Accumulation without Homomorphism

Benedikt Bünz
bb@nyu.edu
New York University

Pratyush Mishra
prat@upenn.edu
University of Pennsylvania

Wilson Nguyen
wdnguyen@stanford.edu
Stanford University

William Wang
ww@priv.pub
New York University

March 25, 2024

Abstract

Accumulation schemes are a simple yet powerful primitive that enable highly efficient constructions of incrementally verifiable computation (IVC). Unfortunately, all prior accumulation schemes rely on homomorphic vector commitments whose security is based on public-key assumptions. It is an interesting open question to construct efficient accumulation schemes that avoid the need for such assumptions.

In this paper, we answer this question affirmatively by constructing an accumulation scheme from *non-homomorphic* vector commitments which can be realized from solely symmetric-key assumptions (e.g. Merkle trees). We overcome the need for homomorphisms by instead performing spot-checks over error-correcting encodings of the committed vectors.

Unlike prior accumulation schemes, our scheme only supports a bounded number of accumulation steps. We show that such *bounded-depth* accumulation still suffices to construct proof-carrying data (a generalization of IVC). We also demonstrate several optimizations to our PCD construction which greatly improve concrete efficiency.

Contents

1 Introduction

Proof-carrying data (PCD) [CT10] is a powerful cryptographic primitive that enables mutually distrustful parties to perform distributed computations that run indefinitely, while ensuring that the correctness of every intermediate step can be verified efficiently. PCD is a generalization of the prior notion of *incrementally-verifiable computation* (IVC) [Val08].[1]

PCD has found numerous applications in both theory and practice, including enforcing language semantics [CTV13], complexity-preserving succinct arguments [BCCT13; BCTV17], verifiable MapReduce computations [CTV15], image provenance [NT16], and consensus protocols and blockchains [Mina; KB20; BMRS20; CCDW20; BCG24]. It is thus of great interest to design efficient PCD schemes and to understand the minimal assumptions they can rely on. Existing efforts on this front fall into two categories.

PCD from succinctly verifiable arguments. The standard construction of PCD is via recursive composition of succinct non-interactive arguments of knowledge (SNARKs) [BCCT13; BCTV14; BCTV17; COS20]. Informally, to prove a t-step computation, the PCD prover proves that the t-th step is correct, and there exists a valid proof for the first $t-1$ steps. The culmination of this line of works is Fractal [COS20], which constructs PCD from SNARKs in the random oracle model.[2] This means that we can achieve PCD from cheap symmetric-key cryptography, but at the cost of relying on the existence of SNARKs, which are complex to construct, impose high proving overheads, and additionally face barriers in instantiantiations from falsifiable assumptions [GW11].

PCD from accumulation. A recent popular approach to avoid this reliance on SNARKs is to construct PCD via *accumulation schemes* [BCMS20; BCLMS21; KST22]. Roughly speaking, instead of recursively checking proofs as above, the PCD prover "accumulates" the proof for each step into a running accumulator, and then the PCD verifier performs a single expensive check on the final accumulator. This line of work has led to simple and efficient PCD schemes with low prover overhead, and so has seen much interest in new constructions and deployments [BCLMS21; KST22; BC23; EG23; KS23b; KS23a]. Unfortunately, all known accumulation schemes incur overheads due to their reliance on relatively expensive homomorphic vector commitments that are only known to exist under public-key assumptions. Additionally, because existing homomorphic vector commitments are not known to achieve post-quantum security, the resulting accumulation schemes are also quantum-insecure.

Our question. We are thus left in a curious state of affairs: on the one hand we have PCD from symmetric-key-based SNARKs, and on the other we have PCD from accumulation that does not rely on SNARKs, but which instead introduces public-key assumptions. This motivates the questions we tackle in this paper: can we design PCD that does not rely on SNARKs, and also does not require public-key assumptions? That is, can we design non-trivial accumulation schemes that depend only on symmetric-key cryptography?

1.1 Our contributions

We answer these questions positively: (1) We introduce a new notion of *bounded-depth* accumulation schemes that support a limited number of accumulations. (2) We show that the latter implies *bounded-depth* PCD, which, by known results [BCCT13], suffices for obtaining polynomial-depth IVC. (3) We construct efficient bounded-depth accumulation schemes from any (non-homomorphic) vector commitment scheme (e.g. random-oracle based Merkle trees) and any linear code. As we show in Table 1, the resulting PCD

[1] IVC is the special case of PCD where the distributed computation graph is a line.

