arbitrary sql queries over dynamic outsourced databases. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 863–880. IEEE (2017), https://doi.org/10.1109/SP.2017.43
46. Zheng, T., Gao, S., Guo, Y., Xiao, B.: Kilonova: Non-uniform pcd with zero-knowledge property from generic folding schemes. Cryptology ePrint Archive (2023)

47. Zhou, W., Cai, Y., Peng, Y., Wang, S., Ma, K., Li, F.: Veridb: An sgx-based verifiable database. In: Proceedings of the 2021 International Conference on Management of Data. pp. 2182–2194 (2021), https://doi.org/10.1145/3448016.3457308

# A Formal Definitions

## A.1 Linear Codes

**Definition 7 (Linear code).** *A* code *over an alphabet $\Sigma$ with message length $k$ and codeword length $n$ is an injective map $\mathcal{C} : \Sigma^k \to \Sigma^n$. A code $\mathcal{C}$ is linear code if $\Sigma = \mathbb{F}$ is a field and $\mathcal{C}$ is linear. In this paper we assume that $n \in 2^{\mathbb{N}}$. The* minimum distance *of a code $\mathcal{C}$ is defied as $\delta(\mathcal{C}) := \mathsf{min}_{u \neq v \in \mathcal{C}} \Delta(u, v)$.*

We often view $\mathcal{C} \subseteq \Sigma^n$ as the map's image and codewords $u \in \mathcal{C}$ as function $u : [n] \to \Sigma$.

**Definition 8 (Zero-evaders).** *A* zero-evader *over $\mathbb{F}$ with error $\epsilon_{\mathsf{zero}}$ is a function $\mathsf{ZE} : D_{\mathsf{ZE}} \to \mathbb{F}^m$ such that*

$$\forall \boldsymbol{v} \in \mathbb{F}^m \setminus \{\boldsymbol{0}\}, \mathsf{Pr}_{\rho \leftarrow D_{\mathsf{ZE}}}[\langle \mathsf{ZE}(\rho), \boldsymbol{v} \rangle = 0] \leq \epsilon_{\mathsf{zero}}. \tag{41}$$

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

$$\mathsf{ZE}(\boldsymbol{\alpha}) := (\tilde{G}(\boldsymbol{\alpha}, \boldsymbol{b}))_{\boldsymbol{b} \in \{0,1\}^{\log k}}$$

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

**Definition 9 (Out-of-domain samples).** *Let $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a linear code, $f : [n] \to \mathbb{F}$ be a function, $s \in \mathbb{N}$ be a repetition parameter, and $\delta \in [0, 1]$ be a distance parameter. Let $\mathsf{ZE} : D_{\mathsf{ZE}} \to \mathbb{F}^k$ be a zero-evader with error $\epsilon_{\mathsf{zero}}$. Then,*

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

## A.2 Polynomial Commitment Scheme

**Definition 10 (Multilinear polynomial commitment scheme [44]).** *A multilinear polynomial commitment scheme* $\mathsf{PC}$ *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 parameters $\lambda$ and the variable length $n \in \mathbb{N}$, outputs commitment key $\mathsf{ck}$ for polynomials in $\mathcal{F}_n^{(<2)}$.*
- $\mathsf{Commit}(\mathsf{ck}, p(\boldsymbol{X})) \to C$*: takes a multilinear polynomial $p(\boldsymbol{X}) \in \mathcal{F}_n^{(<2)}$ and outputs a commitment $C$.*
- $\mathsf{Open}(\mathsf{ck}, C, p(\boldsymbol{X})) \to b$ *takes a commitment $C$ and a multilinear polynomial $p(\boldsymbol{X}) \in \mathcal{F}_n^{(<2)}$, outputs a bit $b$.*

- $\mathsf{Eval}(\mathcal{P}(p(\boldsymbol{X})), \mathcal{V}(\mathsf{ck}, C, \boldsymbol{z}, y))$ *is a protocol between* $\mathcal{P}$ *and* $\mathcal{V}$ *with public inputs a commitment* $C$*, an evaluation point* $\boldsymbol{z} \in \mathbb{F}^n$ *and a value* $y \in \mathbb{F}$*.* $\mathcal{P}$ *additionally knows a polynomial* $p(\boldsymbol{X}) \in \mathcal{F}_n^{(<2)}$ *and aims to convince* $\mathcal{V}$ *that* $p(\boldsymbol{X})$ *is an opening to* $C$ *and* $p(\boldsymbol{z}) = y$*. The verifier outputs a bit* $b$*.*

For the security properties, we require the multilinear PCS scheme above to satisfy *completeness*, *binding* property, and *knowledge soundness* (also known as extractability in [23]) as defined in Appendix A.2.

For security definitions, we require the $\mathsf{PC}$ in Definition 10 to retain completeness, binding, and knowledge soundness.

**Definition 11 (Completeness).** *The scheme* $\mathsf{PC}$ *satisfies* completeness *if for any bound* $n \in \mathbb{N}$*, for any polynomial* $p(\boldsymbol{X}) \in \mathcal{F}_n^{(<2)}$ *and any point* $\boldsymbol{z} \in \mathbb{F}^n$*,*

$$\mathsf{Pr}\left[\mathsf{Eval}(\mathcal{P}(p(\boldsymbol{X})), \mathcal{V}(\mathsf{ck}, C, \boldsymbol{z}, y)) = 1 \;\middle|\; \begin{matrix} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, n) \\ C \leftarrow \mathsf{Commit}(\mathsf{ck}, p(\boldsymbol{X})) \end{matrix}\right] = 1.$$

**Definition 12 (Binding.).** *The scheme* $\mathsf{PC}$ *is (computationally)* binding *if for any* $n \in \mathbb{N}$ *and PPT adversary* $\mathcal{A}$*,*

$$\mathsf{Pr}\left[\begin{matrix} b_0 = b_1 = 1 \\ \wedge\ p_0(\boldsymbol{X}) \neq p_1(\boldsymbol{X}) \end{matrix} \;\middle|\; \begin{matrix} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, n) \\ (C, p_0(\boldsymbol{X}), p_1(\boldsymbol{X})) \leftarrow \mathcal{A}(\mathsf{ck}) \\ b_0 \leftarrow \mathsf{Open}(\mathsf{ck}, C, p_0(\boldsymbol{X})) \\ b_1 \leftarrow \mathsf{Open}(\mathsf{ck}, C, p_1(\boldsymbol{X})) \end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

**Definition 13 (Knowledge soundness).** $\mathsf{PC}$ *satisfies* knowledge soundness *if* $\mathsf{Eval}$ *is an argument of knowledge for the relation*

$$\mathcal{R}_{\mathsf{Eval}}(\mathsf{ck}) := \{(C, \boldsymbol{z}, y; p(\boldsymbol{X})) : p(\boldsymbol{z}) = y \wedge \mathsf{Open}(\mathsf{ck}, C, p(\boldsymbol{X})) = 1\}$$

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

$$\mathsf{Pr}\left[\begin{matrix} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, n) \\ (C, \boldsymbol{z}, y) \leftarrow \mathcal{A}(\mathsf{ck}, r) \\ \mathsf{Eval}(\tilde{\mathcal{P}}(r), \mathcal{V}(\mathsf{ck}, C, \boldsymbol{z}, y)) = 1 \end{matrix}\right] \approx \mathsf{Pr}\left[\begin{matrix} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, n) \\ (C, \boldsymbol{z}, y) \leftarrow \mathcal{A}(\mathsf{ck}, r) \\ p(\boldsymbol{X}) \leftarrow \mathcal{E}(\mathsf{ck}, r) \\ (C, \boldsymbol{z}, y; p(\boldsymbol{X})) \in \mathcal{R}_{\mathsf{Eval}}(\mathsf{ck}) \end{matrix}\right]$$

### A.3 Multi-Instance Accumulation Schemes

We adopt the definition of the accumulation scheme for NARK given in BCLMS21 [16]. To accommodate the Virtual Machines setting, we explicitly define the notations related to the multi-predicate instance as a special case of the multiple accumulation scheme [17]). Specifically, we define a non-interactive argument of knowledge $\mathsf{NARK} := (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ in the random oracle model for an indexed relation $\mathcal{R}$ relative to a radnom oracle $\mathsf{RO}$ such that the proofs have a canonical partition into pairs $\pi := (\pi.\mathbb{x}, \pi.\mathbb{w})$. Typically, the NARK system should satisfy the following security properties.

