Zero-Knowledge Proof-Carrying Data from Accumulation Schemes

Tianyu Zheng[1], Shang Gao[1], and Xun Liu[1]

Department of Computing
The Hong Kong Polytechnic University, Hong Kong, China
{tian-yu.zheng, compxun.liu}@connect.polyu.hk
{shang-jason.gao}@polyu.edu.hk

Abstract. Proof-carrying data (PCD) is a powerful cryptographic primitive for ensuring computational integrity in distributed settings. State-of-the-art PCD constructions based on accumulation schemes achieve practical prover efficiency and support a wide range of applications. However, realizing zero-knowledge for accumulation-based PCD remains challenging, particularly for high-degree relations. Existing solutions often incur substantial overhead due to the need for zero-knowledge in both the underlying non-interactive arguments of knowledge (NARKs) and accumulation schemes. In this work, we present new theoretical and practical improvements for zero-knowledge PCD. First, we propose a novel construction that eliminates the need for zero-knowledge NARKs by separating the compliance predicate and accumulation verification, thereby reducing proof size and improving efficiency. Second, we design an efficient zero-knowledge accumulation scheme for special-sound protocols, introducing techniques such as masking vectors and zero-knowledge sum-check protocols to ensure privacy with minimal overhead. The theoretical analysis demonstrates that our construction achieves logarithmic proof size and verification time for d-degree NP relations, and outperforms existing solutions in both asymptotic and concrete complexity.

1 Introduction

Proof-Carrying Data (PCD) [CT10] is a foundational cryptographic primitive that empowers mutually distrustful parties to collaboratively perform distributed computations with strong integrity guarantees. By enabling each participating node to attach succinct proofs certifying the correctness of every intermediate state, PCD ensures that the validity of complex, potentially unbounded computations can be efficiently verified at any point. As a generalization of incrementally verifiable computation (IVC) [Val08], PCD has catalyzed advances in diverse applications, including secure enforcement of language semantics [CTV13], verifiable MapReduce computations [CTV15], consensus protocols [BCG21], scalable blockchains [BMRS20b,BMRS20a], zero-knowledge virtual machine [nex24], and verifiable inference [CCYY25].

Recursive vs. Accumulation. There are two main approaches to constructing PCD. The first, exemplified by recursive SNARKs [BCCT13,BSCTV17,COS20], requires the prover at each step to not only prove the computation of the compliance predicate, but also verify the validity of previous proofs. This demands embedding SNARK verification logic into the circuit, which is both theoretically and practically challenging due to the need for sublinear and circuit-friendly verification [CCDW20]. More recently, a practical alternative based on accumulation or folding schemes [BCMS20,BCL+21,KS24,ZGGX23,NDC+24,BMNW25,BCFW25] have attracted significant attention. Here, the prover performs the compliance predicate computation and defers verification of previous proofs using accumulation schemes, replacing costly SNARK verification with a much more efficient check on the accumulation proof. Consequently, the underlying non-interactive arguments of knowledge (NARKs) need not be succinct, resulting in improved prover efficiency.

Zero-knowledge PCD. As mentioned in previous works [COS20,BCMS20,BCL+21], achieving zero-knowledge is essential for PCD in privacy-sensitive applications such as private smart contracts, confidential transactions, and anonymous blocklisting [BBB+18,Dia21,ZGSX23,KKC+25]. In the zero-knowledge PCD setting, each node's output proof should be cryptographically shielded, ensuring that no sensitive information about the witness is leaked to the distrustful next node. This requirement is notably stronger than that of zero-knowledge IVC, which involves only a single prover and typically demands zero-knowledge only for the proof presented to the verifier. We illustrate this difference in Figure 1.

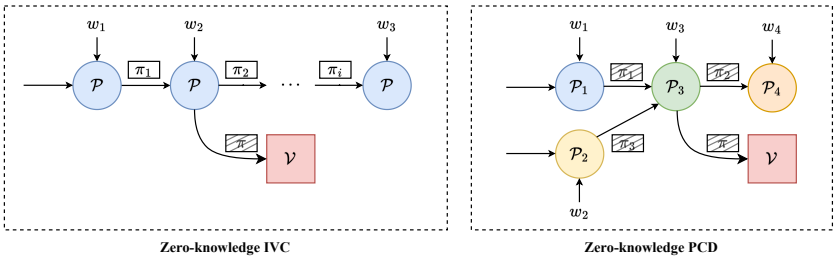


Fig. 1. Comparison of the workflow between zk-IVC and zk-PCD: zero-knowledge PCD requires every node's output to remain confidential with respect to its witness, whereas IVC only achieves confidentiality for the proof presented to the verifier.

Challenges. Despite various accumulation-based PCD solutions, only a few of them [BCMS20,BCL+21,BMNW25] achieve practical zero-knowledge for low-degree relations such as R1CS. Canonical accumulation-based PCD schemes require both an efficient zero-knowledge NARK and a corresponding zero-knowledge accumulation scheme [BCL+21]. In contrast, recursive SNARK-based PCD only

needs the SNARK to be zero-knowledge, for which many efficient solutions exist [BBB+18,GWC19,BSBHR19]. Intuitively, achieving zero-knowledge for PCD constructions based on accumulations is inherently more challenging. Practically, implementing zero-knowledge for both the NARK and accumulation schemes faces severe efficiency issues, especially for high-degree relations. This is because standard techniques like random masking introduce exponential overhead—specifically, $O(2^d)$ elements for a NARK proof and $O(m \cdot 2^d)$ for an accumulation proof with m input instances, both for d-degree relations in existing solutions [BCL+21,BC23,KS24]. Thus, designing efficient zero-knowledge PCD schemes for high-degree relations based on accumulation schemes remains a central challenge in research.

1.1 Our contributions

Our work makes both theoretical and practical contributions as follows:

- To address the efficiency issues of achieving zero-knowledge for NARKs, we propose a new zero-knowledge PCD construction that directly avoids this requirement in Section 4. Unlike previous approaches, our construction splits the recursive circuit into two parts: the first represents the compliance predicate over the witness, and the second represents accumulation verification without the witness. By instantiating the first circuit as a NARK instance-proof pair and then accumulating it with a zero-knowledge accumulation scheme, we observe that the NARK itself need not be zero-knowledge. Therefore, we eliminate the significant overhead of achieving zero-knowledge NARKs. We formalize this intuition and provide rigorous proofs: a zero-knowledge PCD can be constructed from a (non-zk) NARK and a zero-knowledge accumulation scheme for the NARK.
- To address the efficiency issues of zero-knowledge accumulation schemes, we present a concrete design for an eligible accumulation scheme for special-sound protocols [BC23] in Section 5. Since the zero-knowledge NARK requirement is removed, the accumulation scheme must provide a stronger zero-knowledge property for both the output accumulator and the accumulation proof. Building on previous solutions [BC23,EG23], we make two key modifications:
  - We introduce a masking vector for the final accumulator's witness. As the witness trivially satisfies the accumulator, this technique adds minimal overhead to the accumulation complexity: the prover and verifier only need to accumulate one more proof-instance pair.
  - We apply the zero-knowledge sum-check protocol from [CFS17] to provide privacy for the accumulation proof. To address the problem of growing accumulator length caused by the newly generated evaluation claims in the zero-knowledge sum-check, we introduce an additional sum-check protocol running in parallel to update and accumulate these claims.

  To the authors' knowledge, this is the first zero-knowledge accumulation scheme with logarithmic proof size and verification time for d-degree NP relations.

Table 1. Comparison of the additional proof size for zero-knowledge across different PCD constructions

SPS Protocols	[BCL+21]	Ours
Degree-d gate [GWC19]	$O(2^d)\mathbb{G}+ O(d\log m)\mathbb{F}$	$2\mathbb{G} + O(d\log m)\mathbb{F}$
Permutation [GWC19]	$4\mathbb{G}+ O(\log m)\mathbb{F}$	$2\mathbb{G} + O(\log m)\mathbb{F}$
Lookup [Hab22]	$(4\ell+2)\mathbb{G}+ O(\log m+\ell)\mathbb{F}$	$3\mathbb{G}+ O(\log(m+\ell))\mathbb{F}$
R1CS [Gro16]	$7\mathbb{G}+ O(\log m)\mathbb{F}$	$2\mathbb{G} + O(\log m)\mathbb{F}$
CCS [STW23]	$O(2^d)\mathbb{G}+ O(d\log m)\mathbb{F}$	$2\mathbb{G} + O(d\log m)\mathbb{F}$

Assume that m is the number of accumulated predicate instance-proof pairs, d is the maximum degree of the relations, and ℓ is the size of the query set for lookup arguments.

To demonstrate the efficiency of our construction, we provide in Table 1 a comparison of the additional PCD proof size incurred by zero-knowledge between the existing zero-knowledge PCD construction in [BCL+21] and our construction. For the existing construction, we follow the result of [BCL+21] and instantiate it with our accumulation scheme in Section 5. This is because no existing work provides zero-knowledge accumulation schemes for high-degree relations. For our scheme, we construct a PCD following the construction given in Section 4 and instantiate it with our accumulation scheme. We compare their performance i.e., the additional size of the PCD proof output by each node, for different compliance predicates including the custom gate and permutation checks in [GWC19], lookup arguments [Hab22], R1CS relations [Gro16], and CCS relations [STW23]. The designs of the special-sound protocols for the above predicates are taken from [BC23].

2 Technique Overview

We begin by explaining our first contribution in Section 2.1, a novel construction for zero-knowledge PCD that obviates the need for a zero-knowledge NARK. This construction represents a theoretical advance applicable to other IVC and PCD frameworks. In Section 2.2, we detail the concrete instantiation of a zero-knowledge accumulation scheme, using R1CS relations as an example.

For notations, we denote by $[n]$ the set $\{1,...,n\} \subseteq \mathbb{N}$. Denote $\mathbb{F}$ as a finite field and $\boldsymbol{a} \in \mathbb{F}^n$ as a vector, where $\boldsymbol{a}[i] \in \mathbb{F}$ is the i-th element. For a non-deterministic polynomial time (NP) relation R, it consists of a tuple of instance x, and witness w. Let $\tilde{f}(\boldsymbol{X})$ denote an n-variate polynomial with $\boldsymbol{X} \in \mathbb{F}^n$. Besides, we denote $eq(\boldsymbol{X}, \mathsf{Bits}(i))$ as an equality function that outputs 1 if $\boldsymbol{X} = \mathsf{Bits}(i)$ and 0 otherwise. A sum-check protocol [LFKN92] for $\tilde{f}(\boldsymbol{X})$ is a $\log(n)$-round interaction between the prover and verifier, the verifier is convinced that $s = \sum_{\boldsymbol{x}\in\{0,1\}^m} \tilde{f}(\boldsymbol{x})$ at the end. More details can be referred to Section 3 and Section 5.1.

2.1 New Construction for Zero-Knowledge PCD

Before presenting our construction, we briefly review how prior work derives PCD from NARKs equipped with accumulation schemes.

Recap of Existing Results. As discussed, zero-knowledge PCD is a versatile primitive with wide-ranging applications. Our starting point is the theorem of [BCL+21], which shows how to obtain PCD from any NARK that admits a split accumulation scheme. In broad terms, an accumulation scheme ACC for a NARK comprises three algorithms $(\mathsf{P}, \mathsf{V}, \mathsf{D})$: the prover produces an accumulation proof pf attesting that a new accumulator acc was computed correctly; the verifier checks the local correctness of this computation i.e., pf is valid for the claimed accumulation of acc; and the decider determines the satisfiability of an accumulator acc, i.e., whether $\mathsf{acc.w}$ is a satisfying witness to the instance $\mathsf{acc.x}$.

We now describe the PCD construction, illustrated on the left side of Figure 2. The prover at node v performs the following actions:

- **Accumulation.** Given an accumulator $\mathsf{acc}^{(u)}$ and a NARK instance-proof pair $(z^{(u)}, \pi^{(u)})$ from the previous node u, the node v invokes an accumulation proving algorithm $\mathsf{ACC.P}$. This generates a new accumulator $\mathsf{acc}^{(v)}$ and an accumulation proof pf, attesting to the correct transition from $((z^{(u)}, \pi^{(u)}), \mathsf{acc}^{(u)})$ to $\mathsf{acc}^{(v)}$.
- **Recursive Circuit Synthesis.** The prover then constructs a recursive circuit. This circuit takes as extra inputs $z^{(u)}$ and witness $w^{(v)}$, and performs two main checks: (1) it evaluates the PCD predicate $\varphi(z^{(u)}, w^{(v)}, z^{(v)})$ on the claimed inputs, and (2) it verifies the accumulation proof pf by running the logic of the accumulation verifier $\mathsf{ACC.V}$. A crucial feature of a *split* accumulation scheme is that $\mathsf{ACC}.V$ only checks the public instance parts of the accumulated proof, e.g., $\pi^{(u)}.\mathsf{x}$, and of the accumulator, e.g., $\mathsf{acc}^{(u)}.\mathsf{x}$, without their large witness parts.
- **Proof Generation.** Finally, the prover uses the NARK to generate a new proof $\pi^{(v)}$ for the satisfiability of this recursive circuit. This new proof $\pi^{(v)}$ and the new accumulator $\mathsf{acc}^{(v)}$ are then passed to the next node in the PCD computation.

The final PCD proof consists of the artifacts output in the last step, including the final accumulation proof and the final NARK proof. The PCD verifier checks them to ensure the validity of the entire PCD computation. Note that the above process considers only inputs from the previous node u for simplicity; we extend it to the multi-input case in the formal definition later.

Challenges. Instantiating the above PCD construction [BCL+21] suffers from an efficiency problem: achieving zero-knowledge for PCD requires *both* the underlying NARK and the accumulation scheme to be zero-knowledge. The primary bottleneck lies in the former one. Specifically, the NARK proof generated from the recursive circuit is *non-succinct*. Hiding this witness within the NARK proof

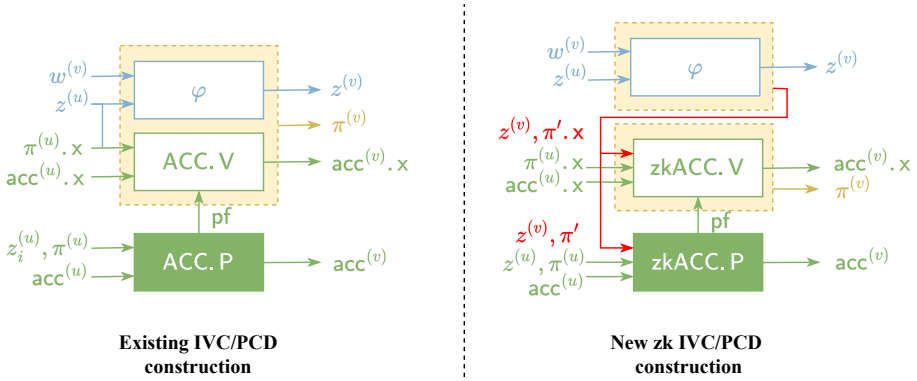


Fig. 2. Comparison between different PCD constructions.

requires applying zero-knowledge mechanisms, e.g., adding a masking string of the same length to the witness. For many high-degree argument systems, such as Plonkish [GWC19] and Customizable Constraint Systems (CCS) [STW23], transforming the relation to be zero-knowledge incurs substantial overhead due to the additional cross terms generated. For instance, adding zero-knowledge to a degree-d relation can introduce $O(2^d)$ new variables and constraints, leading to an exponential blowup in the size of the recursive circuit, which significantly limits the practicality of the PCD. Given that there are no known methods to efficiently enforce zero-knowledge for non-succinct argument systems with high degree, our work pursues an alternative path: we propose a new construction that eliminates the need for a zero-knowledge NARK.

New Zero-Knowledge Construction. To illustrate our idea, we begin by formalizing the recursive circuit used in prior PCD constructions (omitting some parameters for conciseness). Typically, the recursive circuit encodes the computation trace (predicate) for a single PCD step, and meanwhile, checks the correctness of previous accumulations. Now, assume that node v receives inputs from m incoming nodes, where m is the arity. It then synthesizes a circuit with public inputs including the predecessor outputs $[z^{(u_i)}]_{i=1}^m$, their corresponding NARK proof instances $[\pi^{(u_i)}.\mathsf{x}]_{i=1}^m$, their accumulator instances $[\mathsf{acc}^{(u_i)}.\mathsf{x}]_{i=1}^m$, and the new accumulator instance $\mathsf{acc}^{(v)}.\mathsf{x}$ with proof pf. The circuit also takes a private witness $w^{(v)}$ as an input. As shown in R below, the circuit performs two main checks:

- First, it checks the step computation with the PCD compliance predicate φ.
- Second, it checks that the new accumulator $\mathsf{acc}^{(v)}$ was correctly derived from the previous ones, i.e., running the accumulation verifier algorithm $\mathsf{ACC.V}$.

$R(z^{(v)}, \mathsf{acc}^{(v)}.\mathsf{x}; w^{(v)}, [z^{(u_i)}, \pi^{(u_i)}.\mathsf{x}, \mathsf{acc}^{(u_i)}.\mathsf{x}]_{i=1}^{m}, \mathsf{pf})$:

1: Check that the compliance predicate $\varphi(z^{(v)}, w^{(v)}, z^{(u_1)}, \ldots, z^{(u_m)})$ accepts.

2: Check that the accumulation verifier accepts if $\exists z^{(u_i)} \neq \perp$:

$$\mathsf{ACC.V}([z^{(u_i)}, \pi^{(u_i)}.\mathsf{x}, \mathsf{acc}^{(u_i)}.\mathsf{x}]_{i=1}^{m}, \mathsf{acc}^{(v)}.\mathsf{x}, \mathsf{pf}) = 1,$$

3: If the above checks hold, output 1; otherwise, output 0.

Our new construction is based on a key observation: the witness $w^{(v)}$ is exclusively used for the predicate compliance check on φ, while the accumulation verification ($\mathsf{ACC.V}$) operates only on public data. This implies that zero-knowledge is only required for the predicate-check part (step 2 in Figure 2) of the computation. This insight allows us to avoid a monolithic, fully zero-knowledge recursive circuit. Instead, we partition the circuit R into two specialized sub-circuits, R_0 and R_1, corresponding to steps 1 and 2, respectively, as shown in Figure 2 (right). In essence, we replace the expensive requirement of a ZK-NARK over the entire recursive logic with a much cheaper ZK-NARK over only the small, predicate-specific sub-circuit R_0. Assuming the accumulation scheme is zero-knowledge, the final NARK proof $\pi^{(v)}$ generated for circuit R_1 reveals nothing about the original witness $w^{(v)}$, since $w^{(v)}$ is only used within the confines of the ZK-NARK proof π'_v. Thus, the entire PCD construction achieves zero-knowledge.