[2] The concrete PCD construction makes non-black-box use of the SNARK verifier, which requires us to heuristically instantiate the random oracle.

scheme	prover overhead	verifier	supported IVC length
Fractal [COS20]	$O(\lambda \log n)$	$O(\lambda \log n)$	$\mathrm{poly}(\lambda)$
this paper	$O(d\lambda)$	$O(dn)$	m^d
+ batch comm. (Sec 2.5)	$O(d\lambda/m)$	$O(dmn)$	m^d
+ low-overhead IVC (Sec 2.6)	$O(d\lambda/m^2)$	$O(dmn)$	m^d
ours up to depth $d^\star$ + Fractal (Sec 2.7)	$O(d^\star\lambda/m^2 + \lambda \log n^\star/m^{d^\star})$	$O(\lambda \log n^\star)$	$\mathrm{poly}(\lambda)$

Table 1: Comparison of IVC schemes constructed from PCD over a tree of depth d and arity m. All costs are number of vector commitment openings, and prover overhead is per-step of IVC. Above n is the size of the recursive circuit, and $n^\star = O(d^\star mn)$ is the circuit size of the accumulation decider for the recursive circuit.

construction asymptotically reduces the prover overhead compared to the best prior random oracle based construction, and achieves plausible post-quantum security.[3] (4) We provide several optimizations for the instantiated PCD scheme, including support for 'batch' accumulation, a new low-overhead compiler from low-depth PCD to IVC, and a new *hybrid* PCD scheme that combines our low-depth PCD with any SNARK-based PCD scheme to achieve the best of both worlds.

1.2 Related work

PCD from symmetric-key assumptions. As noted in Section 1, the only end-to-end construction of PCD from symmetric-key assumptions is that of Chiesa, Ojha, and Spooner [COS20]. We provide a quantitative comparison in Table 1, and focus here on a qualitative comparison. Their construction is based on the Fractal SNARK, which they prove secure in the random oracle model.[4]

Boneh, Drake, Fisch, and Gabizon [BDFG21] propose an optimization of the foregoing approach that batches the most expensive component, the low-degree test, across multiple proofs. While this concretely reduces prover cost, it does not lead to an asymptotic improvement in the prover overhead.

Like Fractal and similar SNARKs, our construction is able to take advantage of recent advances in the design of efficient code-based Interactive Oracle Proofs (IOPs) [BCS16; RRR21]. For example, like recent work [Sta21; Pol; DP23b], we can greatly improve efficiency by relying on extension fields of small characteristic. Furthermore, unlike existing works, our fields do not need to have any special algebraic structure (e.g. large multiplicative subgroups).

PCD from public-key assumptions. Except the foregoing, all existing concretely-efficient IVC/PCD constructions [BCTV17; BGH19; BCMS20; BCLMS21; BDFG21; KST22; KS23b; KS23a; BC23; EG23] rely on public-key assumptions, and in particular rely on the hardness of computing discrete logarithms over elliptic curve groups, which forces the usage of cryptographically large fields. Furthermore, efficient implementations require *cycles of elliptic curves*, which have proved unwieldy to implement correctly in practice [NBS23]. In comparison, our construction avoids the need for public-key assumptions and this additional algebraic structure, and is able to use non-cryptographic field sizes.

Concurrent work. The concurrent work LatticeFold [BC24] takes a complementary approach to constructing plausibly post-quantum accumulation-based PCD: it constructs an accumulation/folding scheme from lattice-based assumptions. Unlike our construction, their accumulation scheme directly supports unbounded depth,

[3] We only claim plausible post-quantum security, as we prove our construction in the random oracle model, instead of the quantum random oracle model [BDFLSZ11].

[4] Like us, to achieve PCD, they must instantiate the random oracle with a concrete hash function, which results in only heuristic security.

allowing it to serve as a drop-in replacement in many existing constructions of accumulation-based PCD. However, this comes at a cost: at each accumulation step, the prover must demonstrate that the accumulation result has a small norm, which incurs a non-trivial computational cost. Furthermore, the dependence on lattice-based assumptions means that their scheme still relies on public-key assumptions.

An interesting direction for future work would be to combine the two approaches to reduce the need for small-norm checks. For example, one could construct a more efficient PCD scheme by first designing a bounded-depth accumulation scheme that relies on the bounded homomorphism supported by lattice commitments. One could then use the transformation of Section 2.7 to combine the resulting (bounded-depth) PCD scheme with the (unbounded-depth) LatticeFold PCD scheme to achieve a more efficient construction than either scheme alone.