**Completeness.** NARK is complete if for every (unbounded) adversary $\mathcal{A}$, the following probability equals to 1

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

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

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

**Definition 14 (Multi-instance accumulation schemes).** *A multi-instance accumulation scheme for* NARK *in the random oracle model consists of a tuple of algorithms as* $\mathsf{ACC}=(\mathcal{G},\mathcal{I},\mathcal{P},\mathcal{V},\mathcal{D})$*, where* $\mathcal{G}$ *is the same as* NARK*,* $\mathcal{I},\mathcal{P},\mathcal{V}$ *have access to the same random oracle* RO*:*

- $\mathsf{ACC}.\mathcal{G}(1^\lambda)\to\mathsf{pp}$*: samples and outputs public parameters* pp*.*
- $\mathsf{ACC}.\mathcal{I}^{\mathsf{RO}}(\mathsf{pp},\mathbb{i})\to(\mathsf{apk},\mathsf{avk},\mathsf{adk})$*: takes as inputs public parameters* pp*, index* $\mathbb{i}$*, and access to random oracle* RO*,* $\mathsf{ACC}.\mathcal{I}$ *deterministically computes and outputs a triple* $(\mathsf{apk},\mathsf{avk},\mathsf{adk})$ *consisting a pair of accumulator proving and verification key* $(\mathsf{apk},\mathsf{avk})$*, and a decision key* adk*.*
- $\mathsf{ACC}.\mathcal{P}^{\mathsf{RO}}(\mathsf{apk},\{\mathbb{x}^{(k)}\}_k,\pi,\mathsf{acc})\to(\mathsf{acc}',\mathsf{pf})$ *takes as inputs multiple predicate instances* $\{\mathbb{x}^{(k)}\}_k$ *and a NARK proof* $\pi=(\pi.\mathbb{x},\pi.\mathbb{w})$*, and an old accumulator* $\mathsf{acc}=(\mathsf{acc}.\mathbb{x},\mathsf{acc}.\mathbb{w})$*,* $\mathsf{ACC}.\mathcal{P}$ *outputs a new accumulator* $\mathsf{acc}'=(\mathsf{acc}'.\mathbb{x},\mathsf{acc}'.\mathbb{w})$ *and an accumulation proof* pf*.*
- $\mathsf{ACC}.\mathcal{V}^{\mathsf{RO}}(\mathsf{avk},\{\mathbb{x}^{(k)}\}_k,\pi.\mathbb{x},\mathsf{acc}.\mathbb{x},\mathsf{acc}'.\mathbb{x},\mathsf{pf})\to b$*: takes as inputs* avk*, multiple predicate instances* $\{\mathbb{x}^{(k)}\}_k$*, a NARK proof instance* $\pi.\mathbb{x}$*, an old accumulator instance* $\mathsf{acc}.\mathbb{x}$*, a new accumulator instance* $\mathsf{acc}'.\mathbb{x}$*, and an accumulation proof* pf*,* $\mathsf{ACC}.\mathcal{V}$ *outputs a bit for accepting or rejecting the proof.*
- $\mathsf{ACC}.\mathcal{D}(\mathsf{adk},\mathsf{acc})\to b$ *takes inputs* $\mathsf{adk}:=(\mathsf{pp},\mathbb{i}')$*, and an accumulator* acc*,* $\mathsf{ACC}.\mathcal{D}$ *outputs a bit indicating whether* acc *is a valid accumulator for the index* $\mathbb{i}'$*.*

The multi-instance accumulation scheme ACC must satisfy two properties, completeness and knowledge soundness. The (perfect completeness) requires that given any valid NARK proof $\pi$ and valid accumulator acc, the proof $\pi$ and the

new accumulator $\mathsf{acc}'$ generated by the honest prover can always pass the verification and decider algorithms. The knowledge soundness requires the existence of an extractor algorithm outputting a valid NARK proof witness $\pi.\mathbb{w}$ and a valid accumulator witness $\mathsf{acc}.\mathbb{w}$ by black-box accessing a malicious prover. The formal definitions are given below.

**Completeness.** For every (unbounded) adversary $\mathcal{A}$, the probability $\mathsf{Pr}[E_1|E_2]$ equals to 1, where

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

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

**Knowledge soundness.** ACC is knowledge sound with respect to auxiliary input distribution $\chi$ if for every non-uniform polynomial-time malicious prover $\tilde{\mathcal{P}}$, there exists a polynomial-time extractor $\mathcal{E}$ such that the probability $\mathsf{Pr}[E_1|E_2]$ is negligibly close to 1:

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

$$E_2 = \left\{ \begin{array}{r} \mathsf{RO} \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{ACC}.\mathcal{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \chi(1^\lambda) \\ (\mathbb{i}, \{\mathbb{x}^{(k)}\}_k, \pi.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}', \mathsf{pf}; \mathsf{tr}) \leftarrow \tilde{\mathcal{P}}^{\mathsf{RO}}(\mathsf{apk}, \mathsf{ai}) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{NARK}.\mathcal{I}(\mathsf{pp}, \mathcal{I}) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{adk}) \leftarrow \mathsf{ACC}.\mathcal{I}(\mathsf{pp}, \mathbb{i}) \\ (\pi.\mathbb{w}, \mathsf{acc}.\mathbb{w}) \leftarrow \mathcal{E}(\mathsf{pp}, \mathbb{i}, \{\mathbb{x}^{(k)}\}_k, \pi.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}', \mathsf{pf}; \mathsf{ai}, \mathsf{tr}) \end{array} \right\}.$$

### A.4 Definitions for Interactive Oracle Reductions

**Definition 15 (Completeness).** *IOR is complete if the following holds. For any* $(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathbb{w}) \in \mathcal{R}$*,*

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

**Definition 16 (Soundness).** *For every proximity bound $\delta \in (0, \delta^*)$, statement $(\mathbb{i}, \mathbb{x}, \mathbb{y}) \notin \mathcal{L}(\mathcal{R})$ with $\Delta_{\mathcal{R}}(\mathbb{i}, \mathbb{x}, \mathbb{y}) > \delta$, and unbounded malicious prover $\mathcal{P}^*$,*

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

Note that we omit $\delta$ when there is no proximity gap, i.e., $\delta = \delta^* = 0$.

We follow the definition given in WARP [15] to cover IORs built on linear codes that do not have an efficient error-tolerant decoder.

**Definition 17 (Knowledge state function for IOR [15]).** *Let $\mathsf{IOR} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ be a $\mu$-round IOR from a relation $\mathcal{R}$ to $\mathcal{R}'$ with proximity radius $\delta^*$. A* knowledge state function *for $\mathsf{IOR}$ is a (possibly inefficient) function $\mathsf{State}$, parameterized by a proximity bound $\delta \in [0, \delta^*)$ that, on input a statement $\mathbb{x}, \mathbb{y}$, an interaction transcript $\mathsf{tr}$, and knowledge state witness $\mathbb{w}$, outputs a bit and has the following syntax:*

- *Empty transcript: if $\mathsf{tr} = \emptyset$, then $\mathsf{State}_\delta(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbb{w}) = 1$ if and only if there exists $\mathbb{y}_*$ satisfying $(\mathbb{i}, \mathbb{x}, \mathbb{y}_*, \mathbb{w}) \in \mathcal{R}$ and $\Delta(\mathbb{y}, \mathbb{y}_*) \leq \delta$.*
- *Prover moves: if $\mathsf{tr}$ is a transcript where the prover is about to move, $\mathsf{State}_\delta(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbb{w}) = 0$, then for every prover message $\pi$, $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}||\pi, \mathbb{w}) = 0$.*
- *Full transcript: if $\mathsf{tr} = (\pi_1, r_1, \ldots, \pi_\mu, r_\mu)$ is a full transcript, then $\mathsf{State}_\delta(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbb{w}) = 1$ if and only if $V^{\mathbb{I},\mathbb{y},\{\pi_i\}_{i\in[k]}}(\mathbb{x}, \{r_i\}_{i\in[k]})$ outputs $\mathbb{x}', \mathbb{y}'$ such that there exist $\mathbb{y}'_*$ satisfying $(\mathbb{i}', \mathbb{x}', \mathbb{y}'_*, \mathbb{w}) \in \mathcal{R}'$ and $\Delta(\mathbb{y}_*, \mathbb{y}'_*) \leq \delta$.*

**Definition 18 (A variant of round-by-round knowledge soundness [15]).** *A $\mu$-round $\mathsf{IOR} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ from relation $\mathcal{R}$ to $\mathcal{R}'$ with proximity radius $\delta^*$ has RBR knowledge soundness errors $(\epsilon_1, \ldots, \epsilon_\mu)$ and extraction time $(\mathsf{et}_1, \ldots, \mathsf{et}_\mu)$ if there exist a knowledge state function $\mathsf{State}$ for $\mathsf{IOR}$ and a deterministic extractor $\mathcal{E}$ with the following property: for every proximity bound $\delta \in (0, \delta^*)$, statement $(\mathbb{i}, \mathbb{x}, \mathbb{y})$, round index $i \in [\mu]$, and interaction transcript $\mathsf{tr} = (\pi_1, r_1, \ldots, \pi_{i-1}, r_{i-1}, \pi_i), i \in [\mu]$ where the verifier is about to move, $\mathcal{E}$ runs in time at most $\mathsf{et}_i$ and*

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

The authors in [15] also prove that the above RBR knowledge soundness implies (straightline) state-restoration knowledge soundness to ensure Fiat–Shamir security. According to previous conclusions [7,18,20], there exists a transform from a public-coin interactive oracle reduction $\mathsf{IOR}$ to a non-interactive reduction $\mathsf{NIR} := \mathsf{FS}[\mathsf{IOR}^{\mathsf{PC}}]$ based on a polynomial commitment scheme $\mathsf{PC}$ and Fiat-Shamir transform $\mathsf{FS}$.

Next, we highlight the definition of non-interactive reductions. We adopt the conclusion for polynomial IOP compilation given in [21] to the polynomial IOR here. The proof of Theorem 4 can be referred to [20].

**Theorem 4 (Polynomial IOR compilation adopted from [21]).** *There is a transformation $T$ from public-coin polynomial interactive oracle reduction $\mathsf{IOR}$ to a non-interactive reduction $\mathsf{NIR}$ with compilation in two steps:*

- *replace every oracle in* $\mathsf{IOR}$ *with a polynomial commitment* $\mathsf{PC}$ *to obtain* $\mathsf{IOR}^{\mathsf{PC}}$*;*
- *apply the Fiat-Shamir transform to the committed* $\mathsf{IOR}^{\mathsf{PC}}$ *to obtain a non-interactive reduction* $\mathsf{FS}[\mathsf{IOR}^{\mathsf{PC}}]$.