The performance benefit of our approach is significant. Compared to prior works [BCMS20,BCL+21], the primary overhead is accumulating one additional proof (π'_v) in each step. For typical accumulation schemes, this cost is negligible, especially when compared to the exponential savings from avoiding a ZK-NARK over the entire recursive circuit. Furthermore, our technique is broadly applicable. It can be adapted to other recursive proof systems, such as IVC schemes built on folding, like Nova [KST22] and Halo [BDFG21]. The main prerequisite is an accumulation or folding scheme that can efficiently combine multiple instances. Theoretically, our work decouples the requirement of a ZK-NARK from the construction of ZK-PCD. This provides a new pathway to achieving zero-knowledge in incrementally verifiable computation, as stated below. We formalize this insight as Theorem 1 in Section 4.

2.2 Zero-Knowledge Accumulation Schemes

Our second contribution is a concrete and practically efficient instantiation of a zero-knowledge accumulation scheme for high-degree relations. We begin by revisiting the accumulation framework of ProtoGalaxy [EG23], and then present an efficient approach that adapts it to zero-knowledge settings.

Recap of Protogalaxy. For simplicity, we review an existing accumulation scheme for R1CS as an example, and extend it to high-degree relations in Section 5 later. Let Commit denote a homomorphic commitment scheme. Given two committed R1CS instance-witness pairs $[(\boldsymbol{x}_i, C_i; \boldsymbol{w}_i)]_{i=0}^{1}$ (also interpreted

as instance-proof pairs in an R1CS NARK system), each pair satisfies the following constraints:

$$\mathsf{R}^{\mathsf{cm}}_{\mathsf{R1CS}} := \left\{ (\boldsymbol{x}, C; \boldsymbol{w}) : C = \mathsf{Commit}(\boldsymbol{w}) \wedge A\boldsymbol{z} \circ B\boldsymbol{z} = C\boldsymbol{z} \right\}$$

where $\boldsymbol{x} \in \mathbb{F}^s$, $\boldsymbol{w} \in \mathbb{F}^t$, and $\boldsymbol{z} := (1, \boldsymbol{x}, \boldsymbol{w})$, with $A, B, C \in \mathbb{F}^{n \times m}$ and $m = 1+s+t$. For notational consistency, we define $\mathsf{V}_{\mathsf{R1CS}}(\boldsymbol{z}) := A\boldsymbol{z} \circ B\boldsymbol{z} = C\boldsymbol{z} = \boldsymbol{0}^n$ as an algebraic map of maximum degree 2. We now present an informal description of the accumulation scheme for $[(\boldsymbol{x}_i, C_i; \boldsymbol{w}_i)]_{i=0}^1$ according to [BC23,EG23]:

- The prover first computes a vector of polynomials:

$$\boldsymbol{F}(X) = A\boldsymbol{z}(X) \circ B\boldsymbol{z}(X) - C\boldsymbol{z}(X) \in (\mathbb{F}^{\leq 2}[X])^n, \tag{1}$$

  where $\boldsymbol{z}(X) = \widetilde{eq}_0(X) \cdot \boldsymbol{z}_0 + \widetilde{eq}_1(X) \cdot \boldsymbol{z}_1$ and $\widetilde{eq}_b(X)$ is a multilinear extension on equality function over $b \in \{0,1\}$, which maps X to 1 if $X = b$ and 0 otherwise.
- The accumulation verifier samples a random $\alpha \leftarrow\!\!\$\ \mathbb{F}$;
- The prover and verifier engage in n parallel sum-check protocols for the following n statements:

$$\sum_{b \in \{0,1\}} eq(\alpha, b) \cdot \boldsymbol{F}(b) = \boldsymbol{0}^n;$$

- The sum-check protocols yield the proof tr and evaluation claims $F(\beta) = \boldsymbol{e}$.

Finally, the prover and verifier agree on a new "relaxed" committed R1CS instance-witness pair $(\boldsymbol{x}, C, E; \boldsymbol{w})$, where $E := \mathsf{Commit}(\boldsymbol{e})$. And the prover sends the sum-check transcript tr as an accumulation proof pf. The accumulated elements are defined as follows:

$$\begin{aligned}
\boldsymbol{x} &:= \widetilde{eq}_0(\beta) \cdot \boldsymbol{x}_0 + \widetilde{eq}_1(\beta) \cdot \boldsymbol{x}_1, \\
C &:= \widetilde{eq}_0(\beta) \cdot C_0 + \widetilde{eq}_1(\beta) \cdot C_1, \\
\boldsymbol{w} &:= \widetilde{eq}_0(\beta) \cdot \boldsymbol{w}_0 + \widetilde{eq}_1(\beta) \cdot \boldsymbol{w}_1.
\end{aligned}$$

The key insight relies on the properties of the equality polynomials $\widetilde{eq}_i(X)$, namely that for all $i \neq j \in \{0,1\}$: (1) $\widetilde{eq}_i(b)^d = \widetilde{eq}_i(b)$ and (2) $\widetilde{eq}_i(b) \cdot \widetilde{eq}_j(b) = 0$ for all $b \in \{0,1\}$ (formally defined in Corollary 1). Therefore, for all $b \in \{0,1\}$, we have:

$$\boldsymbol{V}(b) = \boldsymbol{V}(\widetilde{eq}_0(b) \cdot \boldsymbol{z}_0 + \widetilde{eq}_1(b) \cdot \boldsymbol{z}_1) = \widetilde{eq}_0(b) \cdot \boldsymbol{V}(\boldsymbol{z}_0) + \widetilde{eq}_1(b) \cdot \boldsymbol{V}(\boldsymbol{z}_1),$$

which justifies the sum-check statement in Equation 1.

Adding Zero-Knowledge. We now consider how to incorporate zero-knowledge into the accumulation scheme described above. If we only require the randomness of the final output accumulator acc, the prover could simply embed an additional dummy R1CS pair into the accumulation prover's inputs, of which

the witness is randomly sampled. Specifically, the prover would accumulate three R1CS instance-witness pairs $[(\boldsymbol{x}_i, C_i; \boldsymbol{w}_i)]_{i=0}^2$, where $\boldsymbol{x}_2$ is a trivially satisfiable instance vector corresponding to a random witness $\boldsymbol{w}_2 \leftarrow\!\!\$\ \mathbb{F}^t$. However, this approach is insufficient, as the accumulation proof pf itselt may still leak information about the witness.

A practical solution is to employ the zero-knowledge sum-check protocol proposed in [CFS17], which achieves zero-knowledge by masking the coefficients of the target n-variate polynomial $F(\boldsymbol{X}) \in \mathbb{F}^{(\leq d)}[\boldsymbol{X}]$ with a random polynomial $G(\boldsymbol{X})$ of the same variables and individual degrees as $F(\boldsymbol{X})$. Furthermore, as discussed in Section 5.1, Xie et al. optimize this approach by reducing the size of $G(\boldsymbol{X})$ from $O(n \cdot d)$ to $O(d)$ [XZZ+19].

Nevertheless, introducing the zero-knowledge sum-check protocol in ACC presents a new challenge: the sum-check prover outputs an additional evaluation claim $\boldsymbol{G}(\beta) = \boldsymbol{v}$ when proving the following statement:

$$\sum_{b \in \{0,1\}} eq(\alpha, b) \cdot \boldsymbol{F}(b) + \gamma \cdot \boldsymbol{G}(b) = \gamma \cdot \boldsymbol{r}$$

where γ is a challenge and $\boldsymbol{r} := \sum_{b \in \{0,1\}} \boldsymbol{G}(b)$. Unlike other claims for $\boldsymbol{F}(\beta)$, the prover cannot simply defer the checking for $\boldsymbol{G}(\beta)$; otherwise, the length of the accumulator would continue to grow with the number of accumulation steps (i.e., with the number of recursive steps in PCD).

To address this issue, we adopt a technique from [KS24], which updates the evaluation point at each new accumulation step. In essence, whenever the prover generates a new evaluation claim $\boldsymbol{G}'(\beta') = \boldsymbol{v}'$, they can also update the point β in the previous claim $\boldsymbol{G}(\beta) = \boldsymbol{v}$ to β' by executing another round of the sum-check protocol (this time without zero-knowledge). As a result, the two claims $\boldsymbol{G}'(\beta')$ and $\boldsymbol{G}(\beta')$ can be linearly combined without incurring additional overhead.

In summary, we obtain an accumulation scheme between a committed R1CS pair $(\boldsymbol{x}_0, C_0; \boldsymbol{w}_0)$ and an accumulator

$$\mathsf{acc} := (\mathsf{acc.x} = (\boldsymbol{x}_1, C_1, E, \boldsymbol{v}),\ \mathsf{acc.w} = (\boldsymbol{w}_1, \boldsymbol{G}(X)))$$

where the size of the accumulator is independent of the number of instances being accumulated. In Section 5, we present a more general accumulation scheme for NP relations, which possesses the following features:

- Utilizes special-sound protocols as defined in prior works [BC23].
- Supports accumulation for multiple predicate pairs and *multiple* accumulators, a capability not achieved in Protostar [BC23].
- Achieves zero-knowledge (not mentioned in Protogalaxy [EG23]) with practical prover efficiency, scaling linearly with the number of accumulation inputs.

3 Preliminary

3.1 Notations

In this paper, we use λ to denote the security parameter. Accordingly, $\mathrm{negl}(\lambda)$ denotes an unspecified function that is negligible in λ. We denote by $[n]$ the set

$\{1, ..., n\} \subseteq \mathbb{N}$. Let $\mathbb{F}$ denote a finite field, e.g., $\mathbb{F}_p$ is a prime field for a large prime p. The vector is denoted as $\boldsymbol{a} \in \mathbb{F}^n$ with elements $a_1, ..., a_n \in \mathbb{F}$. $\boldsymbol{a}[i]$ is also used to denote the i-th element of $\boldsymbol{a}$ when the element is not specified with a concrete value. To represent a set, we use $\{a_i\}_{i=1}^n$ as a shorthand for $\{a_1, ..., a_n\}$. For a finite set S, let $x \leftarrow\!\$\ S$ denote sampling x from S uniformly at random. We use "PPT algorithms" to refer to "Probabilistic Polynomial Time" algorithms. For a non-deterministic polynomial time (NP) *indexed relation* R parameterized over public parameters pp (e.g., including the field $\mathbb{F}$), it consists of a triple of index i, instance x, and witness w (the secret inputs).

3.2 NARK

Specifically, we define a non-interactive argument of knowledge NARK for an indexed relation R as a tuple of algorithms $(\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$ below, each with access to the random oracle RO:

- $\mathsf{G}^{\mathsf{RO}}(1^\lambda) \to \mathsf{pp}$ the generator takes as inputs the security parameter λ in unary, and outputs the public parameter pp.
- $\mathsf{I}^{\mathsf{RO}}(\mathsf{pp}, \mathsf{i}) \to (\mathsf{ipk}, \mathsf{ivk})$ the indexer takes as inputs pp and an index i under relation R, and outputs a long index proving key ipk and a short index verification key ivk.
- $\mathsf{P}^{\mathsf{RO}}(\mathsf{ipk}, \mathsf{x}, \mathsf{w}) \to \pi$ the prover takes as inputs the ipk, and a tuple (x, w) under relation R, and outputs a NARK proof π.
- $\mathsf{V}^{\mathsf{RO}}(\mathsf{ivk}, \mathsf{x}, \pi) \to b$ the verifier takes as inputs the ivk, and a tuple (x, π), and outputs a bit b indicating whether π is valid.

NARK is secure in the random oracle model if the completeness and knowledge soundness hold as defined in Appendix C.3.

3.3 Accumulation Schemes

We adopt the definition of the accumulation scheme for NARK given in [BCL+21]. Let $\Phi : \{0,1\}^* \to \{0,1\}$ be a (relation) predicate and $\mathcal{H}$ be a randomized algorithm that can be accessed as an oracle and, given Φ, outputs predicate parameters pp_Φ. A (split) *accumulation scheme* for $(\Phi, \mathcal{H})$ is a tuple of algorithms $\mathsf{AS} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V}, \mathsf{D})$, where P and V have access to the same random oracle RO. Typically, we denote an accumulator acc with the same form as a predicate input, consisting of the instance part $\mathsf{acc.x}$ and the witness part $\mathsf{acc.w}$, and a decider algorithm D reads the accumulator in its entirety and checks its validity. Generally, these algorithms are expected to satisfy the following security properties:

Definition 1 (Completeness). ACC *is complete if for every (unbounded) adversary* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{c} \forall j \in [m], \mathsf{ACC.D}(\mathsf{adk}, \mathsf{acc}_j) = 1 \\ \forall i \in [n], \Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{qx}_i, \mathsf{qw}_i) = 1 \\ \Downarrow \\ \mathsf{acc.V}^{\mathsf{RO}}\left(\begin{array}{c} \mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, \\ [\mathsf{acc}_j.\mathsf{x}]_{j=1}^m, \\ \mathsf{acc.x}, \mathsf{pf} \end{array}\right) = 1 \\ \mathsf{acc.D}(\mathsf{adk}, \mathsf{acc}) = 1 \end{array} \middle| \begin{array}{r} \mathsf{RO} \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{ACC.G}(1^\lambda) \\ \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda) \\ \left(\begin{array}{c} \mathsf{i}_\Phi, [\mathsf{acc}_j]_{j=1}^m, \\ [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n \end{array}\right) \leftarrow \mathcal{A}^{\mathsf{RO}}(\mathsf{pp}, \mathsf{pp}_\Phi) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{adk}) \leftarrow \mathsf{ACC.I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \\ (\mathsf{acc}, \mathsf{pf}) \leftarrow \mathsf{ACC.P}^{\mathsf{RO}}\left(\begin{array}{c} \mathsf{apk}, [\mathsf{acc}_j]_{j=1}^m, \\ [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n \end{array}\right) \end{array}\right] = 1$$

Note that for $m = n = 0$*, the precondition on the left-hand side holds vacuously, and this is required for the completeness condition to be non-trivial.*

Definition 2 (Knowledge Soundness). ACC *is knowledge sound with respect to auxiliary input distribution* Ψ *if for every (non-uniform) expected polynomial-time adversary* $\tilde{\mathsf{P}}$*, there exists an expected polynomial-time extractor* $\mathcal{E}$ *such that*

$$\Pr\left[\begin{array}{c} \mathsf{ACC.V}^{\mathsf{RO}}\left(\begin{array}{c} \mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, \\ [\mathsf{acc}_j.\mathsf{x}]_{j=1}^m, \\ \mathsf{acc.x}, \mathsf{pf} \end{array}\right) = 1 \\ \wedge \mathsf{ACC.D}(\mathsf{adk}, \mathsf{acc}) = 1 \\ \Downarrow \\ \forall i \in [n], \\ \Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{qx}_i, \mathsf{qw}_i) = 1 \\ \forall j \in [m], \\ \mathsf{ACC.D}(\mathsf{adk}, (\mathsf{acc}_j.\mathsf{x}, \mathsf{acc}_j.\mathsf{w})) = 1 \end{array} \middle| \begin{array}{r} \mathsf{RO} \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{ACC.G}(1^\lambda) \\ \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda) \\ \mathsf{ai} \leftarrow \Psi(1^\lambda) \\ \left(\begin{array}{c} \mathsf{i}_\Phi, [\mathsf{qx}_i]_{i=1}^m, \\ [\mathsf{acc}_j.\mathsf{x}]_{j=1}^m, \\ \mathsf{acc}, \mathsf{pf}; r \end{array}\right) \leftarrow \tilde{\mathsf{P}}^{\mathsf{RO}}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}) \\ \left(\begin{array}{c} [\mathsf{qw}_i]_{i=1}^n, \\ [\mathsf{acc}_j.\mathsf{w}]_{j=1}^m \end{array}\right) \leftarrow \mathcal{E}^{\tilde{\mathsf{P}}, \mathsf{RO}}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}, r) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{adk}) \leftarrow \mathsf{ACC.I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \end{array}\right] \approx 1$$

3.4 Proof-Carrying Data

According to [BCL+21], a proof-carrying data scheme consists of a tuple of algorithms $\mathsf{PCD} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$. We start by defining some necessary terminology and then state the security definitions.

Definition 3. *A* transcript T *is a directed acyclic graph where each vertex* $u \in V(\mathsf{T})$ *is labeled by local data* $z_{\mathsf{loc}}^{(u)}$ *and each edge* $e \in E(\mathsf{T})$ *is labeled by a message* $z^{(e)} \neq \bot$*. The* output *of a transcript* T*, denoted by* $\circ(\mathsf{T})$*, is a message* $z^{(e)}$ *where* $e = (u, v)$ *is the lexicographically first edge such that* v *is a sink.*

Definition 4. *A vertex* $u \in V(\mathsf{T})$ *is* φ-compliant *for* $\varphi \in \mathsf{F}$ *if for all outgoing edges* $e = (u, v) \in E(\mathsf{T})$*:*

- *(base case) if u has no incoming edges, $\varphi(z^{(e)}, z_{\mathsf{loc}}^{(u)}, \bot, ..., \bot)$ accepts;*
- *(recursive case) if u has incoming edges $e_1, ..., e_m$, $\varphi(z^{(e)}, z_{\mathsf{loc}}^{(u)}, z^{(e_1)}, ..., z^{(e_m)})$ accepts.*

T *is φ-compliant if each of its vertices is φ-compliant.*

Next, we define the security properties of PCD. Note that the completeness and zero-knowledge are defined for each possible vertex in a transcript. Hence, for simplicity, we omit superscripts indicating the vertex and edge in $(z^{(e)}, z_{\mathsf{loc}}^{(u)}, z^{(e_1)}, ..., z^{(e_m)})$.