*Denote the resulting protocol as* $\mathsf{NIR} := \mathsf{FS}[\mathsf{IOR}^{\mathsf{PC}}]$. *If* $\mathsf{IOR}$ *is RBR knowledge sound and* $\mathsf{PC}$ *is extractable, then* $\mathsf{NIR}$ *is RBR knowledge sound under the random oracle model. The efficiency of* $\mathsf{NIR}$ *depends on the efficiency of* $\mathsf{IOR}$ *and* $\mathsf{PC}$*:*

- *The prover time is equal to the sum of (1) prover time of* $\mathsf{IOR}$*, (2) the oracle length times the commitment time, and (3) the query complexity times the prover time of* $\mathsf{PC}$.
- *The verifier time is equal to the sum of (1) verifier time of* $\mathsf{IOR}$*, and (2) the verifier time of* $\mathsf{PC}$ *times the query complexity of* $\mathsf{IOR}$.
- *The proof size is equal to the sum of (1) the message complexity of the* $\mathsf{IOR}$ *times the commitment size and (2) the query complexity times the proof size of* $\mathsf{PC}$.

### A.5 Definitions for Special Sound Protocols

For an $\mathcal{NP}$ relation $\mathcal{R} := \{(\mathsf{x}; \mathsf{w})\}$, an interactive proof for $\mathcal{R}$ allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that a statement $\mathsf{x}$ admits a witness $\mathsf{w}$. Typically, in a $(2\mu - 1)$-move protocol, the prover and the verifier interactively send $\mu$ messages and $\mu - 1$ challenges. If $\mathcal{V}$ generates and sends all its messages uniformly at random and independently of the prover's messages, then the protocol is called a *public coin* protocol. Public coin protocols can be transformed into non-interactive forms according to the Fiat-Shamir heuristic [27]. For security definitions, we require the interactive proof to satisfy completeness and knowledge soundness. Furthermore, to simplify the security proof, researchers may refer to the *special soundness* definition that implies knowledge soundness (see Lemma 7). Briefly speaking, an interactive proof satisfies special soundness if there exists an efficient extractor outputting a valid witness with inputs of the instance $\mathbb{x}$ and a tree of transcripts, which is a collection of accepting transcripts generated by a malicious prover, where the challenges differ in a specific pattern across the transcripts. For defining the special soundness property, we first present a denotation of the tree of transcript.

**Definition 19 (Tree of transcripts).** *Let* $k \in \mathbb{N}$ *and* $(a_1, ..., a_k) \in \mathbb{N}^k$. *A* $(a_1, ..., a_k)$*-tree of transcripts constitutes a set of* $\prod_{i=1}^{k} a_i$ *transcripts of a tree-like structure. The edges within this tree represent the challenges of the verifier, while the vertices are the messages from the prover, which can be empty. Each node at depth* $i$ *has* $a_i$ *child nodes, corresponding to* $a_i$ *distinct challenges. Every transcript is uniquely represented by one path from the root node to a leaf node.*

**Definition 20 ($(a_1, ..., a_k)$-special soundness).** *$\Pi$ provides* $(a_1, ..., a_k)$*-special soundness, if there is an effective PPT extraction algorithm* $\mathcal{E}$ *that is capable of extracting the witness* $w$ *given* $x$ *and any* $(a_1, ..., a_k)$*-tree of accepting transcripts* $T$ *[4] (defined above). Specifically, for all PPT adversaries* $\mathcal{A}$

$$\Pr\left[\, \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), (x, T) \leftarrow \mathcal{A}(\mathsf{pp}), \mathsf{w} \leftarrow \mathcal{E}(\mathsf{pp}, \mathsf{x}, T) : (\mathsf{x}; \mathsf{w}) \in \mathcal{R} \,\right] \approx 1.$$

According to the conclusion given in [3], we can imply knowledge soundness from special soundness:

**Lemma 7 ($(a_1, ..., a_k)$-special soundness implies knowledge soundness [4]).** *Let $k, a_1, ..., a_k \in \mathbb{N}$, $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ be a $(a_1, ..., a_k)$-special sound $(2k+1)$-move interactive protocol for relation $\mathcal{R}$, where $\mathcal{V}$ samples each challenge uniformly at random from $\mathbb{F}$. Then $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is knowledge sound with knowledge error*

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

### A.6 Definitions for Multi-Instance IVC

We extend the definitions of IVC in [34] with a single instance at each step to a multi-instance IVC by referring to [16]. To achieve generality, we also consider multiple predicate settings, i.e., at each step $i$ the IVC prover is allowed to select a distinct predicate $\varphi_i$ from a specified set $\Phi$.

**Definition 21 (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:*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$ *on input security parameter $\lambda$, samples and outputs public parameter* $\mathsf{pp}$.
- $\mathcal{I}(\mathsf{pp}, \boldsymbol{\varphi}) \to (\mathsf{ipk}, \mathsf{ivk})$ *on input public parameter* $\mathsf{pp}$*, a predicate vector $\boldsymbol{\varphi} \subseteq \Phi$, outputs a prover key* $\mathsf{ipk}$ *and a verifier key* $\mathsf{ivk}$.
- $\mathcal{P}(\mathsf{ipk}, i, \{z_i^{(k)}, w_i^{(k)}, z_{i+1}^{(k)}\}_k, \Pi_i) \to \Pi_{i+1}$ *on input public key* $\mathsf{ipk}$*, a counter $i$, multiple public inputs $\{z_i^{(k)}\}_k$, private inputs $\{w_i^{(k)}\}_k$, outputs $\{z_{i+1}^{(k)}\}_k$, and an IVC proof $\Pi_i$, the prover outputs a new IVC proof $\Pi_{i+1}$ to attest the correctness of $\{z_{i+1}^{(k)}\}_k$.*
- $\mathcal{V}(\mathsf{ivk}, i, \{z_i^{(k)}\}_k, \Pi_i) \to b$ *on input verifier key* $\mathsf{ivk}$*, a counter $i$, multiple public inputs $\{z_i^{(k)}\}_k$, and an IVC proof $\Pi_i$, the verifier checks whether $\Pi_i$ is valid for testing $\{z_i^{(k)}\}_k$, and outputs a bit $b$ to indicate reject or accept.*

A multi-instance IVC quadruple $(\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ should satisfy the perfect completeness, (computational) knowledge soundness, and (statistical) zero knowledge properties as defined below.

**Perfect Completeness.** $\mathsf{IVC}$ has perfect completeness if for every adversary $\mathcal{A}$, the probability $\Pr[E_1|E_2]$ equals to 1, where

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

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

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

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

and

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

where $\mathsf{ai}$ and $\mathsf{ao}$ are auxiliary inputs and outputs. These modifications aim to ensure closeness in distribution between the outputs of the prover and the extractor for the strong extraction guarantee [9].

According to [16], such an IVC/PCD can be constructed from a NARK with its corresponding accumulation scheme, we draw a similar conclusion for our multi-instance IVC in Theorem 5, of which the proof can be referred to [16].

**Theorem 5 (Multi-instance IVC from accumulation schemes [16]).** *Given a NARK for circuit satisfiability and a multi-instance accumulation scheme for that NARK under 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, the PCD is also secure against quantum adversaries. If* both *the NARK and accumulation scheme are zero-knowledge, the PCD is also zero-knowledge.*

# B Complementary Protocols

## B.1 Non-Interactive Multi-Cast Reduction for SPS

We define the non-interactive prover and verifier algorithms for the non-interactive reduction $\mathsf{NIR}_{\mathsf{cast}} := (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ with respect to a random oracle RO as mentioned in Section 4.2. Assume the generator algorithm outputs public parameters as $\mathsf{pp} \leftarrow \mathsf{NIR}_{\mathsf{cast}}.\mathcal{G}(1^\lambda)$, the indexer algorithm outputs a pair of key and a new index as $(\mathsf{pk}, \mathsf{vk}, \mathbb{i}') \leftarrow \mathsf{NIR}_{\mathsf{cast}}.\mathcal{I}(\mathsf{pp}, \mathbb{i})$. We describe the prover and verifier algorithms in Figures 7 and 8.

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

1 : $r_0 \leftarrow \mathsf{RO}(\{\mathbb{x}^{(k)}\}_{k\in[\ell]})$

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

$a$ : Set $\boldsymbol{m}_i^{(k)} \leftarrow \mathsf{P}_{\mathsf{sps}}(\mathbb{x}^{(k)}, \mathbb{w}^{(k)}, \{\boldsymbol{m}_j^{(k)}, r_j\}_{j=0}^{i-1})\ \forall k \in [\ell]$

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

$c$ : Set $C_{\cup,i} = \mathsf{Commit}(\mathsf{ck}, \tilde{m}_{\cup,i}(\boldsymbol{Y}, \boldsymbol{X}))$

$d$ : $r_i \leftarrow \mathsf{RO}(r_{i-1}, C_{\cup,i})$

3 : Set $\alpha := r_\mu, \boldsymbol{\alpha} := (\alpha, \alpha^2, \ldots, \alpha^{2^{\log m - 1}})$

4 : $\boldsymbol{r}_y \leftarrow \mathsf{RO}(\boldsymbol{\alpha})$

5 : Sets $G(\boldsymbol{Y}) := F(\sum_{k\in[\ell]} \tilde{eq}_{k-1}(\boldsymbol{Y}) \cdot \mathbb{x}^{(k)}, \{\sum_{k\in[\ell]} \tilde{eq}_{k-1}(\boldsymbol{Y}) \cdot \boldsymbol{m}_i^{(k)}\}_{i\in[\mu]}, \boldsymbol{r}_F) \cdot \tilde{eq}(\boldsymbol{Y}, \boldsymbol{r}_y)$