Definition 5 (Perfect Completeness). PCD *has perfect completeness if for every adversary $\mathcal{A}$ the following holds:*

$$\Pr\left[\begin{array}{c|r} \left(\begin{array}{c} \varphi \in \mathsf{F} \\ \wedge \varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m) = 1 \\ \wedge \forall i, z_i = \bot \vee \mathsf{PCD.V}(\mathsf{ivk}, z_i, \pi_i) = 1 \\ \Downarrow \\ \mathsf{PCD.V}(\mathsf{ivk}, z, \pi) = 1 \end{array}\right) & \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{PCD.G}(1^\lambda); \\ (\varphi, z, z_{\mathsf{loc}}, [z_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathsf{pp}); \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathsf{PCD.I}(\mathsf{pp}, \varphi); \\ \mathit{\Pi} \leftarrow \mathsf{PCD.P}(\mathsf{ipk}, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \end{array} \end{array}\right] = 1$$

Definition 6 (Knowledge Soundness). PCD *has knowledge soundness (w.r.t. an auxiliary input distribution $\mathcal{D}$) if for every expected polynomial-time adversary P^*, there exists an expected polynomial-time extractor $\mathsf{Ext}^{\mathsf{P}^*}$ such that for every set Z:*

$$\Pr\left[\begin{array}{c|r} \begin{array}{c} \varphi \in \mathsf{F} \\ \wedge(\mathsf{pp}, \mathsf{ai}, \varphi, \circ(\mathsf{T}), \mathsf{ao}) \in Z \\ \wedge \mathsf{T} \textit{ is } \varphi\textit{-compliant} \end{array} & \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{PCD.G}(1^\lambda); \\ \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}); \\ (\varphi, \mathsf{T}, \mathsf{ao}) \leftarrow \mathsf{Ext}^{\mathsf{P}^*}(\mathsf{pp}, \mathsf{ai}) \end{array} \end{array}\right],$$

$$\geq \Pr\left[\begin{array}{c|r} \begin{array}{c} \varphi \in \mathsf{F} \\ \wedge(\mathsf{pp}, \mathsf{ai}, \varphi, \circ, \mathsf{ao}) \in Z \\ \wedge \mathsf{PCD.V}(\mathsf{ivk}, \circ, \pi) = 1 \end{array} & \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{PCD.G}(1^\lambda); \\ \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}); \\ (\varphi, \circ, \pi, \mathsf{ao}) \leftarrow \mathsf{P}^*(\mathsf{pp}, \mathsf{ai}); \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathsf{PCD.I}(\mathsf{pp}, \varphi, \mathsf{F}) \end{array} \end{array}\right].$$

where ai *and* 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 [BCCT13]. As a result, we can prove post-quantum security for our PCD as mentioned in Theorem 1.*

Definition 7 (Zero-Knowledge). PCD *has (statistical) zero-knowledge if there exists a probabilistic polynomial-time simulator* Sim *such that for every polynomial-size honest adversary $\mathcal{A}$, the distributions below are computationally indistin-*

guishable:

$$\left\{ (\mathsf{pp}, \varphi, z, \pi) \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{PCD.G}(1^\lambda); \\ (\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathsf{pp}); \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathsf{PCD.I}(\mathsf{pp}, \varphi, \boldsymbol{F}); \\ \pi \leftarrow \mathsf{PCD.P}(\mathsf{ipk}, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \end{array} \right\}$$

and

$$\left\{ (\mathsf{pp}, \varphi, z, \pi) \;\middle|\; \begin{array}{r} (\mathsf{pp}, \tau) \leftarrow \mathsf{Sim}(1^\lambda); \\ (\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathsf{pp}); \\ \pi \leftarrow \mathsf{Sim}(\tau, \varphi, z) \end{array} \right\}.$$

An adversary is honest if its output satisfies the antecedent of the completeness condition with probability 1, namely: $\varphi \in \mathsf{F}, \varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m) = 1$*, and either* $\forall i, z_i = \perp$ *or* $\forall i, \mathsf{PCD.V}(\mathsf{ivk}, z_i, \pi_i) = 1$*.*

4 New Construction for Zero-Knowledge PCD

We first formalize our main theorem for constructing zero-knowledge PCD in Section 4.1. Next, in Section 4.2, we propose the full construction of zero-knowledge PCD from zero-knowledge accumulation schemes, which proves the main theorem. Finally, we present the security analysis of the proposed PCD construction in Section 4.3.

4.1 Main Theorem

To formalize the theorem, we first present the necessary definitions and describe the syntax of zero-knowledge accumulation schemes.

Definition 8 (Zero-Knowledge Accumulation Schemes). *Let* $\mathsf{NARK} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$ *be a non-interactive argument of knowledge, define* $\Phi : \{0,1\}^* \to \{0,1\}$ *as a predicate for* $\mathsf{NARK.V}$*, and let* $\mathcal{H} := \mathsf{NARK.G}$ *be a randomized oracle algorithm that outputs predicate parameters* pp_Φ*, where* φ_{NARK} *is defined below:*

$\varphi_{\mathsf{NARK}}(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{qx} = (\mathsf{x}, \pi.\mathsf{x}), \mathsf{qw} = \pi.\mathsf{w})$*:*

1 : $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathsf{NARK.I}(\mathsf{pp}_\Phi, \mathsf{i}_\Phi)$.
2 : *Output* $\mathsf{NARK.V}(\mathsf{ivk}, \mathsf{x}, (\pi.\mathsf{x}, \pi.\mathsf{w}))$.

An accumulation scheme for $(\Phi, \mathcal{H})$ *in the random oracle model consists of a tuple of algorithms as* $\mathsf{ACC} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V}, \mathsf{D})$*, where* G *is the same as* NARK*,* P, V *have access to the same random oracle* RO*. We describe their syntaxes as follows:*

- $\mathsf{ACC.G}(1^\lambda) \to \mathsf{pp}$*: on inputs a security parameter* λ *in unary, the generator* $\mathsf{ACC.G}(1^\lambda)$ *samples and outputs public parameters* pp*.*

- $\mathsf{ACC.I}^{\mathsf{RO}}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \to (\mathsf{apk}, \mathsf{avk}, \mathsf{adk})$*: on inputs public parameters* pp*, predicate parameters* pp_Φ *generated by* $\mathcal{H}$*, and a predicate index* i_Φ*, the indexer* $\mathsf{ACC.I}$ *deterministically computes and outputs a triple* $(\mathsf{apk}, \mathsf{avk}, \mathsf{adk})$ *consisting of an accumulator proving key* apk*, an accumulator verification key* avk*, and a decision key* adk*.*
- $\mathsf{ACC.P}^{\mathsf{RO}}(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m) \to (\mathsf{acc}, \mathsf{pf})$*: on inputs the key* apk*, predicate inputs* $[(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n$*, and old accumulators* $[\mathsf{acc}_j]_{j=1}^m = [(\mathsf{acc}_j.\mathsf{x}, \mathsf{acc}_j.\mathsf{w})]_{j=1}^m$*, the prover* $\mathsf{ACC.P}$ *outputs a new accumulator* $\mathsf{acc} = (\mathsf{acc.x}, \mathsf{acc.w})$ *and a proof* pf *for the accumulation verifier.*
- $\mathsf{ACC.V}^{\mathsf{RO}}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, [\mathsf{acc}_j.\mathsf{x}]_{j=1}^m, \mathsf{acc.x}, \mathsf{pf}) \to b$*: on inputs the key* avk*, predicate input instances* $[\mathsf{qx}_i]_{i=1}^n$*, old accumulator instance* $[\mathsf{acc}_j.\mathsf{x}]_{j=1}^m$*, a new accumulator instance* $\mathsf{acc.x}$*, and a proof* pf*, the verifier* $\mathsf{ACC.V}$ *outputs a bit indicating whether* $\mathsf{acc.x}$ *correctly accumulated* $[\mathsf{qx}_i]_{i=1}^n$ *and* $[\mathsf{acc}_j.\mathsf{x}]_{j=1}^m$*.*
- $\mathsf{ACC.D}(\mathsf{adk}, \mathsf{acc}) \to b$*: on inputs the key* adk*, and an accumulator* $\mathsf{acc} = (\mathsf{acc.x}, \mathsf{acc.w})$*, the decider* $\mathsf{ACC.D}$ *outputs a bit indicating whether* acc *is a valid accumulator.*

The accumulation scheme ACC defined above should satisfy completeness, knowledge soundness given in Definitions 1 and 2, and additionally zero-knowledge as defined below. Note that our definition is a variant of the one given in [BCL+21] that only requires the simulator to simulate an indistinguishable accumulator acc without an accumulation proof pf. Since our PCD construction removes the requirement that the NARK itself be zero-knowledge, our zero-knowledge accumulation scheme must ensure the privacy of both the accumulator and the accumulation proof.

Definition 9 (Zero-Knowledge). *Assume a polynomial-time "honest" adversary* $\mathcal{A}$ *that outputs a tuple* $(\mathsf{i}_\Phi, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m)$ *with probability 1, such that* $\Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{qx}_i, \mathsf{qw}_i) = 1$ *and* $\mathsf{ACC.D}(\mathsf{adk}, \mathsf{acc}_j) = 1$ *for all* $i \in [n]$ *and* $j \in [m]$*.* ACC *is zero-knowledge if there exists a polynomial-time simulator* Sim *such that the following two distributions are (statistically/computationally) indistinguishable.*

$$\left\{ (\mathsf{RO}, \mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{acc}, \mathsf{pf}) \;\middle|\; \begin{aligned} \mathsf{RO} &\leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} &\leftarrow \mathsf{ACC.G}(1^\lambda) \\ \mathsf{pp}_\Phi &\leftarrow \mathcal{H}(1^\lambda) \\ (\mathsf{i}_\Phi, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m) &\leftarrow \mathcal{A}^{\mathsf{RO}}(\mathsf{pp}, \mathsf{pp}_\Phi) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{adk}) &\leftarrow \mathsf{ACC.I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \\ (\mathsf{acc}, \mathsf{pf}) &\leftarrow \mathsf{ACC.P}^{\mathsf{RO}}(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m) \end{aligned} \right\}$$

$$\left\{ (\mathsf{RO}[\mu], \mathsf{pp}, \mathsf{pp}_\varphi, \mathsf{i}_\Phi, \mathsf{acc}, \mathsf{pf}) \;\middle|\; \begin{aligned} \mathsf{RO} &\leftarrow \mathcal{U}(\lambda) \\ (\mathsf{pp}, \tau) &\leftarrow \mathsf{Sim}^{\mathsf{RO}}(1^\lambda) \\ \mathsf{pp}_\Phi &\leftarrow \mathcal{H}(1^\lambda) \\ (\mathsf{i}_\Phi, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m) &\leftarrow \mathcal{A}^{\mathsf{RO}}(\mathsf{pp}, \mathsf{pp}_\Phi) \\ (\mathsf{acc}, \mathsf{pf}, \mu) &\leftarrow \mathsf{Sim}^{\mathsf{RO}}(\tau, \mathsf{pp}_\Phi, \mathsf{i}_\Phi, [\mathsf{qx}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m) \end{aligned} \right\},$$

where $\mathsf{RO}[\mu]$ denotes the random oracle, domain-separated or modified to be consistent with the query-answer list μ. For the description of PCD constructions, we propose the following definition for the representation of accumulation verification in recursive circuits.

Definition 10 (Accumulation Verifier Circuit). *We say that an accumulation scheme for a non-interactive argument of knowledge* NARK *is an accumulation scheme* $\mathsf{ACC} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V}, \mathsf{D})$ *for the pair* $(\Phi, \mathcal{H})$*. We denote* $V^{\lambda,m,N,k}$ *with respect to security parameter* λ*, the number* m *of all accumulated predicate inputs and accumulators, the maximum index size* N*, and the maximum instance size* k*, as the circuit corresponding to the computation of the accumulation verifier* ACC.V*. Moreover, we denote* $v(\lambda, m, N, k)$ *as the size of the circuit* $V^{\lambda,m,N,k}$*,* $|\mathsf{avk}(\lambda, m, N)|$ *as the size of the accumulator verification key* avk*, and* $|\mathsf{acc.x}(\lambda, m, N)|$ *as the size of an accumulator instance.*

Note that the accumulator instance size of acc.x is required to be independent of the instance size bound k. This can be achieved by mapping the original k-length instance acc.x into an image with a collision-resistant hash function in the design of the accumulation scheme[1]. Correspondingly, the new accumulator instance only contains $H(\mathsf{acc.x})$ and acc.x is taken as the witness.

Next, we formalize this theoretical result in Theorem 1. Different from previous works [BCMS20,BCL+21,ZZD23], we introduce a novel construction for zero-knowledge PCD that only depends on the zero-knowledge property of accumulation schemes, thus eliminating the additional cost incurred in achieving zero-knowledge NARKs.

Theorem 1 (Zero-Knowledge PCD Construction (adopted from [BCL+21])). *There exists a polynomial-time transformation* T *such that if* $\mathsf{NARK} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$ *is a (non-zero-knowledge) non-interactive argument of knowledge for circuit satisfiability and* ACC *is a* zero-knowledge *accumulation scheme for* NARK*, then* $\mathsf{PCD} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V}) := T(\mathsf{NARK}, \mathsf{ACC})$ *is a zero-knowledge Proof-Carrying Data scheme for constant-depth compliance predicates, provided*

$$\exists \epsilon \in (0,1) \text{ and a polynomial } \alpha \text{ s.t. } v^*(\lambda, m, N, \sigma) = O(N^{1-\epsilon} \cdot \alpha(\lambda, m, \sigma)).$$

where v^* *is defined as the size of the accumulation verifier invoked in the recursive circuit, where the instance consists of an accumulator verification key* avk*, an accumulator instance, and some additional data of size* σ*. Specifically, we have the following equation that is sublinear in* N*:*

$$v^*(\lambda, m, N, \sigma) := v(\lambda, m, N, |\mathsf{avk}(\lambda, m, N)| + |\mathsf{acc.x}(\lambda, m, N)| + \sigma).$$

Moreover, if NARK *and* ACC *are secure against quantum adversaries, then* PCD *is secure against quantum adversaries. The theoretical performance of* PCD *is highlighted below, where the size of the predicate* $\varphi : \mathbb{F}^{(m+2)\sigma} \to \mathbb{F}$ *is* $f = \omega(\alpha(\lambda, m, \sigma)^{1/\epsilon})$*:*

[1] In Nova, such an operation is explicitly performed by the IVC/PCD prover and verifier.

- *the cost of running* $\mathsf{PCD.I}$ *is equal to the cost of running both* $\mathsf{NARK.I}$ *and* $\mathsf{ACC.I}$ *on an index of size* f *and an index of size* $\circ(f)$*;*
- *the cost of running* $\mathsf{PCD.P}$ *is equal to the cost of accumulating* $m+1$ *instance-proof pairs using* $\mathsf{ACC.P}$*, and running* $\mathsf{NARK.P}$ *twice, one on an index of size* f*, and one on an index of size* $\circ(f)$*.*
- *the cost of running* $\mathsf{PCD.V}$ *is equal to the cost of running* $\mathsf{NARK.V}$ *on an index of size* $\circ(f)$*, and running* $\mathsf{ACC.D}$ *on two indexes of size* f *and* $\circ(f)$ *respectively.*

Note that the theoretical performance of the new PCD construction is slightly larger than the previous construction, since there are two accumulators for different predicates, as explained later. Next, we present the detailed construction of our zero-knowledge PCD as a proof of Theorem 1.

4.2 Construction

Let $\mathsf{NARK} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$ be a (non-zero-knowledge) non-interactive argument of knowledge for circuit satisfiability, and let ACC be a *zero-knowledge* accumulation scheme for NARK. For ease of presentation, we slightly abuse the notation acc to represent a pair of accumulators, i.e., $\mathsf{acc} := (\mathsf{acc}^{(0)}, \mathsf{acc}^{(1)})$, for different predicates $\Phi^{(0)}, \Phi^{(1)}$, respectively. We refer to $\mathsf{acc.x}$ as $(\mathsf{acc}^{(0)}.\mathsf{x}, \mathsf{acc}^{(1)}.\mathsf{x})$. We construct a zero-knowledge PCD as $\mathsf{PCD} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$ as follows.

Given a compliance predicate $\varphi : \mathbb{F}^{(m+2)\sigma} \to \mathbb{F}$, the circuit that realizes the recursion is defined as follows.

$R_{\varphi}^{(\lambda,N,k)}(z; z_{\mathsf{loc}}, [z_i]_{i=1}^m)$:

1: Check that the compliance predicate $\varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m)$ accepts.
2: If the above check holds, output 1; otherwise, output 0.

$R_{V}^{(\lambda,N,k)}(\mathsf{avk}, \mathsf{acc.x}; [z_i, \pi_i.\mathsf{x}, \mathsf{acc}_i.\mathsf{x}]_{i=1}^m, \mathsf{pf})$:

1: If there exists $i \in [m]$ such that $z_i \neq \bot$, check that the verifier accepts
$$V^{(\lambda,m,N,k)}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^m, [\mathsf{acc}_i.\mathsf{x}]_{i=1}^m, \mathsf{acc.x}, \mathsf{pf}) = 1,$$
where $\mathsf{qx}_i := ((\mathsf{avk}, z_i, \mathsf{acc}_i.\mathsf{x}), \pi_i.\mathsf{x})$.
2: If the above check holds, output 1; otherwise, output 0.

Next, we describe the syntax for algorithms in $\mathsf{PCD} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$.

- $\mathsf{PCD.G}(1^\lambda)$: sample $\mathsf{pp}_{\mathsf{NARK}} \leftarrow \mathsf{NARK.G}(1^\lambda)$ and $\mathsf{pp}_{\mathsf{ACC}} \leftarrow \mathsf{ACC.G}(1^\lambda)$, and output $\mathsf{pp} := (\mathsf{pp}_{\mathsf{NARK}}, \mathsf{pp}_{\mathsf{ACC}})$.
- $\mathsf{PCD.I}(\mathsf{pp}, \varphi)$:
  1. compute the integer $N := N(\lambda, |\varphi|, m, \sigma)$, where N is defined in Lemma 1 below.

2. construct the circuits $R^{(0)} := R_{\varphi}^{(\lambda,N,k)}$ and $R^{(1)} := R_{V}^{(\lambda,N,k)}$, where $k := |\mathsf{avk}(\lambda, N)| + |\mathsf{acc.x}(\lambda, m, N)| + \sigma$.
3. compute the index key pairs for the NARK:
   a. $(\mathsf{ipk}^{(0)}, \mathsf{ivk}^{(0)}) \leftarrow \mathsf{NARK.I}(\mathsf{pp}_{\mathsf{NARK}}, R^{(0)})$;
   b. $(\mathsf{ipk}^{(1)}, \mathsf{ivk}^{(1)}) \leftarrow \mathsf{NARK.I}(\mathsf{pp}_{\mathsf{NARK}}, R^{(1)})$.
4. compute the index key triples for the accumulator:
   a. $(\mathsf{apk}^{(0)}, \mathsf{avk}^{(0)}, \mathsf{adk}^{(0)}) := \mathsf{ACC.I}(\mathsf{pp}_{\mathsf{ACC}}, \mathsf{pp}_{\varphi} = \mathsf{pp}_{\mathsf{NARK}}, \mathsf{i}_{\varphi} = R^{(0)})$;
   b. $(\mathsf{apk}^{(1)}, \mathsf{avk}^{(1)}, \mathsf{adk}^{(1)}) := \mathsf{ACC.I}(\mathsf{pp}_{\mathsf{ACC}}, \mathsf{pp}_{\varphi} = \mathsf{pp}_{\mathsf{NARK}}, \mathsf{i}_{\varphi} = R^{(1)})$.
5. output the proving key $\mathsf{ipk} := (\mathsf{ipk}^{(0)}, \mathsf{ipk}^{(1)}, \mathsf{apk}^{(0)}, \mathsf{apk}^{(1)})$ and verification key $\mathsf{ivk} := (\mathsf{ivk}^{(0)}, \mathsf{ivk}^{(1)}, \mathsf{avk}^{(0)}, \mathsf{avk}^{(1)}, \mathsf{adk})$, where $\mathsf{adk} := (\mathsf{adk}^{(0)}, \mathsf{adk}^{(1)})$.

- $\mathsf{PCD.P}(\mathsf{ipk}, z, z_{\mathsf{loc}}, [z_i, \pi_i, \mathsf{acc}_i]_{i=1}^m)$:

1. sample $\pi^{(0)} \leftarrow \mathsf{NARK.P}(\mathsf{ipk}^{(0)}, z; z_{\mathsf{loc}}, [z_i]_{i=1}^m)$.
2. set predicate instance-proof pair as $(\mathsf{qx}_0, \mathsf{qw}_0) =: ((\mathsf{qx}_0^{(0)}, \perp), (\mathsf{qw}_0^{(0)}, \perp))$, where $(\mathsf{qx}_0^{(0)}, \mathsf{qw}_0^{(0)}) := (z, \pi^{(0)}.\mathsf{x}; \pi^{(0)}.\mathsf{w})$.
3. if $z_i = \perp$ for all $i \in [m]$ then set $\mathsf{acc} := (\mathsf{acc}^{(0)}, \mathsf{acc}^{(1)})$, $\mathsf{pf} := (\mathsf{pf}^{(0)}, \mathsf{pf}^{(1)})$, by sampling
   a. $(\mathsf{acc}^{(0)}, \mathsf{pf}^{(0)}) \leftarrow \mathsf{ACC.P}(\mathsf{apk}^{(0)}, (\mathsf{qx}_0, \mathsf{qw}_0), \perp)$;
   b. $(\mathsf{acc}^{(1)}, \mathsf{pf}^{(1)}) \leftarrow \mathsf{ACC.P}(\mathsf{apk}^{(1)}, \perp)$.
4. if $z_i \neq \perp$ for some $i \in [m]$ then:
   a. parse each π_i as $(\perp, \pi_i^{(1)})$;
   b. set predicate input instance $\mathsf{qx}_i := (\perp, \mathsf{qx}_i^{(1)})$, where $\mathsf{qx}_i^{(1)} = (\mathsf{avk}, z_i, \pi_i^{(1)}.\mathsf{x}, \mathsf{acc}_i.\mathsf{x})$;
   c. set predicate input witness $\mathsf{qw}_i := (\perp, \mathsf{qw}_i^{(1)})$, where $\mathsf{qw}_i^{(1)} = (\pi_i^{(1)}.\mathsf{w}, \mathsf{acc}_i.\mathsf{w})$;
   d. compute $(\mathsf{acc}, \mathsf{pf}) \leftarrow \mathsf{ACC.P}(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=0}^m, [\mathsf{acc}_i]_{i=1}^m)$, where $\mathsf{apk} := (\mathsf{apk}^{(0)}, \mathsf{apk}^{(1)})$.
3. compute $\pi^{(1)} \leftarrow \mathsf{NARK.P}(\mathsf{ipk}^{(1)}, \mathsf{avk}, \mathsf{acc.x}; (z, \pi^{(0)}.\mathsf{x}), [z_i, \pi_i.\mathsf{x}, \mathsf{acc}_i.\mathsf{x}]_{i=1}^m, \mathsf{pf})$
4. output $(\pi := (\perp, \pi^{(1)}), \mathsf{acc})$.

- $\mathsf{PCD.V}(\mathsf{ivk}, z, (\pi, \mathsf{acc}))$:

- parse $\pi = (\perp, \pi^{(1)}), \mathsf{acc} = (\mathsf{acc}^{(0)}, \mathsf{acc}^{(1)})$;
- $b_0 \leftarrow \mathsf{NARK.V}(\mathsf{ivk}^{(1)}, (\mathsf{avk}, z, \mathsf{acc.x}), \pi^{(1)})$;
- $b_1 \leftarrow \mathsf{ACC.D}(\mathsf{adk}^{(0)}, \mathsf{acc}^{(0)})$;
- $b_2 \leftarrow \mathsf{ACC.D}(\mathsf{adk}^{(1)}, \mathsf{acc}^{(1)})$;
- output $b_0 \wedge b_1 \wedge b_2$.

Theorem 2. $\mathsf{PCD} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$ *for a set of compliance predicates* $\boldsymbol{F}$ *with constant depth in the construction above satisfies the perfect completeness, computational knowledge soundness, and statistical zero-knowledge under the random oracle model.*

We provide the proof of Theorem 2 above in the next subsection.

Lemma 1 (Efficiency [BCL+21]). *Suppose that for every security parameter* $\lambda \in \mathbb{N}$*, arity* m*, and message size* $t \in \mathbb{N}$*, the ratio of accumulation verifier circuit size to index size* $v^*(\lambda, m, N, k)/N$ *is monotone decreasing in* N*. Then there exists a size function* $N(\lambda, f, m, t)$ *such that*

$$\forall \lambda, f, m, t \in \mathbb{N}, S(\lambda, f, m, t, N(\lambda, f, m, t)) \leq N(\lambda, f, m, t).$$

4.3 Security proofs

Completeness. Assume $\mathcal{A}$ as an arbitrary adversary that breaks the completeness of PCD, i.e., causes the completeness condition given in Definition 5 to be satisfied with probability $1 - p$. We demonstrate that there exists an adversary $\mathcal{B}$, that causes the completeness condition of ACC in Definition 1 to be satisfied with probability at most $1 - p$. The constructed adversary $\mathcal{B}$ runs as follows:

1. set $\mathsf{pp} := (\mathsf{pp}_{\mathsf{NARK}}, \mathsf{pp}_{\mathsf{ACC}})$;
2. compute $(\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i, \mathsf{acc}_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathsf{pp})$;
3. set $(\mathsf{apk}, \mathsf{avk}, \mathsf{adk}) := \mathsf{ACC.I}((\mathsf{pp}_{\mathsf{ACC}}, \mathsf{pp}_{\mathsf{NARK}}), \varphi)$;
4. construct $[(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^m$ as in the algorithm $\mathsf{PCD.P}$;
5. output $(\mathsf{i}_\varphi, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^m, [\mathsf{acc}_i]_{i=1}^m)$, where $\mathsf{i}_\varphi := (R_\varphi^{\lambda,N,k}, R_V^{\lambda,N,k})$.

Suppose that the adversary $\mathcal{A}$ successfully outputs $(\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i, \mathsf{acc}_i]_{i=1}^m)$ such that the precondition holds, i.e., (1) $\varphi \in \mathsf{F}$, (2) $\varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m) = 1$, and (3) for all $i \in [m]$, either $z_i = \perp$ or $\mathsf{PCD.V}(\mathsf{ivk}, z_i, \pi_i) = 1$. Then with probability p, the completeness condition does not hold, which implies $\mathsf{PCD.V}(\mathsf{ivk}, z_i, \pi_i) = 0$. According to the algorithm $\mathsf{PCD.V}$, at least one of the following happens:

1. $0 \leftarrow \mathsf{NARK.V}(\mathsf{ivk}^{(1)}, (\mathsf{avk}, z, \mathsf{acc.x}), \pi^{(1)})$;
2. $0 \leftarrow \mathsf{ACC.D}(\mathsf{adk}^{(0)}, \mathsf{acc}^{(0)})$;
3. $0 \leftarrow \mathsf{ACC.D}(\mathsf{adk}^{(1)}, \mathsf{acc}^{(1)})$.

Next, we discuss these cases one by one.

Case 1. If $z_i = \perp$ for all $i \in [m]$, then, by the perfect completeness of NARK, both instance-proof pairs $(z, \pi^{(0)}.\mathsf{x}; \pi^{(0)}.\mathsf{w})$ and $(\mathsf{avk}, z, \mathsf{acc.x}, \pi^{(1)}.\mathsf{x}; \pi^{(0)}.\mathsf{x})$ trivially hold. Therefore, items 1–3 are all satisfied, which contradicts the assumption that one of them fails.

Case 2. If $z_i \neq \perp$ for some $i \in [m]$, we have precondition (3): $\mathsf{PCD.V}(\mathsf{ivk}, z_i, \pi_i) = 1$ for every relevant i. Given that Item 1 happens, i.e., $0 \leftarrow \mathsf{NARK.V}(\mathsf{ivk}^{(1)}, (\mathsf{avk}, z, \mathsf{acc.x}), \pi^{(1)})$, we know that $R_V^{(\lambda,N,k)}$ rejects $(\mathsf{avk}, \mathsf{acc.x}; [z_i, \pi_i.\mathsf{x}, \mathsf{acc}_i.\mathsf{x}]_{i=1}^m, \mathsf{pf})$, and so

$$\mathsf{ACC.V}(\mathsf{avk}, [\mathsf{qx}_i]_{i=0}^m, [\mathsf{acc}_i.\mathsf{x}]_{i=0}^m, \mathsf{acc.x}, \mathsf{pf}) = 0.$$

Otherwise, for item 2 and 3, we have $\mathsf{ACC.D}(\mathsf{adk}, \mathsf{acc}) = 0$. Now consider the completeness experiment of ACC with adversary $\mathcal{B}$. Since $\mathsf{pp}, \mathsf{pp}_{\mathsf{ACC}}$ are drawn identically to the PCD experiment, the distribution of the output of $\mathcal{A}$ is identical. Hence it holds that for all $i \in [m]$, $\Phi_\varphi(\mathsf{pp}, R_\varphi^{(\lambda,N,k)}, \mathsf{qx}_i^{(0)}; \mathsf{qw}_i^{(0)}) = 1$, $\Phi_V(\mathsf{pp}, R_V^{(\lambda,N,k)}, \mathsf{qx}_i^{(1)}; \mathsf{qw}_i^{(1)}) = 1$, and $\mathsf{ACC.D}(\mathsf{adk}, \mathsf{acc}_i) = 1$. Particularly, for $i = 0$, by the perfect completeness of NARK, the predicate instance-proof pair must satisfy $\Phi_\varphi(\mathsf{pp}, R_\varphi^{(\lambda,N,k)}, \mathsf{qx}_0^{(0)}; \mathsf{qw}_0^{(0)}) = 1$ because $\varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m) = 1$. (precondition in Definition 1). While according to the above discussion, we have either $\mathsf{ACC.V}(\mathsf{avk}, [\mathsf{qx}_i]_{i=0}^m, [\mathsf{acc}_i.\mathsf{x}]_{i=0}^m, \mathsf{acc.x}, \mathsf{pf}) = 0$ or $\mathsf{ACC.D}(\mathsf{adk}, \mathsf{acc}) = 0$. As a result, the adversary $\mathcal{B}$ causes the completeness condition for the accumulation scheme ACC with probability at most $1 - p$.

Knowledge Soundness For convenience, we associate a node u with labels $(z^{(u,v)}, z_{\mathsf{loc}}^{(u)}, \pi^{(u)}, \mathsf{acc}^{(u)})$, where $z^{(u,v)}$ is denoted as $z^{(u)}$ if it is the unique outgoing edge of u, $\pi^{(u)}$ denotes the NARK proof, and $\mathsf{acc}^{(u)}$ denotes the accumulator.

Now given a malicious prover P^*, we can construct an extractor $\mathcal{E}_{\mathsf{P}^*}$ that satisfies Definition 6 through an inductive process.

Base case. In the base case, we consider an extractor $\mathcal{E}_0$ that outputs a tree of depth 1. Trivially, we define $\mathcal{E}_0(\mathsf{pp}_{\mathsf{NARK}}, \mathsf{ai})$ to compute $(\varphi, \circ, \pi, \mathsf{acc}) \leftarrow \mathsf{P}^*(\mathsf{pp}_{\mathsf{NARK}}, \mathsf{ai})$ and output (φ, T_0), where T_0 is a single node labeled with $(\circ, \pi, \mathsf{acc})$.

Inductive step. Next, we inductively consider the iterative case that constructs an extractor $\mathcal{E}_j$ that outputs a tree of depth j. We use the notation $L_{\mathsf{T}}(j)$ to denote the vertices of T at depth j. Assume we already have a constructed extractor $\mathcal{E}_{j-1}$; then $\mathcal{E}_j$ can be constructed in the following steps.

1. Construct a malicious NARK prover $\mathsf{NARK.P}_j^{(1)}$ as follows:

$\mathsf{NARK.P}_j^{(0)}(\mathsf{pp}_{\mathsf{NARK}}, \mathsf{pp}_{\mathsf{ACC}}, \mathsf{ai})$

1: Compute $(\varphi, \mathsf{T}_{j-1}, \mathsf{ao}) \leftarrow \mathcal{E}_{j-1}(\mathsf{pp}_{\mathsf{NARK}}, \mathsf{pp}_{\mathsf{ACC}}, \mathsf{ai})$.
2: For each vertex $v \in L_{\mathsf{T}_{j-1}}(j)$, denote its label by $(z^{(v)}, \pi^{(v)}, \mathsf{acc}^{(v)})$.
3: Run the NARK indexer $(\mathsf{ipk}^{(1)}, \mathsf{ivk}^{(1)}) := \mathsf{NARK.I}(\mathsf{pp}_{\mathsf{NARK}}, R^{(1)})$.
4: Run the accumulator indexer $(\mathsf{apk}^{(0)}, \mathsf{avk}^{(0)}, \mathsf{adk}^{(0)}) := \mathsf{ACC.I}(\mathsf{pp}, R^{(0)})$.
5: Run the accumulator indexer $(\mathsf{apk}^{(1)}, \mathsf{avk}^{(1)}, \mathsf{adk}^{(1)}) := \mathsf{ACC.I}(\mathsf{pp}, R^{(1)})$.
6: Output $(\mathbf{i}, \mathbf{x}, \boldsymbol{\pi}, \mathsf{ao}') :=$
$\Big(\boldsymbol{R}^{(1)}, ((\mathsf{avk}^{(0)}, \mathsf{avk}^{(1)}), z^{(v)}, \mathsf{acc}^{(v)})_{v \in L_{\mathsf{T}_{j-1}}(j)}, (\pi^{(v,1)})_{v \in L_{\mathsf{T}_{j-1}}(j)}, (\varphi, \mathsf{T}_{j-1}, \mathsf{ao})\Big)$

2. Let $\mathcal{E}_{\mathsf{NARK.P}_j^{(1)}}$ be the extractor that corresponds to $\mathsf{NARK.P}_j^{(1)}$ in terms of $\mathsf{ipk}^{(0)}$, via the knowledge soundness of the non-interactive argument NARK.
3. Construct a malicious accumulation scheme prover $\mathsf{ACC.P}_j^*$ as follows:

$\mathsf{ACC.P}_j^*(\mathsf{pp}_{\mathsf{ACC}}, \mathsf{pp}_{\mathsf{NARK}}, \mathsf{ai})$

1: Run the extractor $(\mathbf{i}, \mathbf{x}, \mathbf{w}, \mathsf{ao}') \to \mathcal{E}_{\mathsf{NARK.P}_j^*}(\mathsf{pp}, \mathsf{pp}_{\mathsf{ACC}}, \mathsf{ai})$.
2: Parse the auxiliary output ao' as $(\varphi, \mathsf{T}', \mathsf{ao})$.
3: For each vertex $v \in L_{\mathsf{T}_{j-1}}(j)$,
obtain $\mathsf{acc}^{(v)}$ from T';
obtain $[z_i^{(v)}, \pi_i^{(v)}.\mathsf{x}, \mathsf{acc}_i^{(v)}.\mathsf{x}]_{i=0}^m$ and $\mathsf{pf}^{(v)}$ from $\mathsf{w}^{(v)} \in \mathbf{w}$;
let $S_j := \{v \in L_{\mathsf{T}_{j-1}}(j) : \exists i \in [m], z_i^{(v)} \neq \bot\}$;
attach m children to each $v \in S_j$, where the i-th child is labeled with $z_i^{(v)}$;
parse $\pi_i^{(v)}$ as $(\bot, \pi_i^{(v,1)})$ for $i \in [m]$
define $\mathsf{qx}_i^{(v,1)} := ((\mathsf{avk}, z_i^{(v)}, \mathsf{acc}_i^{(v)}.\mathsf{x}), \pi_i^{(v,1)}.\mathsf{x})$ $\mathsf{qx}_i^{(v)} := (\bot, \mathsf{qx}_i^{(v,1)})$ for $i \in [m]$
define $\mathsf{qx}_i^{(v)} := (\bot, \mathsf{qx}_i^{(v,1)})$ for $i \in [m]$
parse $\pi_0^{(v)}$ as $(\pi_0^{(v,0)}, \bot)$ for $i \in [m]$
define $\mathsf{qx}_0^{(v,0)} := (z_i^{(v)}, \pi_0^{(v,0)})$
define $\mathsf{qx}_0^{(v)} := (\mathsf{qx}_0^{(v,0)}, \bot)$
4: Output $\Big((\mathsf{i}^{(v)}, \mathsf{acc}^{(v)}, \mathsf{pf}^{(v)}, [\mathsf{qx}_i^{(v)}]_{i=1}^m, [\mathsf{acc}_i^{(v)}.\mathsf{x}]_{i=1}^m)_{v \in S_j}, (\varphi, \mathsf{T}', \mathsf{ao})\Big)$.