6 : Runs a $(\log \ell)$-round non-interactive sumcheck for $\sum_{\boldsymbol{y}\in B_{\log \ell}} G(\boldsymbol{y}) = 0$,

and obtains $\log \ell$ sumcheck polynomials, and the random vector $\boldsymbol{\tau} \in \mathbb{F}^{\log \ell}$

7 : Sets $e := G_{\log \ell}(\tau_{\log \ell})$

8 : Sets $\tilde{m}_i(\boldsymbol{X}) := \tilde{m}_{\cup,i}(\boldsymbol{\tau}, \boldsymbol{X})\ \forall i \in [\mu]$

9 : Sets $C_i = \mathsf{Commit}(\mathsf{ck}, \tilde{m}_i(\boldsymbol{X}))\ \forall i \in [\mu]$

10 : $\boldsymbol{r}_x \leftarrow \mathsf{RO}(\boldsymbol{r}_y, \{C_i\}_{i\in[\mu]})$

11 : $\mathcal{P}$ implicitly outputs the witness $\mathbb{w}_{\mathsf{acc}}$ and the proof $\pi$ including:

$a$ : Commitments$\{C_{\cup,i}\}_{i\in[\mu]}, \{C_i\}_{i\in[\mu]}$

$b$ : Challenges $\boldsymbol{r}_F, \boldsymbol{\tau}, \boldsymbol{r}_x$

$c$ : Sumcheck proofs $\{G_i(Y)\}_{i\in[\mu]}, G_{\log \ell}(\tau_{\log \ell})$

**Fig. 7.** The Prover Algorithm $\mathsf{NIR}_{\mathsf{cast}}.\mathcal{P}^{\mathsf{RO}}$ for Non-Interactive Oracle Reduction

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

1 : Parse $\pi = \{C_{\cup,i}\}_{i\in[\mu]}, \{C_i\}_{i\in[\mu]}, \boldsymbol{r}_F, \boldsymbol{\tau}, \boldsymbol{r}_x, \{G_i(Y)\}_{i\in[\mu]}, e$

2 : Parse $\boldsymbol{r}_F = \boldsymbol{r}||\boldsymbol{\alpha}$

3 : Set $b_1 = 1$ if

- $a$ : $r_0 \leftarrow \mathsf{RO}(\{\mathbb{x}^{(k)}\}_{k\in[\ell]})$
- $b$ : $r_i \leftarrow \mathsf{RO}(r_{i-1}, C_{\cup,i})\ \forall i \in [\mu]$
- $c$ : $G_1(0) + G_1(1) = 0$
- $d$ : $G_{i+1}(0) + G_{i+1}(1) = G_i(\tau_i),\ \forall i \in [\log \ell - 1]$

4 : Set $b_2 = 1$ if

- $a$ : $\boldsymbol{r}_y \leftarrow \mathsf{RO}(\boldsymbol{\alpha})$
- $b$ : $\tau_i = \mathsf{RO}(G_i(Y)), i \in [\log \ell]$
- $c$ : $\boldsymbol{r}_x \leftarrow \mathsf{RO}(\boldsymbol{r}_y, \{C_i\}_{i\in[\mu]})$

5 : Sets $e = G_{\log \ell}(\tau_{\log \ell}) \cdot \tilde{eq}^{-1}(\boldsymbol{r}_y, \boldsymbol{\tau})$

6 : Sets $\boldsymbol{x} = \sum_{k\in\ell} \tilde{eq}_{k-1}(\boldsymbol{\tau}) \cdot \boldsymbol{x}^{(k)}$

7 : Outputs instance: $\mathbb{x}_{\mathsf{acc}} := (\boldsymbol{x}, \{C_{\cup,i}\}_{i\in[\mu]}, \{C_i\}_{i\in[\mu]}, \boldsymbol{r}_F, \boldsymbol{\tau}, \boldsymbol{r}_x, e)$

**Fig. 8.** The Verifier Algorithm $\mathsf{NIR}_{\mathsf{cast}}.\mathcal{V}^{\mathsf{RO}}$ for Non-Interactive Oracle Reduction

### B.2 Non-Interactive 2-to-1 Reduction

We define the non-interactive prover and verifier algorithms for the non-interactive reduction $\mathsf{NIR}_{\mathsf{multicast}} := (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ with respect to a random oracle $\mathsf{RO}$ as mentioned in Section 4.2. Assume the generator algorithm outputs public parameters as $\mathsf{pp} \leftarrow \mathsf{NIR}_{\mathsf{fold}}.\mathcal{G}(1^\lambda)$, the indexer algorithm outputs a pair of key and a new index as $(\mathsf{pk}, \mathsf{vk}, \mathbb{i}') \leftarrow \mathsf{NIR}_{\mathsf{fold}}.\mathcal{I}(\mathsf{pp}, \mathbb{i})$. We describe the prover and verifier algorithms in Figures 9 and 10.

### B.3 Construction for Accumulation Scheme

**NARK construction** First, we build the NARK system from the non-interactive reduction $\mathsf{NIR}_{\mathsf{cast}}$. The general idea is simple. Given an instance-witness tuple $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_F^{\mathsf{PC}}$, the prover and verifier can respectively run the proving and verifying algorithms of reduction $\mathsf{NIR}_{\mathsf{cast}}$ from $\mathcal{R}_F^{\mathsf{PC}}$ to $\mathcal{R}_{\mathsf{acc}}^{\mathsf{PC}}$. Since the NARK system does not require the proof to be succinct, the prover can directly sends the reduced witness $\mathbb{w}_{\mathsf{acc}}$ as the argument proof. The verifier simply derives the reduced instance $\mathbb{x}_{\mathsf{acc}}$ and oracle string $\mathbb{y}_{\mathsf{acc}}$ and then checks whether $(\mathbb{x}_{\mathsf{acc}}, \mathbb{y}_{\mathsf{acc}}, \mathbb{w}_{\mathsf{acc}})$ satisfies the relation $\mathcal{R}_{\mathsf{acc}}^{\mathsf{PC}}$. We sketch the algorithms for $\mathsf{NARK} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ as follows:

- $\mathsf{NARK}.\mathcal{G}(1^\lambda)$ takes inputs as the security parameter $\lambda$ in unary and outputs public parameters $\mathsf{pp}$.

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

1 : $\boldsymbol{\gamma}, r_z \leftarrow \mathsf{RO}(\{\mathsf{acc}^{(i)}.\mathbb{x}, \mathsf{acc}^{(i)}.\mathbb{w}\}_{i\in[1]})$

2 : Sets $G(Z)$ as per Equation 31.

3 : Runs a 1-round non-interactive sumcheck for $\sum_{z\in\{0,1\}} G(z) = 0$,
and obtains a sumcheck polynomial $G(Z)$, and a random value $\sigma \in \mathbb{F}$

4 : Sets $v_G := G(\sigma)$

5 : Computes batched witness $\{\tilde{\boldsymbol{w}}_{\cup,i}(\sigma), \tilde{\boldsymbol{m}}_i(\sigma)\}_{i\in[\mu]}$

6 : Computes $\eta, \{\eta_i, \eta_{\cup,i}\}_{i\in[\mu]}$ as per Equations 32-34

7 : Computes $\tilde{e}(\sigma), \{\tilde{v}_i(\sigma)\}_{i\in[\mu]}$

8 : Runs $2\mu$ times oracle batching proving algorithm $\mathsf{NIR}_{\mathsf{batch}}.\mathcal{P}$ and obtains
batched commitments $\{C_{\cup,i}\}_{i\in[\mu]}, \{C_i\}_{i\in[\mu]}$, and proof $\pi_{\mathsf{batch}}$

9 : $\mathcal{P}$ implicitly outputs the witness $\mathsf{acc}.\mathbb{w}$ and the proof $\pi$ including:

$a$ : Challenges $\boldsymbol{\gamma}, r_z, \sigma$

$b$ : Sumcheck proofs $G(Z), v_G$

$c$ : Evaluation claims $\eta, \{\eta_i, \eta_{\cup,i}\}_{i\in[\mu]}$

$d$ : Commitments $\{C_{\cup,i}\}_{i\in[\mu]}, \{C_i\}_{i\in[\mu]}$

$e$ : Batch proof $\pi_{\mathsf{batch}}$

**Fig. 9.** The Prover Algorithm $\mathsf{NIR}_{\mathsf{fold}}.\mathcal{P}^{\mathsf{RO}}$ for Non-Interactive Oracle Reduction

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

1 : $\boldsymbol{\gamma}, r_z \leftarrow \mathsf{RO}(\{\mathsf{acc}^{(i)}.\mathbb{x}, \mathsf{acc}^{(i)}.\mathbb{w}\}_{i\in[1]})$

2 : Checks if $G(0) + G(1) = 0$

3 : Checks if $v_G = \tilde{eq}(r_z, \sigma) \cdot (\eta + \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{\gamma}) \cdot \eta_i + \sum_{i\in[\mu]} \mathsf{pow}_{i+\mu}(\boldsymbol{\gamma}) \cdot \eta_{\cup,i})$

4 : Computes $\tilde{e}(\sigma), \{\tilde{v}_i(\sigma)\}_{i\in[\mu]}$

5 : Runs $2\mu$ times oracle batching verifying algorithm $\mathsf{NIR}_{\mathsf{batch}}.\mathcal{V}$ to check
batched commitments $\{C_{\cup,i}\}_{i\in[\mu]}, \{C_i\}_{i\in[\mu]}$, with proof $\pi_{\mathsf{batch}}$