4. Let $\mathcal{E}_{\mathsf{ACC.P}_j^*}$ be the extractor that corresponds to $\mathsf{ACC.P}_j^*$, via the knowledge soundness of the accumulation scheme ACC.
5. Construct a malicious NARK prover $\mathsf{NARK.P}_j^{(0)}$ as follows:

$\mathsf{NARK.P}_j^{(1)}(\mathsf{pp}_{\mathsf{NARK}}, \mathsf{pp}_{\mathsf{ACC}}, \mathsf{ai})$

1: Run the extractor $\Big((\mathsf{i}^{(v)}, \mathsf{acc}^{(v)}, \mathsf{pf}^{(v)}, [\mathsf{qx}_i^{(v)}, \mathsf{qw}_i^{(v)}]_{i=0}^m, [\mathsf{acc}_i^{(v)}]_{i=1}^m)_{v \in S_j}, \mathsf{ao}'\Big) \leftarrow \mathcal{E}_{\mathsf{ACC.P}_j^*}(\mathsf{pp}_{\mathsf{NARK}}, \mathsf{pp}_{\mathsf{ACC}}, \mathsf{ai})$.
2: For $i = 0$, parse $(\mathsf{qx}_0^{(v)}, \mathsf{qw}_0^{(v)})$ as $(\mathsf{qx}_0^{(v,0)}, \mathsf{qw}_0^{(v,0)}) := ((z, \pi_0^{(v,0)}.\mathsf{x}), \pi_0^{(v,0)}.\mathsf{w})$.
3: Run the NARK indexer $(\mathsf{ipk}^{(0)}, \mathsf{ivk}^{(0)}) := \mathsf{NARK.I}(\mathsf{pp}_{\mathsf{NARK}}, R^{(0)})$.
4: Output $(\mathbf{i}, \mathbf{x}, \boldsymbol{\pi}, \mathsf{ao}') :=$
$$\Big(\boldsymbol{R}^{(0)}, z^{(v)}, (\pi^{(v,0)})_{v \in L_{\mathsf{T}_{j-1}}(j)}, (\varphi, \mathsf{T}_{j-1}, \mathsf{ao})\Big)$$

6. Let $\mathcal{E}_{\mathsf{NARK.P}_j^{(0)}}$ be the extractor that corresponds to $\mathsf{NARK.P}_j^{(0)}$ in terms of $\mathsf{ipk}^{(1)}$, via the knowledge soundness of the non-interactive argument NARK.
7. Define the extractor $\mathcal{E}_j$ as follows:

$\mathcal{E}_j(\mathsf{pp}_{\mathsf{NARK}}, \mathsf{pp}_{\mathsf{ACC}}, \mathsf{ai})$

1: Run the extractor $\left((\mathsf{i}^{(v)}, \mathsf{acc}^{(v)}, \mathsf{pf}^{(v)}, [\mathsf{qx}_i^{(v)}, \mathsf{qw}_i^{(v)}]_{i=0}^m, [\mathsf{acc}_i^{(v)}]_{i=1}^m)_{v \in S_j}, \mathsf{ao}'\right) \leftarrow \mathcal{E}_{\mathsf{ACC.P}_j^*}(\mathsf{pp}_{\mathsf{NARK}}, \mathsf{pp}_{\mathsf{ACC}}, \mathsf{ai})$.

2: Parse the auxiliary output ao' as $(\varphi, \mathsf{T}', \mathsf{ao})$. 3: Let $S_j := \{v \in L_{\mathsf{T}'}(j) : \exists i \in [m], z_i^{(v)} \neq \bot\}$;

4: Run the extractor $(\mathbf{i}, \mathbf{x}, \mathbf{w}, \mathsf{ao}') \leftarrow \mathcal{E}_{\mathsf{NARK.P}_j^{(0)}}(\mathsf{pp}_{\mathsf{NARK}}, \mathsf{pp}_{\mathsf{ACC}}, \mathsf{ai})$.

5: Obtain the local data $z_{\mathsf{loc}}^{(b)}, [z_i^{(v)}]_{i=1}^m$ from $\mathsf{w}^{(v)} \in \mathbf{w}$

6: Parse each $\mathsf{qx}_i^{(v)}$ as $((\mathsf{avk}^{(v)}, z_i^{(v)}, \mathsf{acc}_i^{(v)}.\mathsf{x}), \pi_i^{(v)}.\mathsf{x})$ and $\mathsf{qw}_i^{(v)}$ as $\pi_i^{(v)}.\mathsf{w}$; combine each pair $(\pi_i^{(v)}.\mathsf{x}, \pi_i^{(v)}.\mathsf{w})$ into a proof $\pi_i^{(v)}$.

7: Append $z_{\mathsf{loc}}^{(v)}$ to the label of v in T' and compare $[z_i^{(v)}]_{i=1}^m$.

8: Construct T_j from T' by adding, for each vertex $v \in S_j$, $(\pi_i^{(v)}, \mathsf{acc}_i^{(v)})$ to the label of its i-th child.

9: Output $(\varphi, \mathsf{T}_j, \mathsf{ao})$.

Given that the extractors for NARK and ACC run in expected polynomial time, the extractor $\mathcal{E}_{\mathsf{PCD.P}} = \mathcal{E}_{d(\varphi)}$ also runs in expected polynomial time. Given that knowledge soundness holds for NARK and ACC, the extracted transcript tree is F-compliant. Further details can be found in [BCL+21].

Zero Knowledge The simulator Sim operates as follows.

$\mathsf{Sim}(1^\lambda)$:

1: Sample parameters for NARK: $\mathsf{pp}_{\mathsf{NARK}} \leftarrow \mathsf{NARK.G}(1^\lambda)$.

2: Sample simulated parameters for ACC: $(\mathsf{pp}_{\mathsf{ACC}}, \tau_{\mathsf{ACC}}) \leftarrow \mathsf{ACC.Sim}(1^\lambda)$.

3: Output $(\mathsf{pp} := (\mathsf{pp}_{\mathsf{NARK}}, \mathsf{pp}_{\mathsf{ACC}}), (\mathsf{pp}_{\mathsf{NARK}}, \mathsf{pp}_{\mathsf{ACC}}, \tau_{\mathsf{ACC}}))$.

$\mathsf{Sim}((\mathsf{pp}_{\mathsf{NARK}}, \mathsf{pp}_{\mathsf{ACC}}, \tau_{\mathsf{ACC}}), \varphi, z)$:

1: Compute accumulator keys: $(\mathsf{apk}^{(0)}, \mathsf{avk}^{(0)}, \mathsf{adk}^{(0)}) := \mathsf{ACC.I}(\mathsf{pp}_{\mathsf{ACC}}, \mathsf{pp}_\Phi := \mathsf{pp}_{\mathsf{NARK}}, \mathsf{i}_\Phi = R^{(0)})$.

2: Compute accumulator keys: $(\mathsf{apk}^{(1)}, \mathsf{avk}^{(1)}, \mathsf{adk}^{(1)}) := \mathsf{ACC.I}(\mathsf{pp}_{\mathsf{ACC}}, \mathsf{pp}_\Phi := \mathsf{pp}_{\mathsf{NARK}}, \mathsf{i}_\Phi = R^{(1)})$.

3: Sample instances $[\mathsf{qx}_i]_{i=0}^n, [\mathsf{acc}_j]_{j=1}^m$.

4: Sample simulated accumulator and accumulation proof: $(\mathsf{acc}, \mathsf{pf}) \leftarrow \mathsf{ACC.Sim}(\tau_{\mathsf{ACC}}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi, [\mathsf{qx}_i]_{i=0}^n, [\mathsf{acc}_j]_{j=1}^m)$

5: Compute NARK proof:

$$\pi^{(1)} \leftarrow \mathsf{NARK.P}(\mathsf{ipk}^{(1)}, \mathsf{avk}, \mathsf{acc.x}; (z, \pi_0.\mathsf{x}), [z_i, \pi_i.\mathsf{x}, \mathsf{acc}_i.\mathsf{x}]_{i=1}^m, \mathsf{pf})$$

6: Output $(\pi := (\bot, \pi^{(1)}), \mathsf{acc})$.

Then we argue that the simulated (π, acc) is indistinguishable. Given that the accumulation scheme is zero-knowledge, it is clear that acc is indistinguishable

from the accumulator output by the honest accumulator prover $\mathsf{ACC.P}$ with inputs $(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=0}^{n}, [\mathsf{acc}_j]_{j=1}^{m})$. The indistinguishability of π is not straightforward, since the NARK itself does not provide zero-knowledge. Nevertheless, we note that $\pi^{(1)}$ is generated by the NARK prover with the instance-witness pair as follows:

$$\mathsf{x} := (\mathsf{ipk}^{(1)}, \mathsf{avk}, \mathsf{acc.x}), \ \ \mathsf{w} := ((z, \pi_0.\mathsf{x}), [z_i, \pi_i.\mathsf{x}, \mathsf{acc}_i.\mathsf{x}]_{i=1}^{m}, \mathsf{pf})$$

where $(z, \pi_0.\mathsf{x})$ can be parsed from qx_0, $z_i, \pi_i.\mathsf{x}$ can be parsed from qx_i for $i \in [n]$. Inductively, $[\mathsf{qx}_i]_{i=1}^{n}$ can be simulated trivially given that the previous PCD proof provides zero-knowledge. The accumulation proof can be sampled from $\mathsf{ACC.Sim}$. Therefore, we only need to argue that qx_0 generated at the current node can be simulated efficiently. Note that $\pi_0.\mathsf{x}$ in qx_0 can be computed directly from public inputs φ, z, excluded the commitment to the witness. While for the witness commitment part, the simulator can trivially compute $\mathsf{Commit}(0)$ that is indistinguishable from the honest $\mathsf{Commit}(\mathsf{w})$, given that the underlying commtiment scheme provides statistically hiding property accroding to Definition 11. Therefore, the simulated π is also indistinguishable from the proof honestly generated by the NARK with an instance-witness pair simulated by $\mathsf{ACC.Sim}$.

5 A Concrete Instantiation: zk-Protogalaxy

In this section, we present an efficient zero-knowledge accumulation scheme, providing a concrete instantiation of zero-knowledge PCD as established in Theorem 1. We begin by reviewing the necessary definitions in Section 5.1. Subsequently, we introduce a zero-knowledge accumulation scheme for special-sound protocols, constructed from a variant of Protogalaxy [EG23]. Finally, we present a formal security proof for the proposed scheme, which complements the analysis in [EG23].

5.1 Basic Definitions

Special-Sound Protocols. We adopt the definitions of special-sound protocols [AC20] given in [BC23,EG23]. Define a $(2\mu-1)$-move special-sound protocol Π_{sps} for relation R according to Definition 14 in Appendix C, with verifier degree d, output length n, and two PPT algorithms $\mathsf{P}_{\mathsf{sps}}, \mathsf{V}_{\mathsf{sps}}$. Let the public parameters pp contain μ, d, n, let the public input instance be $\mathsf{x} = \boldsymbol{x}$, and let the private input witness be $\mathsf{w} = \boldsymbol{w}$. At each round $i \in [\mu]$, the prover runs $\mathsf{P}_{\mathsf{sps}}$ to compute the next message $\boldsymbol{m}_i \in \mathbb{F}^{\mu}$ and sends it to the verifier based on previous information including the instance vector $\boldsymbol{x}$, witness vector $\boldsymbol{w}$, and transcripts $\{\boldsymbol{m}_j, r_j\}_{j\in[i]}$ from the previous i rounds. 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 $\mathsf{V}_{\mathsf{sps}}$ to check the validity of the

witness $\boldsymbol{w}$. Note that $\mathsf{V}_{\mathsf{sps}}$ is an algebraic map with maximum degree d as below, where $f_k^{\mathsf{V}_{\mathsf{sps}}}$ is a homogeneous algebraic map with degree k:

$$\mathsf{V}_{\mathsf{sps}}(\boldsymbol{x}, [\boldsymbol{m}_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu-1}) := \sum_{k=0}^{d} f_k^{\mathsf{V}_{\mathsf{sps}}}(\boldsymbol{x}, [\boldsymbol{m}_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu-1}) = \mathbf{0}^n, \tag{2}$$

where degree-k homogeneity means that each monomial term of $f_k^{\mathsf{V}_{\mathsf{sps}}}$ has degree exactly k. We denote the $(2\mu-1)$-move protocol as Π_{sps} and provide the step-by-step description below.

Special-sound protocol Π_{sps} for relation $\mathcal{R}$

P's inputs: $\mathsf{pp}_{\mathsf{sps}}, \boldsymbol{x}; \boldsymbol{w}$

V's inputs: $\mathsf{pp}_{\mathsf{sps}}, \boldsymbol{x}$

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

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

$\mathsf{P} \rightarrow \mathsf{V} : \boldsymbol{m}_i$

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

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

3: $\mathsf{P} \rightarrow \mathsf{V} : \boldsymbol{m}_\mu$

4: V checks if $\mathsf{V}_{\mathsf{sps}}(\boldsymbol{x}, [\boldsymbol{m}_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu-1}) = \mathbf{0}^n$

Zero-Knowledge Sum-Check Protocols. The sum-check protocol is an interactive proof proposed by Lund et al. [LFKN92]. Assume that $\tilde{f}(\boldsymbol{X})$ is an m-variate polynomial with maximum individual degree d. The prover aims to convince the verifier that $s = \sum_{\boldsymbol{x} \in \{0,1\}^m} \tilde{f}(\boldsymbol{x})$. We denote the $(2m+1)$-move sum-check protocol as Π_{sc} and provide the step-by-step description in Π_{sc}. We mention that the interactive sum-check protocol satisfies both completeness and soundness according to [LFKN92], where soundness ensures that any malicious prover cannot forge a valid proof of a false statement with probability more than $d \cdot m/|\mathbb{F}|$ by the Schwartz-Zippel lemma in Lemma 2.

To further achieve the zero-knowledge property, Chiesa et al. proposed an approach in [CFS17] that masks the polynomial f in the sum-check protocol with a random polynomial g, where g has the same variables and individual degrees as f. Xie et al. in [XZZ+19] reduce the size of the random polynomial to $O(d \cdot m)$. Briefly, the random polynomial can be constructed as

$$g(\boldsymbol{X}) = r_0 + r_1(X_1) + r_2(X_2) + \cdots + r_m(X_m), \tag{3}$$

where $r_i(x_i) = r_{i,1}x_i + r_{i,2}x_i^2 + \cdots + r_{i,d}x_i^d, i \in [m]$ is a random univariate polynomial of degree d. We adopt the results in previous works [CFS17,XZZ+19], which show that a simulator producing an indistinguishable transcript can be constructed.

Sum-check protocol Π_{sc}

P_{sc}'s inputs: $\mathsf{pp}, s; \tilde{f}(\boldsymbol{X})$

V_{sc}'s inputs: pp, s

1: For $i \in [m]$

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

$\mathsf{P}_{\mathsf{sc}} \to \mathsf{V}_{\mathsf{sc}} : f_i(X)$

for $i = 1$, V_{sc} checks if $s + \gamma \cdot r = f_1(0) + f_1(1)$

else, V_{sc} checks if $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$

2: V_{sc} checks if $f_m(r_m) = \tilde{f}(r_1, \ldots, r_m)$.

Lemma 2 (Schwartz-Zippel lemma). *Assume an m-variate non-zero polynomial $\tilde{f}(\boldsymbol{X}) \in \mathbb{F}^{(\leq d)}[\boldsymbol{X}], \boldsymbol{X} \in \mathbb{F}^m$, then it holds that*

$$\mathsf{Pr}_{\boldsymbol{r} \leftarrow \$ \mathbb{F}^m}[\tilde{f}(\boldsymbol{r}) = 0] \leq \frac{d \cdot m}{|\mathbb{F}|} \tag{4}$$

Equality Polynomials. 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.

To reduce the potential cross terms generated in accumulating predicate pairs and accumulators, Eagen et al. introduced an elegant linear combination technique based on Lagrange bases in [EG23], which is extended to equality polynomials in [ZGGX25] as the following corollary:

Corollary 1. *A multilinear equality polynomial $\widetilde{eq}_i(\boldsymbol{X})$ defined over the Boolean hypercube $\{0,1\}_{\log m}$ satisfies the following facts:*

- $(\widetilde{eq}_i(\boldsymbol{b}))^d = \widetilde{eq}_i(\boldsymbol{b})$ *holds for all* $d \in \mathbb{N}$ *and* $\boldsymbol{b} \in \{0,1\}_{\log m}$,
- $\widetilde{eq}_i(\boldsymbol{b}) \cdot \widetilde{eq}_j(\boldsymbol{b}) = 0$ *holds for all* $\boldsymbol{b} \in \{0,1\}_{\log m}$ *if and only if* $i \neq j \in [\ell]$.

The proof idea can be found in [ZGGX25]. The above corollary inspires us to conduct an accumulation without introducing cross terms, which is especially important in accumulating multiple high-degree instances. For example, when

computing the accumulation of two equations $V(\boldsymbol{\alpha}) = v_\alpha$ and $V(\boldsymbol{\beta}) = v_\beta$, the derived equation holds for all $b \in \{0,1\}$:

$$V(\boldsymbol{\alpha}) \cdot \widetilde{eq}_0(b) + V(\boldsymbol{\beta}) \cdot \widetilde{eq}_1(b) = v_\alpha \cdot \widetilde{eq}_0(b) + v_\beta \cdot \widetilde{eq}_1(b).$$

Therefore, the prover only needs to provide an additional zero-check proof.

We mention another common technique [XZZ+19,Set20] used in our construction, which can "replace" the random point of an evaluation claim with a new one. That is, given a claim $\widetilde{f}(\boldsymbol{r}_x) = v$, we can derive a new claim $\widetilde{f}(\boldsymbol{r}'_x) = v'$ by running a sum-check protocol for the target polynomial $\widetilde{G}(\boldsymbol{x}) = \widetilde{eq}(\boldsymbol{r}_x, \boldsymbol{x}) \cdot \widetilde{f}(\boldsymbol{x})$ with randomness $\boldsymbol{r}'_x$.

5.2 Full Scheme

Let $\rho_{\mathsf{ACC}}, \rho_{\mathsf{NARK}}, H$ be three random oracles. Next, we describe a zero-knowledge accumulation scheme for the verification algorithm $\mathsf{V}_{\mathsf{NARK}}$ of $\mathsf{NARK} := \mathsf{FS}[\Pi^{\mathsf{cm}}_{\mathsf{sps}}]$, where the NARK is transformed from the special-sound protocol Π_{sps} by committing all messages and applying the Fiat-Shamir heuristic. For simplicity, we assume the commitment scheme provides a hiding property, where each commitment consists of a random element implicitly. The construction of the accumulation scheme generally follows the framework in [BC23,EG23], with the differences from the non-zero-knowledge version highlighted in blue text.

Predicate. The predicate $\Phi := \mathsf{V}_{\mathsf{NARK}}$ to be accumulated is the verification algorithm of the NARK system $\mathsf{FS}[\Pi^{\mathsf{cm}}_{\mathsf{sps}}]$ for relation $\mathcal{R}$. Namely, the predicate takes inputs as public parameters $\mathsf{pp}_{\mathsf{sps}}$, commitment key ck, index $\mathsf{i}_\Phi := \mathsf{V}_{\mathsf{sps}}$, instance vector $\boldsymbol{x} \in \mathbb{F}^k$, random challenges $[r_i]_{i=1}^{\mu}$, and a NARK proof $\pi = (\pi.\mathsf{x}, \pi.\mathsf{w})$, where $\pi.\mathsf{x} = [C_i]_{i=1}^{\mu}$, $\pi.\mathsf{w} = [\boldsymbol{m}_i]_{i=1}^{\mu}$, $C_i = \mathsf{Commit}(\mathsf{ck}, \boldsymbol{m}_i), \forall i \in [\mu]$. Denote qx, qw as

$$\mathsf{qx} := (\boldsymbol{x}, [C_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu-1}), \quad \mathsf{qw} := ([\boldsymbol{m}_i]_{i=1}^{\mu}).$$

Hence, the predicate can be checked by accessing a random oracle ρ_{NARK} as follows:

- $r_i = \rho_{\mathsf{NARK}}(r_{i-1}, C_i)$ for all $i \in [\mu - 1]$ ($r_0 = \rho_{\mathsf{NARK}}(\mathsf{pi})$);
- $C_i = \mathsf{Commit}(\mathsf{ck}, \boldsymbol{m}_i)$ for all $i \in [\mu]$;
- $\sum_{k=0}^{d} f_k^{\mathsf{V}_{\mathsf{sps}}}(\boldsymbol{x}, [\boldsymbol{m}_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu-1}) = \boldsymbol{0}^n$,

Here $f_k^{\mathsf{V}_{\mathsf{sps}}}$ is a degree-k homogeneous algebraic map that outputs n field elements. In this paper, we do not consider the non-uniform case for simplicity, i.e., accumulating instances with different indexes i_Φ. Therefore, the function $\mathsf{V}_{\mathsf{sps}}$ only inputs $\boldsymbol{x}$ as part of pi and leaves the public parameters and index as invariants.

Accumulator. The accumulator has the following format:

- Accumulator instance $\mathsf{acc.x} := (\boldsymbol{x}, [C_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu-1}, E, \boldsymbol{\beta}, \boldsymbol{v}_g)$, where $\boldsymbol{x}$ is the accumulated NARK prover instance vector, $E \in \mathcal{C}$ is the accumulated commitment to the error terms, $\boldsymbol{\beta} \in \mathbb{F}^{\log m+1}$ is a challenge vector, and $\boldsymbol{v}_g \in \mathbb{F}^n$ is a vector of evaluations.
- Accumulator witness $\mathsf{acc.w} := ([\boldsymbol{m}_i]_{i=1}^{\mu}, \boldsymbol{G}(\boldsymbol{X}))$, where $[\boldsymbol{m}_i]_{i=1}^{\mu}$ are the accumulated NARK prover messages, and $\boldsymbol{G}(\boldsymbol{X}) \in \mathbb{F}^{(\leq d)}[\boldsymbol{X}]$ is a vector of random polynomials, which satisfy:

$$\sum_{k=0}^{d} f_k^{\mathsf{V}_{\mathsf{sps}}}(\boldsymbol{x}, [\boldsymbol{m}_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu-1}) = \boldsymbol{e} \wedge E = \mathsf{Commit}(\mathsf{ck}, \boldsymbol{e}) \wedge \boldsymbol{v}_g = \boldsymbol{G}(\boldsymbol{\beta})$$

Generator. The accumulation generator $\mathsf{ACC.G}$ receives as input the security parameter 1^λ and outputs pp.

Indexer. The accumulation indexer $\mathsf{ACC.I}$ receives as inputs accumulation parameters pp and predicate parameters pp_Φ, commitment key ck, and a predicate index i_Φ. The indexer outputs

- accumulator proving key $\mathsf{apk} := (\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ck}, \mathsf{i}_\Phi)$;
- accumulator verification key $\mathsf{avk} := (\mathsf{pp}, \mathsf{i}_\Phi)$;
- decision key $\mathsf{adk} := (\mathsf{pp}, \mathsf{ck})$.

Accumulation prover. On input accumulator proving key apk, accumulators $[\mathsf{acc}_j]_{j=1}^m$, predicate instance-proof pairs $[(\mathsf{qx}_j, \mathsf{qw}_j)]_{j=1}^m$ as defined above, the prover $\mathsf{ACC.P}$ accumulates them into a new accumulator acc by running the following process:

1. For $j \in [m]$, check the consistency of challenges: $\mathsf{qx}_j.r_i = \rho_{\mathsf{NARK}}(\mathsf{qx}_j.r_{i-1}, \mathsf{qx}_j.C_i)$ for all $i \in [\mu - 1]$ $(\mathsf{qx}_j.r_1 = \rho_{\mathsf{NARK}}(\mathsf{qx}_j.\boldsymbol{x}))$.
2. Sample a random predicate instance-witness pair $\mathsf{qx}_0, \mathsf{qw}_0$ with $\mathsf{qx}_0.\boldsymbol{x} \leftarrow\!\!\$\ \mathbb{F}^k$ and $\mathsf{qw}_0.\boldsymbol{m}_i \leftarrow\!\!\$\ \mathbb{F}^t$ for all $i \in [\mu]$, where $\sum_{k=0}^{d} f_k^{\mathsf{V}_{\mathsf{sps}}}(\boldsymbol{x}, [\boldsymbol{m}_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu-1}) = \boldsymbol{e}_0$.
3. Compute a vector of $(\log m + 1)$-variate polynomials as:

$$\boldsymbol{F}(\boldsymbol{X}) := \sum_{k=0}^{d} f_k^{\mathsf{V}_{\mathsf{sps}}}(\boldsymbol{x}(\boldsymbol{X}), [\boldsymbol{m}_i(\boldsymbol{X})]_{i=1}^{\mu}, [r_i(\boldsymbol{X})]_{i=1}^{\mu-1}) \in (\mathbb{F}^{(\leq d)}[\boldsymbol{X}])^n$$

$$\boldsymbol{x}(\boldsymbol{X}) = \widetilde{eq}_0(\boldsymbol{X}) \cdot \mathsf{qx}_0.\boldsymbol{x} + \sum_{j=1}^{m} \left[\widetilde{eq}_{j-1}(\boldsymbol{X}) \cdot \mathsf{qx}_j.\boldsymbol{x} + \widetilde{eq}_{j+m-1}(\boldsymbol{X}) \cdot \mathsf{acc}_j.\mathsf{w}.\boldsymbol{x}\right],$$

$$\boldsymbol{m}_i(\boldsymbol{X}) = \widetilde{eq}_0(\boldsymbol{X}) \cdot \mathsf{qw}_0.\boldsymbol{m}_i + \sum_{j=1}^{m} \left[\widetilde{eq}_{j-1}(\boldsymbol{X}) \cdot \mathsf{qw}_j.\boldsymbol{m}_i + \widetilde{eq}_{j+m-1}(\boldsymbol{X}) \cdot \mathsf{acc}_j.\mathsf{w}.\boldsymbol{m}_i\right],$$

$$r_i(\boldsymbol{X}) = \widetilde{eq}_0(\boldsymbol{X}) \cdot \mathsf{qx}_0.r_i + \sum_{j=1}^{m} \left[\widetilde{eq}_{j-1}(\boldsymbol{X}) \cdot \mathsf{qx}_j.r_i + \widetilde{eq}_{j+m-1}(\boldsymbol{X}) \cdot \mathsf{acc}_j.\mathsf{x}.r_i\right],$$

where $\boldsymbol{F}(\boldsymbol{b}) = \widetilde{eq}_0(\boldsymbol{b}) \cdot \boldsymbol{e}_0 + \sum_{j=1}^{m} \widetilde{eq}_{j+m-1}(\boldsymbol{b}) \cdot \mathsf{acc}_j.\mathsf{w}.\boldsymbol{e} = \boldsymbol{e}(\boldsymbol{b})$ for all $\boldsymbol{b} \in \{0,1\}^{\log m+1}$.

3. $h \leftarrow H(\boldsymbol{x})$.
4. $\boldsymbol{\alpha} \in \mathbb{F}^{\log m+1}, \gamma \in \mathbb{F} \leftarrow \rho_{\mathsf{ACC}}([\mathsf{acc}_j.\mathsf{x}]_{j=1}^{m}, [\mathsf{qx}_j]_{j=1}^{m}, h)$
5. Run non-interactive sum-check protocols $\mathsf{FS}[\Pi_{\mathsf{sc}}]$ in parallel for the following $n + m \cdot n$ statements

$$\sum_{\boldsymbol{b}\in\{0,1\}^{\log m+1}} [eq(\boldsymbol{b},\boldsymbol{\alpha}) \cdot \boldsymbol{F}(\boldsymbol{b}) + \gamma \cdot \boldsymbol{G}'(\boldsymbol{b})] = \sum_{\boldsymbol{b}\in\{0,1\}^{\log m+1}} eq(\boldsymbol{b},\boldsymbol{\alpha}) \cdot \boldsymbol{e}(\boldsymbol{b}) + \gamma \cdot \boldsymbol{e}'_g \in \mathbb{F}^n.$$

$$\sum_{\boldsymbol{b}\in\{0,1\}^{\log m+1}} eq(\mathsf{acc}_j.\mathsf{x}.\boldsymbol{\beta},\boldsymbol{b}) \cdot \mathsf{acc}_j.\mathsf{w}.\boldsymbol{G}(\boldsymbol{b}) = \mathsf{acc}_j.\mathsf{x}.\boldsymbol{v}_g \forall j \in [m]$$

where $\boldsymbol{G}'(\boldsymbol{b}) \in (\mathbb{F}^{(\leq d)}[\boldsymbol{X}])^n$ is a vector of random polynomials and $\boldsymbol{e}'_g := \sum_{\boldsymbol{b}\in\{0,1\}^{\log m+1}} \boldsymbol{G}'(\boldsymbol{b})$ according to Equation 3. The obtained transcript is denoted as:

$$\mathsf{tr} := (\boldsymbol{e}'_g, [\boldsymbol{F}_j(X) \in (\mathbb{F}^{(\leq d)}[X])^{n+m\cdot n}]_{j=1}^{\log m+1}, \boldsymbol{\beta}, \gamma, \boldsymbol{v} \in \mathbb{F}^n, [\boldsymbol{v}'_{g,j}]_{j=1}^{m}).$$

where $\boldsymbol{F}_j(X)$ is a $n+m\cdot n$-length vector of polynomials in round j, $\boldsymbol{v}'_{g,j} := \mathsf{acc}_j.\mathsf{w}.\boldsymbol{G}(\boldsymbol{\beta})$.

6. Compute $\boldsymbol{v}'_g := \boldsymbol{G}'(\boldsymbol{\beta}) \in \mathbb{F}^n$, and set

$$\boldsymbol{e} := \boldsymbol{F}(\boldsymbol{\beta}) = (\boldsymbol{v} - \gamma \cdot \boldsymbol{v}'_g) \cdot eq^{-1}(\boldsymbol{\beta},\boldsymbol{\alpha}) \tag{5}$$

7. Compute $E \leftarrow \mathsf{Commit}(\mathsf{ck}, \boldsymbol{e}) \in \mathcal{C}$.
8. Set vectors

$$\boldsymbol{v}_0 = \left([\mathsf{qx}_0.r_i]_{i=1}^{\mu-1}, [\mathsf{qx}_0.C_i]_{i=1}^{\mu}, \boldsymbol{v}'_g, \mathsf{qw}_0.\boldsymbol{x}, [\mathsf{qw}_0.\boldsymbol{m}_i]_{i=1}^{\mu}, \boldsymbol{G}'(\boldsymbol{X})\right)$$

$$\boldsymbol{v}_j = \left([\mathsf{qx}_j.r_i]_{i=1}^{\mu-1}, [\mathsf{qx}_j.C_i]_{i=1}^{\mu}, \perp, \mathsf{qw}_j.\boldsymbol{x}, [\mathsf{qw}_j.\boldsymbol{m}_i]_{i=1}^{\mu}, \perp\right), j \in [1,m]$$

$$\boldsymbol{v}_{j+m} = \begin{pmatrix} [\mathsf{acc}_j.\mathsf{x}.r_i]_{i=1}^{\mu-1}, [\mathsf{acc}_j.\mathsf{x}.C_i]_{i=1}^{\mu}, \boldsymbol{v}'_{g,j}, \\ \mathsf{acc}_j.\mathsf{w}.\boldsymbol{x}, [\mathsf{acc}_j.\mathsf{w}.\boldsymbol{m}_i]_{i=1}^{\mu}, \mathsf{acc}_j.\mathsf{w}.\boldsymbol{G}(\boldsymbol{X}) \end{pmatrix}, j \in [1,m].$$

8. Compute the accumulated vector

$$\boldsymbol{v} := ([r_i]_{i=1}^{\mu-1}, [C_i]_{i=1}^{\mu}, \boldsymbol{v}_g, \boldsymbol{x}, [\boldsymbol{m}_i]_{i=1}^{\mu}, \boldsymbol{G}) = \sum_{j=0}^{2m} \widetilde{eq}_{j-1}(\boldsymbol{\beta}) \cdot \boldsymbol{v}_j.$$

9. Set the new accumulator acc as

$$\mathsf{acc.x} := (\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \boldsymbol{x}, [C_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu-1}, E, \boldsymbol{\beta}, \boldsymbol{v}_g)$$
$$\mathsf{acc.w} := ([\boldsymbol{m}_i]_{i=1}^{\mu}, \boldsymbol{G}(\boldsymbol{X})).$$

10. Set accumulation proof $\mathsf{pf} := (\mathsf{qx}_0, \mathsf{tr}, \boldsymbol{v}'_g)$.

Accumulation verifier. On input accumulator verification key avk, accumulator instances $[\mathsf{acc}_j.\mathsf{x}]_{j=1}^m$, predicate instances $[\mathsf{qx}_j]_{j=1}^m$, a new accumulator instance $\mathsf{acc.x}$, and an accumulation proof pf, the verifier $\mathsf{ACC.V}$ checks the accumulation by running the following process:

1. For $j \in [0, m]$, check the consistency of challenges: $\mathsf{qx}_j.r_i = \rho_{\mathsf{NARK}}(\mathsf{qx}_j.r_{i-1}, \mathsf{qx}_j.C_i)$ for all $i \in [\mu - 1]$ ($\mathsf{qx}_j.r_1 = \rho_{\mathsf{NARK}}(\mathsf{qx}_j.\boldsymbol{x})$).
2. Check the correctness of sum-check proof $\mathsf{pf.tr}$ by running the verifier of $\mathsf{FS}[\Pi_{\mathsf{sc}}]$ in terms of $\boldsymbol{\beta}, \gamma \leftarrow \rho_{\mathsf{ACC}}([\mathsf{acc}_j.\mathsf{x}]_{j=1}^m, [\mathsf{qx}_j]_{j=1}^m, h)$.
3. Set vectors

$$\begin{aligned}
\boldsymbol{v}_0 &= \left([\mathsf{qx}_0.r_i]_{i=1}^{\mu-1}, [\mathsf{qx}_0.C_i]_{i=1}^{\mu}, \perp, \boldsymbol{v}'_g\right) \\
\boldsymbol{v}_j &= \left([\mathsf{qx}_j.r_i]_{i=1}^{\mu-1}, [\mathsf{qx}_j.C_i]_{i=1}^{\mu}, \perp, \perp\right), j \in [1, m] \\
\boldsymbol{v}_{j+m} &= \left([\mathsf{acc}_j.\mathsf{x}.r_i]_{i=1}^{\mu-1}, [\mathsf{acc}_j.\mathsf{x}.C_i]_{i=1}^{\mu}, \mathsf{acc}_j.\mathsf{x}.E, \boldsymbol{v}'_{g,j}\right), j \in [1, m].
\end{aligned}$$

4. Check if the tuple $([r_i]_{i=1}^{\mu-1}, [C_i]_{i=1}^{\mu}, E, \boldsymbol{v}_g)$ in $\mathsf{acc.x}$ equals $\sum_{j=0}^{2m} \widetilde{eq}_{j-1}(\boldsymbol{\beta}) \cdot \boldsymbol{v}_j$.

Decider. On inputs the decider key adk and an accumulator

$$\mathsf{acc} = (\mathsf{acc.x} = (\boldsymbol{x}, [C_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu-1}, E, \boldsymbol{\beta}, \boldsymbol{v}_g), \mathsf{acc.w} = ([\boldsymbol{m}_i]_{i=1}^{\mu}, \boldsymbol{G}(\boldsymbol{X})))$$

the decider does the checks described as follows.

1. check $C_i \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, \boldsymbol{m}_i)$ for all $i \in [\mu]$;
2. compute $\boldsymbol{e} \leftarrow \sum_{k=0}^{d} f_k^{\mathsf{V}_{\mathsf{sps}}}(\boldsymbol{x}, [\boldsymbol{m}_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu-1})$;
3. check if $E = \mathsf{Commit}(\mathsf{ck}, \boldsymbol{e})$,
4. check if $\boldsymbol{G}(\boldsymbol{\beta}) = \boldsymbol{v}_g$.

Theorem 3. *Let $\mathsf{V}_{\mathsf{NARK}}$ be the verifier of the NARK system $\mathrm{FS}[\Pi_{\mathsf{sps}}^{\mathsf{cm}}]$ in terms of ρ_{sps} for relation R^n under the random oracle model. Let $\mathsf{CM} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ be a homomorphic commitment scheme with binding and hiding properties. Let ρ_{acc} be another random oracle. The accumulation scheme $\mathsf{ACC} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V}, \mathsf{D})$ for $\mathsf{V}_{\mathsf{NARK}}$ satisfies perfect completeness, knowledge soundness and zero-knowledge.*

We present the proof of Theorem 3 in Appendix A.