6 : Parses $\{\boldsymbol{y}_j, \boldsymbol{x}_j.v_j\}_j, \{\boldsymbol{x}_t, v_t\}_t$ from $\pi_{\mathsf{batch}}$

7 : Computes the batched instance $\tilde{\boldsymbol{x}}(\sigma)$

8 : Outputs instance:
$\mathsf{acc}.\mathbb{x} := (\boldsymbol{x}, \{C_{\cup,i}\}_{i\in[\mu]}, \{C_i\}_{i\in[\mu]}, \boldsymbol{r}_F, e, \{\boldsymbol{y}_j, \boldsymbol{x}_j, v_j\}_j, \{\boldsymbol{x}_t, v_t\}_t)$

**Fig. 10.** The Verifier Algorithm $\mathsf{NIR}_{\mathsf{fold}}.\mathcal{V}^{\mathsf{RO}}$ for Non-Interactive Oracle Reduction

- $\mathsf{NARK}.\mathcal{I}(\mathsf{pp}, \mathbb{i})$ takes inputs as $\mathsf{pp}$ and an index $\mathbb{i}$, the indexer computes $(\mathsf{pk}_{\mathsf{cast}}, \mathsf{vk}_{\mathsf{cast}}, \mathbb{i}') \leftarrow \mathsf{NIR}_{\mathsf{cast}}.\mathcal{I}^{\mathsf{RO}}(\mathsf{pp}, \mathbb{i})$, and outputs $(\mathsf{pk}, \mathsf{vk}) := (\mathsf{pk}_{\mathsf{cast}}, (\mathsf{vk}_{\mathsf{cast}}, \mathbb{i}', \mathsf{pp}))$.
- $\mathsf{NARK}.\mathcal{P}^{\mathsf{RO}}(\mathsf{pk}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_{k\in[\ell]})$ takes inputs as $\mathsf{pk}$ and an instance-witness pair $\{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_{k\in[\ell]}$ satisfying the multi-instance committed relation $(\mathcal{R}_F^{\mathsf{PC}})^\ell$, the prover first computes $(\pi_{\mathsf{cast}}, \mathbb{w}_{\mathsf{acc}}) \leftarrow \mathsf{NIR}_{\mathsf{cast}}.\mathcal{P}^{\mathsf{RO}}(\mathsf{pk}_{\mathsf{cast}}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_{k\in[\ell]})$ and assign $(\pi.\mathbb{x}, \pi.\mathbb{w}) := (\pi_{\mathsf{cast}}, \mathbb{w}_{\mathsf{acc}})$. Then the prover outputs $\pi := (\pi.\mathbb{x}, \pi.\mathbb{w})$.
- $\mathsf{NARK}.\mathcal{V}^{\mathsf{RO}}(\mathsf{vk}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi)$ takes inputs as $\mathsf{vk} = (\mathsf{vk}_{\mathsf{cast}}, \mathbb{i}', \mathsf{pp})$, $\ell$ instances $\{\mathbb{x}^{(k)}\}_{k\in[\ell]}$, and proof $\pi = (\pi.\mathbb{x}, \pi.\mathbb{w})$, the verifier first computes $\mathbb{x}_{\mathsf{acc}} \leftarrow \mathsf{NIR}_{\mathsf{cast}}.\mathcal{V}^{\mathsf{RO}}(\mathsf{vk}_{\mathsf{cast}}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi_{\mathsf{cast}})$ and assign $(\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w}) := (\mathbb{x}_{\mathsf{acc}}, \pi.\mathbb{w})$. Then the verifier checks if $(\mathbb{i}', \mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w}) \in \mathcal{R}_{\mathsf{acc}}^{\mathsf{RO}}(\mathsf{pp})$.

Given that $\mathsf{NIR}_{\mathsf{cast}} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ has completeness and knowledge soundness in the random oracle model according to Lemma 5, it is obvious that the $\mathsf{NARK}$ also has completeness and knowledge soundness in the random oracle model.

*Proof Sketch.* Since $\mathsf{NARK}$ is a trivial wrapper around the non-interactive reduction $\mathsf{NIR}_{\mathsf{cast}}$, completeness follows immediately from the completeness of $\mathsf{NIR}_{\mathsf{cast}}$. For the knowledge soundness, we prove that for an arbitrary polynomial-time adversary $\tilde{\mathcal{P}}$ against $\mathsf{NARK}$, an adversary $\tilde{\mathcal{P}}_{\mathsf{cast}}^{\mathsf{RO}}$ against $\mathsf{NIR}_{\mathsf{cast}}$ can be built as follows:

- compute $(\mathbb{i}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_k, \pi; \mathsf{tr}) \leftarrow \tilde{\mathcal{P}}^{\mathsf{RO}}(\mathsf{pp}, \mathsf{ai})$;
- assign $(\pi_{\mathsf{cast}}, \mathbb{w}_{\mathsf{acc}}) := \pi$;
- output $(\mathbb{i}, \{\mathbb{x}^{(k)}\}, \pi_{\mathsf{cast}}, \mathbb{w}_{\mathsf{acc}}; \mathsf{tr})$.

By the knowledge soundness of the non-interactive reduction $\mathsf{NIR}_{\mathsf{cast}}$, there exists a corresponding extractor $\mathcal{E}_{\mathsf{cast}}$. Then we can construct an extractor for $\mathsf{NARK}$ as follows:

- assign $\pi_{\mathsf{cast}}, \mathbb{w}_{\mathsf{acc}} := \pi$;
- compute $\mathbb{w} \leftarrow \mathcal{E}_{\mathsf{cast}}(\mathsf{pp}, \mathbb{i}, \{\mathbb{x}^{(k)}\}_k, \pi_{\mathsf{cast}}, \mathbb{w}_{\mathsf{acc}}, \mathsf{ai}, \mathsf{tr})$;
- output $\mathbb{w}$.

We claim that the error bound is tight, i.e., the following two probabilities are negligibly close:

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

$$\Pr\left[\begin{array}{l|r} (\mathbb{i}', \mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w}) \in \mathcal{R}_{\mathsf{acc}}^{\mathsf{PC}}(\mathsf{pp}) & \mathsf{RO} \leftarrow \mathcal{U}(\lambda) \\ \wedge & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (\mathbb{i}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_k \notin \mathcal{R}^*(\mathsf{pp})) & \mathsf{ai} \leftarrow \chi(1^\lambda) \\ & (\mathbb{i}, \mathbb{x}, \pi_{\mathsf{cast}}, \mathbb{w}_{\mathsf{acc}}; \mathsf{tr}) \leftarrow \tilde{\mathcal{P}}_{\mathsf{cast}}(\mathsf{pp}, \mathsf{ai}) \\ & (\mathsf{pk}_{\mathsf{cast}}, \mathsf{vk}_{\mathsf{cast}}) \leftarrow \mathsf{NIR}_{\mathsf{cast}}.\mathcal{I}^{\mathsf{RO}}(\mathsf{pp}, \mathbb{i}) \\ & \mathbb{x}_{\mathsf{acc}} \leftarrow \mathsf{NIR}_{\mathsf{cast}}.\mathcal{V}^{\mathsf{RO}}(\mathsf{vk}_{\mathsf{cast}}, \{\mathbb{x}^{(k)}\}_k, \pi_{\mathsf{cast}}) \\ & \mathbb{w} \leftarrow \mathcal{E}_{\mathsf{cast}}(\mathsf{pp}, \mathbb{i}, \{\mathbb{x}^{(k)}\}_k, \pi_{\mathsf{cast}}, \mathbb{w}_{\mathsf{acc}}, \mathsf{ai}, \mathsf{tr}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

**Accumulation scheme** Similarly, we build an accumulation scheme for the $\mathsf{NARK}$ above from the non-interactive reduction $\mathsf{NIR}_{\mathsf{fold}}$. Given a tuple $(\mathtt{i}, \{\mathbb{x}^{(k)}, \mathbb{w}^{(k)}\}_{k\in[\ell]})$ satisfying the multi-instance relation $(\mathcal{R}_F^{\mathsf{PC}})^\ell$, the accumulation prover first runs $\mathsf{NIR}_{\mathsf{cast}}.\mathcal{V}$ to obtain a reduced instance $\mathsf{acc}.\mathbb{x} = \mathbb{x}_{\mathsf{acc}}$. Now that we have a tuple $\{\mathtt{i}, (\mathbb{x}^{(i)}, \mathbb{y}^{(i)}, \mathbb{w}^{(i)})\}_{i\in[1]}$ in multi-instance relation $(\mathcal{R}_{\mathsf{acc}}^{\mathsf{PC}})^2$, the accumulation prover and verifier together runs the reduction $\mathsf{NIR}_{\mathsf{fold}}$ from $(\mathcal{R}_{\mathsf{acc}}^{\mathsf{PC}})^2$ to $\mathcal{R}_{\mathsf{acc}}^{\mathsf{PC}}$. The decider simply checks whether the reduce tuple satisfies $\mathcal{R}_{\mathsf{acc}}$. We sketch the algorithms for $\mathsf{ACC} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V}, \mathcal{D})$ as follows:

- $\mathsf{ACC}.\mathcal{G}(1^\lambda)$ takes inputs as the security parameter $\lambda$ in unary and outputs public parameters $\mathsf{pp}$.
- $\mathsf{ACC}.\mathcal{I}(\mathsf{pp}, \mathtt{i})$ takes inputs as $\mathsf{pp}$ and an index $\mathtt{i}$, the indexer
  - computes $(\mathsf{pk}_{\mathsf{cast}}, \mathsf{vk}_{\mathsf{cast}}, \mathtt{i}') \leftarrow \mathsf{NIR}_{\mathsf{cast}}.\mathcal{I}^{\mathsf{RO}}(\mathsf{pp}, \mathtt{i})$.
  - computes $(\mathsf{pk}_{\mathsf{fold}}, \mathsf{vk}_{\mathsf{fold}}, \mathtt{i}') \leftarrow \mathsf{NIR}_{\mathsf{fold}}.\mathcal{I}^{\mathsf{RO}}(\mathsf{pp}, \mathtt{i})$.
  - outputs $\mathsf{apk} := (\mathsf{vk}_{\mathsf{cast}}, \mathsf{pk}_{\mathsf{fold}}, \mathsf{vk}_{\mathsf{fold}})$, $\mathsf{avk} := (\mathsf{vk}_{\mathsf{cast}}, \mathsf{vk}_{\mathsf{fold}})$, and $\mathsf{adk} := (\mathtt{i}', \mathsf{pp})$.
- $\mathsf{ACC}.\mathcal{P}^{\mathsf{RO}}(\mathsf{apk}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi, \mathsf{acc}^{(0)})$ takes inputs as $\mathsf{apk} := (\mathsf{vk}_{\mathsf{cast}}, \mathsf{pk}_{\mathsf{fold}}, \mathsf{vk}_{\mathsf{fold}})$, $\ell$ instances $\{\mathbb{x}^{(k)}\}_{k\in[\ell]}$, a proof $\pi = (\pi.\mathbb{x}, \pi.\mathbb{w}) = (\pi_{\mathsf{cast}}, \mathbb{w}_{\mathsf{acc}})$ and an accumulator $\mathsf{acc}^{(0)}$. Then the prover
  - computes $\mathbb{x}_{\mathsf{acc}} \leftarrow \mathsf{NIR}_{\mathsf{cast}}.\mathcal{V}^{\mathsf{RO}}(\mathsf{vk}_{\mathsf{cast}}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi_{\mathsf{cast}})$ and assigns $(\mathsf{acc}^{(1)}.\mathbb{x}, \mathsf{acc}^{(1)}.\mathbb{w}) := (\mathbb{x}_{\mathsf{acc}}, \pi.\mathbb{w})$.
  - computes $(\pi_{\mathsf{fold}}, \mathsf{acc}.\mathbb{w}) \leftarrow \mathsf{NIR}_{\mathsf{fold}}.\mathcal{P}^{\mathsf{RO}}(\mathsf{pk}_{\mathsf{fold}}, \{\mathsf{acc}^{(i)}.\mathbb{x}, \mathsf{acc}^{(i)}.\mathbb{w}\}_{i\in[1]})$.
  - computes $\mathsf{acc}.\mathbb{x} \leftarrow \mathsf{NIR}_{\mathsf{fold}}.\mathcal{V}^{\mathsf{RO}}(\mathsf{vk}_{\mathsf{fold}}, \{\mathsf{acc}^{(i)}.\mathbb{x}\}_{i\in[1]}, \pi_{\mathsf{fold}})$.
  - outputs $\mathsf{acc} \leftarrow (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ and $\mathsf{pf} \leftarrow \pi_{\mathsf{fold}}$.
- $\mathsf{ACC}.\mathcal{V}^{\mathsf{RO}}(\mathsf{avk}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi.\mathbb{x}, \mathsf{acc}^{(0)}.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$ takes inputs as $\mathsf{avk} := (\mathsf{vk}_{\mathsf{cast}}, \mathsf{vk}_{\mathsf{fold}})$, $\ell$ instances $\{\{\mathbb{x}^{(k)}\}_{k\in[\ell]}$ and proof $\mathsf{vk} = (\mathsf{vk}_{\mathsf{cast}}, \mathtt{i}', \mathsf{pp})$, instance $\mathbb{x}$, a proof instance $\pi.\mathbb{x} = \pi_{\mathsf{cast}}$, an old accumulator instance $\mathsf{acc}^{(0)}$, a new accumulator instance $\mathsf{acc}.\mathbb{x}$ and accumulation proof $\mathsf{pf}$. Then the verifier
  - computes $\mathbb{x}_{\mathsf{acc}} \leftarrow \mathsf{NIR}_{\mathsf{cast}}.\mathcal{V}^{\mathsf{RO}}(\mathsf{vk}_{\mathsf{cast}}, \{\mathbb{x}^{(k)}\}_{k\in[\ell]}, \pi_{\mathsf{cast}})$ and assigns $(\mathsf{acc}^{(1)}.\mathbb{x}, \mathsf{acc}^{(1)}.\mathbb{w}) := (\mathbb{x}_{\mathsf{acc}}, \pi.\mathbb{w})$.
  - checks that $\mathsf{acc}.\mathbb{x} = \mathsf{NIR}_{\mathsf{fold}}.\mathcal{V}^{\mathsf{RO}}(\mathsf{vk}_{\mathsf{fold}}, \{\mathsf{acc}^{(i)}.\mathbb{x}\}_{i\in[1]}, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$.
- $\mathsf{ACC}.\mathcal{D}(\mathsf{adk}, \mathsf{acc})$ takes inputs as $\mathsf{adk} := (\mathtt{i}', \mathsf{pp})$ and an accumulator $\mathsf{acc}$. The decider checks that $(\mathtt{i}', \mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w}) \in \mathcal{R}_{\mathsf{acc}}^{\mathsf{PC}}(\mathsf{pp})$.

Given that $\mathsf{NIR}_{\mathsf{cast}} = (\mathcal{I}, \mathcal{P}, \mathcal{V}), \mathsf{NIR}_{\mathsf{fold}} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ has completeness and knowledge soundness in the random oracle model according to Lemma 5 and Lemma 6, it is obvious that the $\mathsf{ACC}$ also has completeness and knowledge soundness in the random oracle model.

*Proof Sketch.* For completeness, given that $\mathsf{NARK}.\mathcal{V}^{\mathsf{RO}}(\mathsf{vk}, \{\mathbb{x}^{(k)}\}_k, \pi) = 1$ and $\mathsf{ACC}.\mathcal{D}(\mathsf{adk}, \mathsf{acc}) = 1$, then $\{\mathsf{acc}^{(i)}.\mathbb{x}, \mathsf{acc}^{(i)}.\mathbb{w}\}_{i\in[1]} \in (\mathcal{R}_{\mathsf{acc}}^{\mathsf{PC}}(\mathsf{pp}))^2$. Since $\mathsf{ACC}$ is a trivial wrapper around the non-interactive reduction $\mathsf{NIR}_{\mathsf{fold}}$, completeness follows immediately from the completeness of $\mathsf{NIR}_{\mathsf{fold}}$. For the knowledge soundness, we prove that for an arbitrary polynomial-time adversary $\tilde{\mathcal{P}}$ against $\mathsf{ACC}$, an adversary $\tilde{\mathcal{P}}_{\mathsf{fold}}^{\mathsf{RO}}$ against $\mathsf{NIR}_{\mathsf{fold}}$ can be built as follows:

- compute $(\mathtt{i}, \{\mathbb{x}^{(k)}\}_k, \pi.\mathbb{x}, \mathsf{acc}^{(0)}.\mathbb{x}, \mathsf{acc}, \mathsf{pf}; \mathsf{tr}) \leftarrow \tilde{\mathcal{P}}^{\mathsf{RO}}(\mathsf{pp}, \mathsf{ai})$;

- assign $(\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w}) := \mathsf{acc}$ and $\pi_{\mathsf{fold}} = \mathsf{pf}$;
- compute $(\mathsf{pk}_{\mathsf{cast}}, \mathsf{vk}_{\mathsf{cast}}, \mathbb{i}') \leftarrow \mathsf{NIR}_{\mathsf{cast}}.\mathcal{I}^{\mathsf{RO}}(\mathsf{pp}, \mathbb{i})$;
- compute $\mathsf{acc}^{(1)}.\mathbb{x} \leftarrow \mathsf{NIR}_{\mathsf{cast}}.\mathcal{V}^{\mathsf{RO}}(\mathsf{vk}_{\mathsf{cast}}, \{\mathbb{x}^{(k)}\}_k, \pi.\mathbb{x})$;
- output $(\mathbb{i}', \{\mathsf{acc}^{(i)}.\mathbb{x}\}_{i\in[1]}, \pi_{\mathsf{fold}}, \mathsf{acc}.\mathbb{w}; \mathsf{tr})$.

By the knowledge soundness of the non-interactive reduction $\mathsf{NIR}_{\mathsf{fold}}$, there exists a corresponding extractor $\mathcal{E}_{\mathsf{fold}}$. Then we can construct an extractor for $\mathsf{ACC}$ as follows:

- assign $(\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w}) := \mathsf{acc}$ and $\pi_{\mathsf{fold}} := \mathsf{pf}$;
- compute $\{\mathsf{acc}^{(i)}.\mathbb{w}\}_{i\in[1]} \leftarrow \mathcal{E}_{\mathsf{fold}}(\mathsf{pp}, \mathbb{i}', \{\mathsf{acc}^{(i)}.\mathbb{x}\}_{i\in[1]}, \pi_{\mathsf{fold}}, \mathsf{acc}.\mathbb{w}, \mathsf{ai}, \mathsf{tr})$;
- assign $\pi.\mathbb{w} := \mathsf{acc}^{(1)}.\mathbb{w}$;
- output $(\pi.\mathbb{w}, \mathsf{acc}^{(0)}.\mathbb{w})$.