Complexity. We analyze the theoretical performance of the accumulation scheme. Denote by s the length of instance vector $\boldsymbol{x}$ and t the maximum length of messages $[\boldsymbol{m}_i]_{i \in [\mu]}$. We assume the commitment scheme is conducted over cyclic groups for simplicity.

- The accumulation prover asks μ queries to ρ_{NARK}, and $\log m$ queries to ρ_{ACC}. Computes $E = \mathsf{Commit}(\mathsf{ck}, \boldsymbol{e})$ with n $\mathbb{G}$ operations. Performs $O(m \cdot (\mu + s + t))$ $\mathbb{F}$ operations and $m \cdot \mu$ $\mathbb{G}$ operations to accumulate predicate pairs and accumulators. Performs $O(m \cdot n \cdot d \cdot t \cdot \mu)$ $\mathbb{F}$ operations for proving the sum-check protocol.

- The accumulation verifier asks μ queries to ρ_{NARK}, and $\log m$ queries to ρ_{ACC}. Performs $O(m \cdot \mu)$ $\mathbb{F}$ operations and $m \cdot (\mu + 1)$ $\mathbb{G}$ operations to accumulate predicate instances and accumulator instances. Performs $O(\log m \cdot n \cdot d)$ $\mathbb{F}$ operations for verification of the sum-check protocol.
- The decider computes the commitments with $O(\mu \cdot t + n)$ $\mathbb{G}$ operations. Performs computations of the degree-s homogeneous algebraic map with inputs $(s + \mu \cdot t + \mu)$-length inputs for all $s \in [d]$.

Optimizations. Note that for simplicity, we do not present some optimizations explicitly in the above accumulation scheme. Here we mention a technique mentioned in [BC23] to compress the n degree-d checks in the verification algorithm $\mathsf{V}_{\mathsf{sps}}$ into one. As a result, the $n + m \cdot n$ sum-check statements can be reduced to only $1 + n$. At the final step, the verifier additionally samples a challenge $\delta \leftarrow\!\!\$\ \mathbb{F}$ and the prover responds with a vector $\boldsymbol{\delta} = (\delta, \delta^2, \delta^4, \ldots, \delta^{2^{\log \nu - 1}})$ (assume n is a power of 2). For $j \in [n]$, let $S \subset \{0, \ldots, \log n - 1\}$ be the set such that $j = \sum_{k \in S} 2^k + 1$. The verifier derives $\delta^j = \mathsf{pow}_j(\boldsymbol{\delta})$, where pow_j is a $\log n$-variate polynomial. Accordingly, the prover and verifier agree on a linearly combined map with degree $D = d + \log n$ as follows:

$$\sum_{i \in [n]} \mathsf{pow}_j(\boldsymbol{\delta}) \cdot \mathsf{V}_j(\mathsf{pi}, [\boldsymbol{m}_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu}) = 0,$$

where $\mathsf{V}_j(\cdot) := \mathsf{V}_{\mathsf{sps}}(\cdot)[j]$ is the j-th equation checked by $\mathsf{V}_{\mathsf{sps}}$.

References

AC20. Thomas Attema and Ronald Cramer. Compressed Σ-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics. In *Proc. of the Annual International Cryptology Conference (CRYPTO)*, pages 513–543. Springer, 2020.

ACK21. Thomas Attema, Ronald Cramer, and Lisa Kohl. A compressed σ-protocol theory for lattices. In *Annual International Cryptology Conference*, pages 549–579. Springer, 2021.

Ajt96. Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108, 1996.

BBB+18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.

BC23. Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special-sound protocols. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 77–110. Springer, 2023.

BC24. Dan Boneh and Binyi Chen. Latticefold: A lattice-based folding scheme and its applications to succinct proof systems. *Cryptology ePrint Archive*, 2024.

BC25. Dan Boneh and Binyi Chen. Latticefold+: faster, simpler, shorter lattice-based folding for succinct proof systems. In *Annual International Cryptology Conference*, pages 327–361. Springer, 2025.

BCCT13. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 111–120, 2013.

BCFW25. Benedikt Bünz, Alessandro Chiesa, Giacomo Fenzi, and William Wang. Linear-time accumulation schemes. *Cryptology ePrint Archive*, 2025.

BCG21. Elette Boyle, Ran Cohen, and Aarushi Goel. Breaking the $o(\sqrt{n})$-bit barrier: Byzantine agreement with polylog bits per party. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 319–330, 2021.

BCL+21. Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In *Annual International Cryptology Conference*, pages 681–710. Springer, 2021.

BCMS20. Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. *Cryptology ePrint Archive*, 2020.

BDFG21. Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. 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*, pages 649–680. Springer, 2021.

BGH19. Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. *Cryptology ePrint Archive*, 2019.

BMNW24. Benedikt Bünz, Pratyush Mishra, Wilson Nguyen, and William Wang. Accumulation without homomorphism. *Cryptology ePrint Archive*, 2024.

BMNW25. Benedikt Bünz, Pratyush Mishra, Wilson Nguyen, and William Wang. Arc: Accumulation for reed–solomon codes. In *Annual International Cryptology Conference*, pages 128–160. Springer, 2025.

BMRS20a. Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. *Cryptology ePrint Archive*, 2020.

BMRS20b. Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Mina: Decentralized cryptocurrency at scale. *New York Univ. O (1) Labs, New York, NY, USA, Whitepaper*, pages 1–47, 2020.

BSBHR19. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *Annual international cryptology conference*, pages 701–732. Springer, 2019.

BSCTV17. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. *Algorithmica*, 79(4):1102–1160, 2017.

CCDW20. Weikeng Chen, Alessandro Chiesa, Emma Dauterman, and Nicholas P Ward. Reducing participation costs via incremental verification for ledger systems. *Cryptology ePrint Archive*, 2020.

CCYY25. Mingshu Cong, Sherman SM Chow, Siu Ming Yiu, and Tsz Hon Yuen. Scalable zksnarks for matrix computations: A generic framework for verifiable deep learning. *Cryptology ePrint Archive*, 2025.

CFS17. Alessandro Chiesa, Michael A Forbes, and Nicholas Spooner. A zero knowledge sumcheck and its applications. *arXiv preprint arXiv:1704.02086*, 2017.

Che25. Binyi Chen. Symphony: Scalable snarks in the random oracle model from lattice-based high-arity folding. *Cryptology ePrint Archive*, 2025.

COS20. Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. 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*, pages 769–793. Springer, 2020.

CT10. Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *ICS*, volume 10, pages 310–331, 2010.

CTV13. Stephen Chong, Eran Tromer, and Jeffrey A Vaughan. Enforcing language semantics using proof-carrying data. *Cryptology ePrint Archive*, 2013.

CTV15. Alessandro Chiesa, Eran Tromer, and Madars Virza. Cluster computing in zero knowledge. In *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II 34*, pages 371–403. Springer, 2015.

Dia21. Benjamin E Diamond. Many-out-of-many proofs and applications to anonymous zether. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1800–1817. IEEE, 2021.

EG23. Liam Eagen and Ariel Gabizon. Protogalaxy: Efficient protostar-style folding of multiple instances. *Cryptology ePrint Archive*, 2023.

FS86. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016.

GWC19. Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

Hab22. Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. *Cryptology ePrint Archive*, 2022.

KKC+25. Jiwon Kim, Abhiram Kothapalli, Orestis Chardouvelis, Riad S Wahby, and Paul Grubbs. Alpaca: Anonymous blocklisting with constant-sized updatable proofs. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 3364–3382. IEEE, 2025.

KS22. Abhiram Kothapalli and Srinath Setty. Supernova: Proving universal machine executions without universal circuits. *Cryptology ePrint Archive*, 2022.

KS24. Abhiram Kothapalli and Srinath Setty. Hypernova: Recursive arguments for customizable constraint systems. In *Annual International Cryptology Conference*, pages 345–379. Springer, 2024.

KST22. Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Annual International Cryptology Conference*, pages 359–388. Springer, 2022.

LFKN92. Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992.

NDC+24. Wilson Nguyen, Trisha Datta, Binyi Chen, Nirvan Tyagi, and Dan Boneh. Mangrove: A scalable framework for folding-based snarks. In *Annual International Cryptology Conference*, pages 308–344. Springer, 2024.

nex24. Nexus zkvm. https://docs.nexus.xyz/zkvm/nexus-zkvm, 2024. The Nexus zkVM is licensed under the source-available Business Source License (BUSL), converting to an open-source dual Apache 2.0 and MIT licensing on February 10th, 2029.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.

STW23. Srinath Setty, Justin Thaler, and Riad Wahby. Customizable constraint systems for succinct arguments. *Cryptology ePrint Archive*, 2023.

Val08. Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of cryptography conference*, pages 1–18. Springer, 2008.

XZZ+19. Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pages 733–764. Springer, 2019.

ZGGX23. Tianyu Zheng, Shang Gao, Yu Guo, and Bin Xiao. Kilonova: Non-uniform pcd with zero-knowledge property from generic folding schemes. *Cryptology ePrint Archive*, 2023.

ZGGX25. Tianyu Zheng, Shang Gao, Yu Guo, and Bin Xiao. quasar: sublinear accumulation schemes for multiple instances. *Cryptology ePrint Archive*, 2025.

ZGSX23. Tianyu Zheng, Shang Gao, Yubo Song, and Bin Xiao. Leaking arbitrarily many secrets: Any-out-of-many proofs and applications to ringct protocols. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2533–2550. IEEE, 2023.

ZZD23. Zibo Zhou, Zongyang Zhang, and Jin Dong. Proof-carrying data from multi-folding schemes. *Cryptology ePrint Archive*, 2023.

A Security Proof of Theorem 3

In this part, we provide the proofs for the security properties of the proposed accumulation scheme. Since the proof of completeness and knowledge soundness follows the general idea in previous works [BC23], we omit some details for simplicity.

Completeness. Assume the inputs consist of m satisfying predicate instance-proof pairs $[\mathsf{qx}_j, \mathsf{qw}_j]_{j=1}^m$ and m satisfying accumulators $[\mathsf{acc}_j]_{j=1}^m$. Let $\mathsf{acc}, \mathsf{pf}$ denote the output of the accumulation prover $\mathsf{ACC.P}$. We argue that both the accumulation verifier and the decider will accept, i.e.,

$$\mathsf{acc.V}^{\mathsf{RO}}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, [\mathsf{acc}_j.\mathsf{x}]_{j=1}^m, \mathsf{acc.x}, \mathsf{pf}) = 1 \wedge \mathsf{acc.D}(\mathsf{adk}, \mathsf{acc}) = 1$$

For $\mathsf{ACC.V}$, since it honestly computes the linear combination, the obtained tuple $([r_i]_{i=1}^{\mu-1}, [C_i]_{i=1}^{\mu}, E, \boldsymbol{v}_g)$ in $\mathsf{acc.x}$ is consistent with the prover's output.

For the decider $\mathsf{ACC.D}(\mathsf{adk}, \mathsf{acc})$ with input accumulator

$$\mathsf{acc} = (\mathsf{acc.x} = (\boldsymbol{x}, [C_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu-1}, E, \boldsymbol{\beta}, \boldsymbol{v}_g), \mathsf{acc.w} = ([\boldsymbol{m}_i]_{i=1}^{\mu}, \boldsymbol{G}(\boldsymbol{X})))$$

we argue that:

1. $\mathsf{acc}.C_i = \mathsf{Commit}(\mathsf{ck}, \boldsymbol{m}_i)$ for all $i \in [\mu]$ due to the homomorphism of the commitment scheme.
2. the computed $\boldsymbol{e} \leftarrow \sum_{k=0}^{d} f_k^{\mathsf{V}_{\mathsf{sps}}}(\boldsymbol{x}, [\boldsymbol{m}_i]_{i=1}^{\mu}, [r_i]_{i=1}^{\mu-1})$ satisfies that

$$\boldsymbol{e} := \boldsymbol{F}(\boldsymbol{\beta}) = \sum_{k=0}^{d} f_k^{\mathsf{V}_{\mathsf{sps}}}(\boldsymbol{x}(\boldsymbol{\beta}), [\boldsymbol{m}_i(\boldsymbol{\beta})]_{i=1}^{\mu}, [r_i(\boldsymbol{\beta})]_{i=1}^{\mu-1}).$$

   Again $E = \mathsf{Commit}(\mathsf{ck}, \boldsymbol{e})$ holds by the homomorphism of the commitment scheme.
3. $\boldsymbol{G}(\boldsymbol{\beta}) = \boldsymbol{v}_g$ because it is the linear combination of the $m+1$ equations

$$\boldsymbol{G}'(\boldsymbol{\beta}) = \boldsymbol{v}'_g, \ \mathsf{acc}_j.\mathsf{w}.\boldsymbol{G}(\boldsymbol{\beta}) = \boldsymbol{v}'_{g,j} \forall j \in [m].$$

Knowledge Soundness. We prove that the accumulation scheme has knowledge soundness by showing that there exists an underlying $(2\log m + 5)$-move interactive proof with $(2, 2, \ldots, 2)$-special soundness. Consider the public-coin interactive proof $\Pi_I = (\mathsf{P}_I, \mathsf{V}_I)$ where P_I first sends image h and V_I responds with challenges $\boldsymbol{\alpha}, \gamma$. Next, Π_I, V_I engage in a $2\log m + 3$-move sum-check protocol interactively, where Π_I sends $\boldsymbol{F}_{\log m+1}(X), \boldsymbol{v}, [\boldsymbol{v}'_{g,j}]_{j=1}^m, \boldsymbol{v}'_g$ at the final move,

Extracting witness. We first show that given $2^{\log m+2} = 4m$ accepting transcripts with distinct challenges, there exists an extractor outputting $[\mathsf{qw}_j]_{j=1}^m, [\mathsf{acc}_j.\mathsf{w}]_{j=1}^m$. Specifically, given $4m$ accepting transcripts with distinct $\gamma, \boldsymbol{\beta}$, the extractor can compute the following

$$\mathsf{qw}_j.\boldsymbol{m}_i, \forall i \in [\mu], j \in [0,m],\ \mathsf{acc}_j.\mathsf{w}.\boldsymbol{x}, \forall j \in [0,m],$$
$$\mathsf{acc}_j.\mathsf{w}.\boldsymbol{m}_i, \forall i \in [\mu], j \in [0,m],\ \mathsf{acc}_j.\mathsf{w}.\boldsymbol{G}(\boldsymbol{X}), \forall j \in [0,m].$$

Otherwise, the extractor will break the binding property of the commitment scheme by finding a collision for one of the linear combinations below, where $i \in [\mu]$:

$$\mathsf{acc.x}.C_i = \widetilde{eq}_0(\boldsymbol{\beta}) \cdot \mathsf{qx}_0.C_i + \sum_{j=1}^{m} \left[\widetilde{eq}_{j-1}(\boldsymbol{\beta}) \cdot \mathsf{qx}_j.C_i + \widetilde{eq}_{j+m-1}(\boldsymbol{\beta}) \cdot \mathsf{acc}_j.\mathsf{x}.C_i\right],$$

which is implied by the precondition: $\mathsf{ACC.V}(\mathsf{avk}, [\mathsf{qx}_j, \mathsf{acc}_j.\mathsf{x}]_{j=1}^m, \mathsf{acc.x}, \mathsf{pf}) = 1$.

Ensuring Predicates. Additionally, the extractor needs to ensure that all the extracted witnesses satisfy the constraint $\mathsf{V}_{\mathsf{sps}}$. Since both $\mathsf{ACC.V}$ and $\mathsf{ACC.D}$ pass, the extracted witness, except with negligible probability, satisfies:

$$eq(\boldsymbol{\beta}, \boldsymbol{\alpha}) \cdot \boldsymbol{F}(\boldsymbol{\beta}) + \gamma \cdot \boldsymbol{v}'_g = \boldsymbol{v}$$
$$\mathsf{acc}_j.\mathsf{w}.\boldsymbol{G}(\boldsymbol{\beta}) = \boldsymbol{v}'_{g,j} \forall j \in [m]$$

Then, by the soundness of the sum-check protocol, we have the following

$$\sum_{\boldsymbol{b} \in \{0,1\}^{\log m+1}} [eq(\boldsymbol{b}, \boldsymbol{\alpha}) \cdot \boldsymbol{F}(\boldsymbol{b}) + \gamma \cdot \boldsymbol{G}'(\boldsymbol{b})] = \sum_{\boldsymbol{b} \in \{0,1\}^{\log m+1}} eq(\boldsymbol{b}, \boldsymbol{\alpha}) \cdot \boldsymbol{e}(\boldsymbol{b}) + \gamma \cdot \boldsymbol{e}'_g \in \mathbb{F}^n.$$
$$\sum_{\boldsymbol{b} \in \{0,1\}^{\log m+1}} eq(\mathsf{acc}_j.\mathsf{x}.\boldsymbol{\beta}, \boldsymbol{b}) \cdot \mathsf{acc}_j.\mathsf{w}.\boldsymbol{G}(\boldsymbol{b}) = \mathsf{acc}_j.\mathsf{x}.\boldsymbol{v}_g \forall j \in [m]$$

Then, by the Schwartz-Zippel lemma, the extracted witness satisfies the following constraints:

$$\sum_{k=0}^{d} f_k^{\mathsf{V}_{\mathsf{sps}}}(\mathsf{qx}_j.\boldsymbol{x}, [\mathsf{qw}_j.\boldsymbol{m}_i]_{i=1}^{\mu}, [\mathsf{qx}_j.r_i]_{i=1}^{\mu-1}) = \boldsymbol{0}^n, \forall j \in [m],$$
$$\sum_{k=0}^{d} f_k^{\mathsf{V}_{\mathsf{sps}}}(\mathsf{acc}_j.\mathsf{w}.\boldsymbol{x}, [\mathsf{acc}_j.\mathsf{w}.\boldsymbol{m}_i]_{i=1}^{\mu}, [\mathsf{acc}_j.\mathsf{x}.r_i]_{i=1}^{\mu-1}) = \mathsf{acc}_j.\mathsf{w}.\boldsymbol{e},$$
$$\mathsf{acc}_j.\mathsf{x}.E = \mathsf{Commit}(\mathsf{ck}, \mathsf{acc}_j.\mathsf{w}.\boldsymbol{e})$$
$$\mathsf{acc}_j.\mathsf{x}.\boldsymbol{v}_g = \mathsf{acc}_j.\mathsf{w}.\boldsymbol{G}(\boldsymbol{\beta}).$$

Otherwise, the extractor finds a collision for the following linear combination:

$$\mathsf{acc}_j.\mathsf{x}.E = \sum_{j=1}^{m} \widetilde{eq}_{j+m-1}(\boldsymbol{\beta}) \cdot \mathsf{acc}_j.\mathsf{x}.E.$$

Zero-Knowledge. Consider an adversary $\mathcal{A}$ outputting $(\mathsf{i}_{\Phi}, [(\mathsf{qx}_j, \mathsf{qw}_j)]_{j=1}^m, [\mathsf{acc}_j]_{j=1}^m)$, such that $\Phi(\mathsf{pp}_{\Phi}, \mathsf{i}_{\Phi}, \mathsf{qx}_j, \mathsf{qw}_j) = 1$ and $\mathsf{ACC.D}(\mathsf{adk}, \mathsf{acc}_j) = 1$ for all $j \in [m]$. We show that it is efficient to build a simulator Sim with inputs $(\tau = \bot, \mathsf{pp}_{\Phi}, \mathsf{i}_{\Phi}, [\mathsf{qx}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m)$, outputting an accumulator acc^* and a proof pf^* that are indistinguishable from those output by an honest prover.