We claim that the error bound is tight, i.e., the two probabilities, $\mathsf{Pr}[E_1|E_2]$ and $\mathsf{Pr}[E_3|E_4]$, are negligibly close:

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

$$E_2 = \left\{ \begin{array}{r} \mathsf{RO} \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{ACC}.\mathcal{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \chi(1^\lambda) \\ (\mathbb{i}, \{\mathbb{x}^{(k)}\}_k, \pi.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}', \mathsf{pf}; \mathsf{tr}) \leftarrow \tilde{\mathcal{P}}^{\mathsf{RO}}(\mathsf{apk}, \mathsf{ai}) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{NARK}.\mathcal{I}(\mathsf{pp}, \mathcal{I}) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{adk}) \leftarrow \mathsf{ACC}.\mathcal{I}(\mathsf{pp}, \mathbb{i}) \\ (\pi.\mathbb{w}, \mathsf{acc}.\mathbb{w}) \leftarrow \mathcal{E}(\mathsf{pp}, \mathbb{i}, \{\mathbb{x}^{(k)}\}_k, \pi.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}', \mathsf{pf}; \mathsf{ai}, \mathsf{tr}) \end{array} \right\}.$$

$$E_3 = \left\{ \begin{array}{c} (\mathbb{i}'\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w}) \in \mathcal{R}^{\mathsf{PC}}_{\mathsf{acc}}(\mathsf{pp}) \\ \wedge \\ (\mathbb{i}'\{\mathsf{acc}^{(i)}.\mathbb{x}, \mathsf{acc}^{(i)}.\mathbb{w}\}_{i\in[1]}) \in (\mathcal{R}^{\mathsf{PC}}_{\mathsf{acc}}(\mathsf{pp}))^2 \end{array} \right\},$$

$$E_4 = \left\{ \begin{array}{r} \mathsf{RO} \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{ACC}.\mathcal{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \chi(1^\lambda) \\ (\mathbb{i}', \{\mathsf{acc}^{(i)}.\mathbb{x}\}_{i\in[1]}, \pi_{\mathsf{fold}}, \mathsf{acc}.\mathbb{w}; \mathsf{tr}) \leftarrow \mathsf{NIR}_{\mathsf{fold}}.\mathcal{P}^{\mathsf{RO}}(\mathsf{pp}, \mathsf{ai}) \\ (\mathsf{pk}_{\mathsf{fold}}, \mathsf{vk}_{\mathsf{fold}}, \mathbb{i}') \leftarrow \mathsf{NIR}_{\mathsf{fold}}.\mathcal{I}^{\mathsf{RO}}(\mathsf{pp}, \mathbb{i}') \\ \mathsf{acc}.\mathbb{x} \leftarrow \mathsf{NIR}_{\mathsf{fold}}.\mathcal{V}^{\mathsf{RO}}(\mathsf{vk}_{\mathsf{fold}}, \{\mathsf{acc}^{(i)}.\mathbb{x}\}_{i\in[1]}, \pi_{\mathsf{fold}}) \\ \{\mathsf{acc}^{(i)}.\mathbb{w}\}_{i\in[1]} \leftarrow \mathcal{E}_{\mathsf{fold}}(\mathsf{pp}, \mathbb{i}', \{\mathsf{acc}^{(i)}.\mathbb{x}\}_{i\in[1]}, \pi_{\mathsf{fold}}, \mathsf{acc}.\mathbb{w}, \mathsf{ai}; \mathsf{tr}) \end{array} \right\}.$$

### B.4 Construction for Multi-Instance IVC

We provide the special sound protocols for the necessary components in building a polynomial IOP for $\mathcal{R}_{\mathsf{plonk}}$, including protocols for checking the gate identity, wiring identity, and instance consistency, which can be further transformed into compressed versions as mentioned in Section 5.1. As a result, we can instantiate an accumulation scheme for these special-sound protocols via the techniques mentioned in Section 5, and further instantiate an IVC with the plonkish constraint systems.

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

**Fig. 11.** Special-Sound Protocol $\Pi_{\mathsf{gate}}$ for the gate identity

| |
|---|
| $\mathcal{P}$'s inputs: $\sigma, \tilde{w}(\boldsymbol{X})$ |
| $\mathcal{V}$'s inputs: $\sigma$ |
| 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. 12.** Special-Sound Protocol $\Pi_{\mathsf{wire}}$ 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 $\tilde{p}(\boldsymbol{x}) = \tilde{w}(\boldsymbol{0}^{\log \mu + \log n - \log m}, \boldsymbol{x})$ for all $\boldsymbol{x} \in \{0,1\}^{\log m}$. |

**Fig. 13.** Special-Sound Protocol $\Pi_{\mathsf{pi}}$ for the instance consistency

Next, we provide the oracle batching protocols based on codeword batching techniques [15] in Figure 14 and highlights it security property in Theorem 6, of which the proof can be referred to [15].

**Theorem 6.** *The reduction* $\mathsf{NIR}_{\mathsf{batch}}$ *based on linear codes has round-by-round knowledge soundness error* $\leq 2^{\lambda}$ *for every* $\delta \in (0,1)$*, if the field* $\mathbb{F}$ *and repetition parameters* $t, s$ *satisfy*

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

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

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

**Interaction phase.**

1 : $\mathcal{P}$ computes $\boldsymbol{u} := \sum_{k\in[1]} \tilde{eq}_k(r) \cdot \boldsymbol{u}_k$ and its oracle $[\![\tilde{u}]\!]$

2 : $\mathcal{P} \to \mathcal{V} : [\![\tilde{u}]\!]$

3 : $\mathcal{V} \to \mathcal{P} : \boldsymbol{\alpha}_{r+1}, \ldots, \boldsymbol{\alpha}_{r+s} \leftarrow\$ \mathbb{F}^{\log(1/\rho)+\log n}$. // out of domain samples

4 : $\mathcal{P}$ computes $\mu_{r+1}, \ldots, \mu_{r+s} \in \mathbb{F}$, where $\mu_{r+j} := \tilde{u}(\boldsymbol{\alpha})$

5 : $\mathcal{P} \to \mathcal{V} : \mu_{r+1}, \ldots, \mu_{r+s}$

6 : $\mathcal{V}$ samples $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_t \leftarrow\$ \{0,1\}^{\log(1/\rho)+\log n}$ and queries $[\![\tilde{u}_k(\boldsymbol{X})]\!]$ // shift queries

**Output phase.**

7 : Define $\mu_{r+s+j} := \sum_{k\in[1]} \tilde{eq}_k(r) \cdot \tilde{u}_k(\boldsymbol{b}_j)\ \forall j \in [t]$.

8 : $\mathcal{V}$ outputs the new instance-oracle pair as $(\mathbb{x} := \{\boldsymbol{\alpha}_i, \mu_i\}_{i\in r+s+t}, \mathbb{y} := [\![\tilde{u}]\!])$

**Fig. 14.** Oracle batching protocol $\mathsf{NIR}_{\mathsf{batch}}$ based on linear codes

# C Security Proofs

## C.1 Proof of Lemma 3

*Proof.* To conduct the proof, we follow the roadmap in [1,20]. The completeness holds trivially. For the RBR knowledge soundness, the protocol $\Pi_{\mathsf{Am}}$ has soundness error $\epsilon^{\mathsf{zc}}, \epsilon_1^{\mathsf{sc}}, \ldots, \epsilon_{\log \ell}^{\mathsf{sc}}, \epsilon^{\mathsf{agg}}$ with extraction time $O(\ell^2 \cdot n^2)$, where:

$$\epsilon^{\mathsf{zc}} = d\ell/|\mathbb{F}|, \epsilon_i^{\mathsf{sc}} = d/|\mathbb{F}|,\ \forall i \in [\log \ell], \epsilon^{\mathsf{agg}} = n/|\mathbb{F}|.$$

We define the state function and prove bounds on the RBR knowledge soundness error as follows:

- State function for empty transcript. Given inputs $\mathbb{i}, \mathbb{x} = \{\mathbb{x}^{(i)}\}_{i\in[\ell]}$, we set $\mathsf{State}(\mathbb{i}, \mathbb{x}, \emptyset) = 1$ if and only if $\mathbb{x}^{(i)} \in \mathcal{L}(\mathcal{R}_F)$ for all $i \in [\ell]$.

- Bounding $\epsilon^{\mathsf{zc}}$. At this stage, the partial transcript has the form $\mathsf{tr} = (\llbracket \tilde{\mathbb{w}}_{\cup} \rrbracket)$ and the verifier sends $\boldsymbol{r}_y$. We set the state function $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr} || \boldsymbol{r}_y) = 1$ if and only if the following hold:

  - $\sum_{\boldsymbol{y} \in B_{\log \ell}} G(\boldsymbol{y}) = 0$ as in Equation 8.

  The extractor outputs the witness $\boldsymbol{w}_{\cup} := \{\boldsymbol{w}^{(i)}\}_{i \in [\ell]}$ in time $O(\ell^2 \cdot n^2)$, where $\boldsymbol{w}^{(i)}[j] = \tilde{w}_{\cup}(\mathsf{Bits}(i), \mathsf{Bits}(j))$. For the knowledge error, we show that if the extracted witness is invalid, i.e., $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{w}_{\cup}, \mathsf{tr}) = 0$, then $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr} || \boldsymbol{r}_y) = 1$ holds with a negligible probability $d\ell/|\mathbb{F}|$. Concretely, if any $F(\boldsymbol{x}^{(i)}, \boldsymbol{w}^{(i)}) = 0$ does not holds, the sum equation $\sum_{\boldsymbol{y} \in B_{\log \ell}} G(\boldsymbol{y})$ as a multilinear extension, evaluates to zero at a randomly sampled point $\boldsymbol{r}_y$ with probability less than $d\ell/|\mathbb{F}|$, which is ensured by Schwartz-Zippel lemma.
- Bounding $\epsilon_i^{\mathsf{sc}}$. We discuss the two cases below. For the first round of sum-check, the partial transcript has the form $\mathsf{tr} = (\llbracket \tilde{w}_{\cup} \rrbracket, \boldsymbol{r}_y, G_1(\boldsymbol{Y}))$. We set the state function $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr} || \tau_1) = 1$ if and only if the following hold:

  - $G_1(\tau_1) = \sum_{x_2, \ldots, x_{\log \ell} \in B_{\log \ell - 1}} G(\tau_1, x_2, \ldots, x_{\log \ell})$.