Concretely, given inputs $\mathsf{pp}_{\Phi}, \mathsf{i}_{\Phi}$, the simulator first samples the elements in acc^* as $\boldsymbol{x}^* \leftarrow\!\!\$\ \mathbb{F}^k$, $\boldsymbol{m}_i^* \leftarrow\!\!\$\ \mathbb{F}^t$ for all $i \in [\mu]$, $r_i^* \leftarrow\!\!\$\ \mathbb{F}$ for all $i \in [\mu - 1]$, and $\boldsymbol{\beta} \leftarrow\!\!\$\ \mathbb{F}^{\log m+1}$. Next, the simulator simulates the accumulation proof pf^*, consisting of a random predicate instance qx_0, a sum-check transcript tr^*, and a vector $(\boldsymbol{v}'_g)^*$. For qx_0, the simulator computes

$$\mathsf{qx}_0.r_i := \widetilde{eq}_0^{-1}(\boldsymbol{\beta}) \cdot \left[r_i^* - \sum_{j=1}^{m} \left(\widetilde{eq}_{j-1}(\boldsymbol{\beta}) \cdot \mathsf{qx}_j.r_i + \widetilde{eq}_{j+m-1}(\boldsymbol{\beta}) \cdot \mathsf{acc}_j.\mathsf{x}.r_i \right) \right], \forall i \in [\mu - 1]$$

$$\mathsf{qx}_0.C_i := \widetilde{eq}_0^{-1}(\boldsymbol{\beta}) \cdot \left[C_i^* - \sum_{j=1}^{m} \left(\widetilde{eq}_{j-1}(\boldsymbol{\beta}) \cdot \mathsf{qx}_j.C_i + \widetilde{eq}_{j+m-1}(\boldsymbol{\beta}) \cdot \mathsf{acc}_j.\mathsf{x}.C_i \right) \right], \forall i \in [\mu]$$

where $C_i^* := \mathsf{Commit}(\mathsf{ck}, \boldsymbol{m}_i^*)$. The predicate instance vector $\mathsf{qx}_0.\boldsymbol{x}$ can be randomly sampled from $\mathbb{F}^k$.

For the sum-check proof, the simulator randomly samples two polynomial vectors $\boldsymbol{F}^*(\boldsymbol{X}), (\boldsymbol{G}'(\boldsymbol{X}))^* \in (\mathbb{F}^{(\leq d)}[\boldsymbol{X}])^n$ satisfying the following evaluation claims:

$$\boldsymbol{F}^*(\boldsymbol{\beta}) = \boldsymbol{e} \wedge (\boldsymbol{G}'(\boldsymbol{\beta}))^* = \boldsymbol{v}'_g$$

Then the simulator Sim runs the simulator $\mathsf{Sim}_{\mathsf{sc}}$ of the zero-knowledge protocol on the above evaluation claims, and obtains a simulated proof tr for the following statement:

$$\sum_{\boldsymbol{b} \in \{0,1\}^{\log m+1}} [eq(\boldsymbol{b}, \boldsymbol{\alpha}) \cdot \boldsymbol{F}(\boldsymbol{b}) + \gamma \cdot \boldsymbol{G}'(\boldsymbol{b})] = \sum_{\boldsymbol{b} \in \{0,1\}^{\log m+1}} eq(\boldsymbol{b}, \boldsymbol{\alpha}) \cdot \boldsymbol{e}(\boldsymbol{b}) + \gamma \cdot \boldsymbol{e}'_g \in \mathbb{F}^n.$$

$$\sum_{\boldsymbol{b} \in \{0,1\}^{\log m+1}} eq(\mathsf{acc}_j.\mathsf{x}.\boldsymbol{\beta}, \boldsymbol{b}) \cdot \mathsf{acc}_j.\mathsf{w}.\boldsymbol{G}(\boldsymbol{b}) = \mathsf{acc}_j.\mathsf{x}.\boldsymbol{v}_g \forall j \in [m]$$

Note that to ensure the simulated sum-check proof is accepted by the verifier, the simulator needs to program the random oracle ρ_{NARK} by writing each query-response pair $[(\boldsymbol{F}_j(X), \beta_{j-1}), \beta_j], \forall j \in [\log m + 1]$.

After that, the simulator constructs the accumulator acc^* by computing the remaining elements accordingly:

$$C_i^* := \mathsf{Commit}(\mathsf{ck}, \boldsymbol{m}_i^*), \forall i \in [\mu]$$

$$E^* := \mathsf{Commit}(\mathsf{ck}, \boldsymbol{e}),\ (\boldsymbol{v}_g)^* = \widetilde{eq}_0(\boldsymbol{\beta}) \cdot (\boldsymbol{v}'_g)^* + \sum_{j=1}^{m} \widetilde{eq}_{j+m-1}(\boldsymbol{\beta}) \cdot \boldsymbol{v}'_{g,j}$$

$$(\mathsf{acc}^*).\mathsf{x} := (\boldsymbol{x}^*, [C_i^*]_{i=1}^{\mu}, [r_i^*]_{i=1}^{\mu-1}, E^*, \boldsymbol{\beta}, (\boldsymbol{v}_g)^*),\ (\mathsf{acc}^*).\mathsf{w} := ([\boldsymbol{m}_i^*]_{i=1}^{\mu}, \boldsymbol{G}(\boldsymbol{X}))$$

Clearly, the accumulator acc^* is accepted by the decider $\mathsf{ACC.D}$. By writing the following query-response pairs into the random oracle ρ_{NARK}, the pair $(\mathsf{acc}^*, \mathsf{pf}^*)$ is accepted by the accumulation verifier $\mathsf{ACC.V}$:

- Write the pair $[\mathsf{qx}_0.\boldsymbol{x}, \mathsf{qx}_0.r_1]$ so that $\mathsf{qx}_0.r_0 = \rho_{\mathsf{NARK}}(\mathsf{qx}_0.\boldsymbol{x})$.
- Write the pair $[(\mathsf{qx}_0.r_{i-1}, \mathsf{qx}_0.C_i), \mathsf{qx}_0.r_i]$ for all $i \in [\mu - 1]$.

Finally, we show that acc^* is indistinguishable from the acc generated in the honest case by $\mathsf{ACC.P}^{\mathsf{RO}}(\mathsf{apk}, [(\mathsf{qx}_j, \mathsf{qw}_j)]_{j=1}^m, [\mathsf{acc}_j]_{j=1}^m)$, because the random predicate pair $(\mathsf{qx}_0, \mathsf{qw}_0)$ is chosen uniformly at random and the resulting random linear combination with the other inputs $[(\mathsf{qx}_j, \mathsf{qw}_j)]_{j=1}^m, [\mathsf{acc}_j]_{j=1}^m$ preserves the target distribution of $(\mathsf{acc.x}, \mathsf{acc.w})$.

B Related Works

Prior to the development of accumulation and folding schemes, the only known method for constructing PCD was through recursive composition of succinct non-interactive arguments (SNARGs) [BCCT13,CTV15,COS20]. In this approach, a proof for the correctness of t computation steps consists of a proof for the claim: "the t-th step was executed correctly, and there exists a proof that the computation was executed correctly for $t-1$ steps." The latter part is encoded using the SNARG verifier itself. This construction yields secure PCD (with IVC as a special case) provided the SNARG satisfies adaptive knowledge soundness (i.e., is a SNARK). The efficiency and security of the resulting PCD scheme correspond to those of a single invocation of the SNARK.

In 2019, Halo [BGH19] and subsequent works [BDFG21,BCMS20,BCL+21] introduced a practical approach to deferring expensive cryptographic operations in proof verification by batching them at the end of IVC/PCD protocols. Instead of verifying SNARK or (non-succinct) NARK proofs from previous steps within the recursive circuit, the prover employs an *accumulation scheme* to defer these checks and continue incremental computation. Specifically, an accumulation scheme enables the prover to convince the verifier that multiple instances can be linearly combined into a single *running instance*, whose verification is significantly more efficient than directly checking all previous proofs. By recursively accumulating at each step, the protocol produces a final instance representing the entire computation, with size comparable to a single step, which is then verified by the IVC/PCD verifier.

More recently, Nova [KST22] introduced a highly efficient IVC construction based on *folding schemes*. Unlike Halo, where the prover accumulates SNARK/NARK *proofs* compiled with polynomial commitment schemes (PCS) such as KZG or Bulletproofs [BBB+18], Nova's approach applies folding schemes—such as random linear combinations—directly to NP *instances* derived from Interactive Oracle Proofs (IOP), without requiring PCS compilation. This distinction leads to substantial performance improvements, as verifying uncompiled NP instances involves fewer cryptographic operations than verifying proofs. Subsequent works,

including SuperNova [KS22], Protostar [BC23], Protogalaxy [EG23], HyperNova [KS24] and KiloNova [ZGGX23] have further developed these ideas. As the distinction between "folding" and "accumulation" has become less pronounced in recent literature, we use these terms interchangeably in this paper.

Lattice-based accumulation schemes can leverage Ajtai's partially homomorphic commitments [Ajt96]. The main challenge in this setting is controlling the norm growth of the committed (and extracted) witness, which is essential for enabling unbounded accumulation. Recent works [BC24,BC25,Che25] have made significant progress in constructing lattice-based accumulation schemes that address this challenge. However, to the best of our knowledge, no existing construction achieves a linear-time prover, due either to superlinear commitment costs or superlinear complexity in the accumulation prover. Achieving a lattice-based accumulation scheme that controls norm growth while maintaining a linear-time prover remains an open problem.

Additionally, some recent constructions in the random oracle model (ROM) utilize Merkle commitment schemes [BMNW24,BCFW25,ZGGX25], but their provers run in quasilinear time due to the use of Reed–Solomon codes and polynomial quotients.

C Definitions

C.1 Commitment Schemes

A commitment scheme is a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{Commit})$ [AC20].

- $\mathsf{ck} \leftarrow \mathsf{Gen}(1^\lambda)$: on input security parameter λ, outputs a commitment key ck, where ck specifies a message space $\mathcal{M}_c$, a key space $\mathcal{K}_c$, a commitment space $\mathcal{C}_c$, a randomness space $\mathcal{R}_c$ and an opener space $\mathcal{O}_c$.
- $c_m \leftarrow \mathsf{Commit}(\mathsf{ck}; m)$: on input commitment key $\mathsf{ck} \in \mathcal{K}_c$, a message $m \in \mathcal{M}_c$, a randomness $r \in \mathcal{R}_c$, outputs a commitment c_m. Parameter r can be omitted when the sampling is implicit.

Typically, a commitment scheme should satisfy binding and hiding properties defined in Appendix C.1.

Definition 11. (Binding Property) *A non-interactive commitment scheme* $(\mathsf{Gen}, \mathsf{Com})$ *is* computationally binding *if for all PPT adversaries* $\mathcal{A}$*, there is a negligible function* $\mu(\lambda)$ *such that*

$$\Pr\left[\begin{array}{l|l} \mathsf{Commit}(\mathsf{ck}; m_0; r_0) & \mathsf{ck} \leftarrow \mathsf{Gen}(1^\lambda); \\ = \mathsf{Commit}(\mathsf{ck}; m_1; r_1) & (m_0, m_1, r_0, r_1) \leftarrow \mathcal{A}(\mathsf{ck}) \\ \wedge\ m_0 \neq m_1 & \end{array}\right] = \mu(\lambda).$$

where $\mathcal{A}$ *outputs* $m_0, m_1 \in \mathcal{M}_c$ *and* $r_0, r_1 \in \mathcal{R}_c$*. The scheme is* ***perfectly binding*** *when* $\mu(\lambda) = 0$*.*

Definition 12. (Hiding Property) *A non-interactive commitment scheme* $(\mathsf{Gen}, \mathsf{Commit})$ *is* computationally hiding *if for all PPT adversaries* $\mathcal{A}$*, there is a negligible function* $\mu(\lambda)$ *such that*

$$\left| \Pr \left[\mathcal{A}(c) = b \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Gen}(1^\lambda); \\ (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{ck}), \\ b \leftarrow \{0,1\}, r \leftarrow\!\!\$\, \mathcal{R}_c, \\ c \leftarrow \mathsf{Commit}(\mathsf{ck}; m_b; r) \end{array} \right] - \frac{1}{2} \right| = \mu(\lambda),$$

where $\mathcal{A}$ *outputs* $m_0, m_1 \in \mathcal{M}$*. The scheme is* ***perfectly hiding*** *when* $\mu(\lambda) = 0$*.*

To guarantee that PCD is statistical zero-knowledge, we require the undelying commitment scheme at least provides statistical hiding in this paper.

C.2 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 P to convince a verifier V that a statement x admits a witness w. Typically, in a $(2\mu - 1)$-move protocol, the prover and the verifier interactively send μ messages and $\mu - 1$ challenges. If 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 [FS86]. 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 3). Briefly speaking, an interactive proof satisfies special soundness if there exists an efficient extractor outputting a valid witness with inputs of the instance 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 the definition of the tree of transcripts.

Definition 13 (Tree of transcripts). *Let* $k \in \mathbb{N}$ *and* $(a_1, \ldots, a_k) \in \mathbb{N}^k$*. A* $(a_1, \ldots, a_k)$*-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 14 ($(a_1, \ldots, a_k)$-special soundness). *Π provides* $(a_1, \ldots, a_k)$*-special soundness if there exists an effective PPT extraction algorithm* $\mathcal{E}$ *that is capable of extracting the witness* w *given* x *and any* $(a_1, \ldots, a_k)$*-tree of accepting transcripts* T *[ACK21] (defined above). Specifically, for all PPT adversaries* $\mathcal{A}$

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

According to the conclusion given in [AC20], we can derive knowledge soundness from special soundness:

Lemma 3 ($(a_1, \ldots, a_k)$-special soundness implies knowledge soundness [ACK21]). *Let $k, a_1, \ldots, a_k \in \mathbb{N}$, $(\mathsf{G}, \mathsf{P}, \mathsf{V})$ be a $(a_1, \ldots, a_k)$-special-sound $(2k+1)$-move interactive protocol for relation $\mathcal{R}$, where V samples each challenge uniformly at random from $\mathbb{F}$. Then $(\mathsf{G}, \mathsf{P}, \mathsf{V})$ is knowledge sound with knowledge error*

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

C.3 Security Definition for NARK

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

$$\Pr\left[\begin{array}{c} (\mathsf{i}, \mathsf{x}, \mathsf{w}) \notin \mathsf{R}(\mathsf{pp}) \\ \vee \\ \mathsf{NARK.V}^{\mathsf{RO}}(\mathsf{ivk}, \mathsf{x}, \pi) = 1 \end{array} \middle| \begin{array}{r} \mathsf{RO} \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NARK.G}^{\mathsf{RO}}(1^\lambda) \\ (\mathsf{i}, \mathsf{x}, \mathsf{w}) \leftarrow \mathcal{A}^{\mathsf{RO}}(\mathsf{pp}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathsf{NARK.I}^{\mathsf{RO}}(\mathsf{pp}, \mathsf{i}) \\ \pi \leftarrow \mathsf{NARK.P}^{\mathsf{RO}}(\mathsf{ipk}, \mathsf{x}, \mathsf{w}) \end{array}\right] = 1.$$

Knowledge soundness. NARK is knowledge sound with respect to auxiliary input distribution Ψ if for every expected polynomial-time adversary $\tilde{\mathsf{P}}$, there exists an expected polynomial-time extractor $\mathcal{E}$ such that for every set Z,

$$\Pr\left[\begin{array}{c} (\mathsf{pp}, \mathsf{ai}, \mathbf{i}, \mathbf{x}, \mathsf{ao}) \in Z \\ \wedge \\ \forall j \in [\ell], \mathsf{NARK.V}^{\mathsf{RO}}(\mathsf{ivk}_j, \mathsf{x}_j, \mathsf{w}_j) = 1 \end{array} \middle| \begin{array}{r} \mathsf{RO} \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NARK.G}(1^\lambda) \\ \mathsf{ai} \leftarrow \Psi(\mathsf{pp}) \\ (\mathbf{i}, \mathbf{x}, \boldsymbol{\pi}, \mathsf{ao}) \leftarrow \tilde{\mathsf{P}}^{\mathsf{RO}}(\mathsf{pp}, \mathsf{ai}) \\ \forall j \in [\ell], (\mathsf{ipk}_j, \mathsf{ivk}_j) \leftarrow \mathsf{NARK.I}^{\mathsf{RO}}(\mathsf{pp}, \mathsf{i}_j) \end{array}\right]$$

$$\leq \Pr\left[\begin{array}{c} (\mathsf{pp}, \mathsf{ai}, \mathbf{i}, \mathbf{x}, \mathsf{ao}) \in Z \\ \wedge \\ \forall j \in [\ell], (\mathsf{i}_j, \mathsf{x}_j, \mathsf{w}_j) \in \mathsf{R}(\mathsf{pp}) \end{array} \middle| \begin{array}{r} \mathsf{RO} \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{NARK.G}(1^\lambda) \\ \mathsf{ai} \leftarrow \Psi(\mathsf{pp}) \\ (\mathbf{i}, \mathbf{x}, \mathbf{w}, \mathsf{ao}) \leftarrow \mathcal{E}^{\tilde{\mathsf{P}}}(\mathsf{pp}, \mathsf{ai}) \end{array}\right] + \mathsf{negl}(\lambda).$$

We omit the random oracle if the context is clear. For more details, we encourage the readers to refer to the definitions in [COS20].