  Since the error is always below $\epsilon^{\mathsf{zc}}$, we do not need the extractor to be able to extract, i.e., the extractor outputs $\perp$ given $\mathbb{x}, \mathsf{tr}$. To bound the error, if $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = 0$, i.e., $\sum_{\boldsymbol{y} \in B_{\log \ell}} G(\boldsymbol{y}) \neq 0$, then $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr} || \tau_1) = 1$ holds with a negligible probability $d/|\mathbb{F}|$. Note that the only way the prover can convince the verifier is by sending a univariate polynomial $G_1'(X)$ that does not equal the prescribed polynomial $G_1(X)$ and yet $G_1'(\tau_1) = G_1(\tau_1)$, of which the probability is bounded by $d/|\mathbb{F}|$. For $i$-th round of the sum-check protocol ($i > 1$), the partial transcript has the form $\mathsf{tr} = (\llbracket \tilde{w}_{\cup} \rrbracket, \boldsymbol{r}_y, G_1(\boldsymbol{Y}), \ldots, G_i(\boldsymbol{Y}), \tau_1, \ldots, \tau_{i-1})$. We set the state function $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr} || \tau_1) = 1$ if and only if the following hold:

  - $G_i(\tau_i) = \sum_{x_{i+1}, \ldots, x_{\log \ell} \in B_{\log \ell - i}} G(\tau_1, \ldots, \tau_i, x_{i+1}, \ldots, x_{\log \ell})$.

  The error is also bounded by $d/|\mathbb{F}|$ similar to the case $i = 1$.
- Bounding $\epsilon^{\mathsf{agg}}$. At this stage, the partial transcript has the form $\mathsf{tr} = (\llbracket \tilde{w}_{\cup} \rrbracket, \boldsymbol{r}_y, G_1(\boldsymbol{Y}), \ldots, G_{\log \ell}(\boldsymbol{Y}), \tau_1, \ldots, \tau_{\log \ell}, \llbracket \tilde{w} \rrbracket)$ and the verifier sends $\boldsymbol{r}_x$. We set the state function $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr} || \boldsymbol{r}_x) = 1$ if and only if the following hold:

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

  The extractor outputs the witness $\boldsymbol{w}'$ in time $O(n^2)$, where $\boldsymbol{w}'[j] = \tilde{w}(\mathsf{Bits}(j)), j \in [n]$ queried to $\llbracket \tilde{w} \rrbracket$. For the knowledge error, we show that if the extracted witness is invalid, i.e., $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{w}, \mathsf{tr}) = 0$, then $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr} || \boldsymbol{r}_x) = 1$ holds with a negligible probability $d\ell/|\mathbb{F}|$. Since $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = 0$, we have

$$\tilde{eq}(\boldsymbol{\tau}, \boldsymbol{r}_y) F(\boldsymbol{x}, \boldsymbol{w}) = G_{\log \ell}(\tau_{\log \ell}) \neq G'(\boldsymbol{\tau}) = \tilde{eq}(\boldsymbol{\tau}, \boldsymbol{r}_y) F(\boldsymbol{x}, \boldsymbol{w}'), \tag{42}$$

  where $\boldsymbol{w}$ is computed from the previously extracted witness. Since $F$ is defined as a sum of homogeneous algebraic maps, the inequality $\tilde{eq}(\boldsymbol{\tau}, \boldsymbol{r}_y) F(\boldsymbol{x}, \boldsymbol{w}) \neq \tilde{eq}(\boldsymbol{\tau}, \boldsymbol{r}_y) F(\boldsymbol{x}, \boldsymbol{w}')$ indicates that $\boldsymbol{w} \neq \boldsymbol{w}'$. Note that $\tilde{w}(\boldsymbol{X}) = \tilde{w}(\boldsymbol{\tau}, \boldsymbol{X})$ and $\tilde{w}'(\boldsymbol{X}) = \sum_{i \in [n]} \tilde{eq}_{i-1}(\boldsymbol{X}) \cdot \boldsymbol{w}[i]$, the probability that two different polynomials happen to be equivalent at a randomly sampled point $\boldsymbol{r}_x$ is at most $n/|\mathbb{F}|$.

□

### C.2 Proof of Lemma 6

*Proof.* To conduct the proof, we follow the roadmap in [1,20]. 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 $\epsilon_0$ 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 $\epsilon^{\mathsf{zc}}$. 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 $\mathbb{w} = (\{\boldsymbol{m}^{(k)}_{\cup,i}\}_{i\in[\mu],k\in[1]}, \{\boldsymbol{m}^{(k)}_{i}\}_{i\in[\mu],k\in[1]})$ vectors 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 to $[\![\tilde{m}_{\cup,i}]\!]$'s, and $\boldsymbol{m}^{(k)}_{i}[j] = \tilde{m}_i(\mathsf{Bits}(j)), j \in [n]$ queried to $[\![\tilde{m}_i]\!]$'s. 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 $\epsilon^{\mathsf{zc}}$. Concretely, we define $\{G^{(k)}(\boldsymbol{Z}_\gamma, Z_r)\}_{k\in[1]}$ as a multivariate polynomial below

$$\begin{aligned} G^{(k)}(\boldsymbol{Z}_\gamma, Z_r) = \tilde{eq}(Z_r, k) \cdot \big[&(F(\tilde{\boldsymbol{x}}(k), \{\boldsymbol{m}_i(k)\}_{i\in[\mu]}, \boldsymbol{r}_F(k)) - e(k)) \\ &+ \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{Z}_\gamma) \cdot (\tilde{m}_i(k, \boldsymbol{r}_x(k))) - v_i(k)) \\ &+ \sum_{i\in[\mu]} \mathsf{pow}_{\mu+i}(\boldsymbol{Z}_\gamma) \cdot (\tilde{m}_{\cup,i}(k, \boldsymbol{\tau}(k), \boldsymbol{r}_x(k)) - v_i(k))\big]. \end{aligned} \tag{43}$$

  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 $\epsilon^{\mathsf{zc}} \le (\mu+1)/|\mathbb{F}|$.
- Bounding $\epsilon^{\mathsf{sc}}$. 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:

$$\begin{aligned} G(\sigma) = \tilde{eq}(r_z, \sigma) \cdot \big[&(F(\tilde{\boldsymbol{x}}(\sigma), \{\boldsymbol{m}_i(\sigma)\}_{i\in[\mu]}, \boldsymbol{r}_F(\sigma)) - e(\sigma)) \\ &+ \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{\gamma}) \cdot (\tilde{m}_i(Z, \boldsymbol{r}_x(Z))) - v_i(Z)) \\ &+ \sum_{i\in[\mu]} \mathsf{pow}_{\mu+i}(\boldsymbol{\gamma}) \cdot (\tilde{m}_{\cup,i}(Z, \boldsymbol{\tau}(Z), \boldsymbol{r}_x(Z)) - v_i(Z))\big] \end{aligned} \tag{44}$$

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 $\epsilon^{\mathsf{sc}}$. Note that 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 $\epsilon^{\mathsf{eval}}$. At this stage, the partial transcript has the form $\mathsf{tr} = (\boldsymbol{\gamma}, r_z, G(Z), \sigma, G(\sigma))$. The prover sends the new evaluation claims as $\boldsymbol{\eta} := (\eta, \{\eta_i, \eta_{\cup,i}\}_{i\in[\mu]})$. 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 $\epsilon_{\mathsf{eval}}$. 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 $\mu/|\mathbb{F}|$:

$$G(\sigma) = \tilde{eq}(r_z, \sigma) \cdot [\eta' + \sum_{i\in[\mu]} \mathsf{pow}_i(\boldsymbol{\gamma}) \cdot \eta'_i + \mathsf{pow}_{i+\mu}(\boldsymbol{\gamma}) \cdot \eta'_{\cup,i}]. \tag{45}$$

- 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 $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}||\boldsymbol{c}) = 1$ if and only if the following hold:
  - $(\mathbb{x} = \{(\boldsymbol{y}_j, \boldsymbol{x}_j, v_j)\}_j, \mathbb{y} = [\![\tilde{m}_{\cup,i}]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}}\ \forall i \in [\mu]$
  - $(\mathbb{x} = \{(\boldsymbol{x}_t, v_t)\}_t, \mathbb{y} = [\![\tilde{m}_i]\!], \mathbb{w} = \bot) \in \mathcal{R}_{\mathsf{eval}}\ \forall i \in [\mu]$
  - $(\mathbb{x} = (\boldsymbol{x}, \boldsymbol{r}_F, e), \mathbb{y} = \bot, \mathbb{w} = \{\boldsymbol{m}_i\}_{i\in[\mu]}) \in \mathcal{R}_F$

  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., $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathbb{w}, \mathsf{tr}) = 0$, then $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}||\boldsymbol{c}) = 1$ holds with a negligible probability $\epsilon_{\mathsf{batch}}$. Since each protocol $\mathsf{IOR}_{\mathsf{batch}}$ has soundness error $\epsilon_0$, the overall probability is restricted by a union bound as $2\mu \cdot \epsilon_0$.

□