Remark 1.1 (security of bounded-depth PCD)**.** All PCD schemes (including ours) only provably support computation graphs of depth $O(1)$. However, while there are no known attacks that break the security of prior schemes when the depth is $\omega(1)$, the same is not true for our scheme. As we explain in Section 2.1, our scheme is vulnerable to a relatively straightforward attack that obviates any security guarantees when the depth of the computation graph exceeds an *a priori* fixed constant. We emphasize that even such bounded-depth PCD is already powerful enough to support many interesting applications, including the primary application of constructing polynomial-length IVC [BCCT13]. See Remarks 2.1 and 2.2 for a more detailed discussion.

2 Techniques

We begin by reviewing the definition of an accumulation scheme [BCMS20; BCLMS21].[5] At a high level, it is used to perform *batch verification* of a predicate which, for us, will be a non-interactive argument's verifier $\mathcal{V}$. In other words, an accumulation scheme is used to check that $\mathcal{V}(\mathbb{x}_1, \pi_1), \ldots, \mathcal{V}(\mathbb{x}_n, \pi_n)$ all accept, more efficiently than the naive approach of individually verifying each instance $\mathbb{x}_i$ and proof π_i.

The workflow of an accumulation scheme is as follows. There are three main algorithms: a prover P, verifier V, and decider D. The prover is initialized with an empty accumulator acc_0, which is used to accumulate an input $(\mathbb{x}_1, \pi_1)$ into a new accumulator acc_1. The prover additionally outputs a proof; we write this as $(\mathsf{acc}_1, \mathsf{pf}_1) \leftarrow \mathrm{P}(\mathbb{x}_1, \pi_1, \mathsf{acc}_0)$. Later, acc_1 can be used to accumulate a second input, i.e. $(\mathsf{acc}_2, \mathsf{pf}_2) \leftarrow \mathrm{P}(\mathbb{x}_2, \pi_2, \mathsf{acc}_1)$, and so on. The correctness of a sequence of accumulations can then be established by checking that: (a) each accumulation step is valid, i.e. $\mathrm{V}(\mathbb{x}_i, \pi_i, \mathsf{acc}_{i-1}, \mathsf{acc}_i, \mathsf{pf}_i) = 1$; and (b) the final accumulator is valid, i.e. $\mathrm{D}(\mathsf{acc}_n) = 1$.

Notice that the decider only acts on the final accumulator, whereas the verifier acts on each accumulation step. Therefore, the crux of an accumulation scheme is making verification as cheap as possible. Towards this, we require that an accumulator acc can be split into a short instance part $\mathsf{acc}.\mathbb{x}$ and a (possibly) long witness part $\mathsf{acc}.\mathbb{w}$; we use $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ as shorthand. Similarly, we require that an argument proof can be split into instance and witness parts $\pi = (\pi.\mathbb{x}, \pi.\mathbb{w})$. The point is that the verifier can only look at the instance parts; we write this as $\mathrm{V}(\mathbb{x}_i, \pi_i.\mathbb{x}, \mathsf{acc}_{i-1}.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}, \mathsf{pf}_i)$.

Definition. An accumulation scheme must satisfy the following properties.

- *Completeness:* The honest accumulation $(\mathsf{acc}', \mathsf{pf}) \leftarrow \mathrm{P}(\mathbb{x}, \pi, \mathsf{acc})$ of *any* valid input and accumulator should pass both the verifier's and decider's checks. That is, if $\mathcal{V}(\mathbb{x}, \pi) = 1$ and $\mathrm{D}(\mathsf{acc}) = 1$, then $\mathrm{V}(\mathbb{x}, \pi.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}'.\mathbb{x}, \mathsf{pf}) = 1$ and $\mathrm{D}(\mathsf{acc}') = 1$.

- *Knowledge soundness:* If a new accumulator acc' passes the verifier's and decider's checks, then an efficient extractor can find a valid input and old accumulator that explains acc'. That is, if $\mathrm{D}(\mathsf{acc}') = 1$ and $\mathrm{V}(\mathbb{x}, \pi.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}'.\mathbb{x}, \mathsf{pf}) = 1$, then an efficient extractor can find the witness part of the proof, $\pi.\mathbb{w}$, and the witness part of the old accumulator, $\mathsf{acc}.\mathbb{w}$, such that $\mathcal{V}(\mathbb{x}, \pi) = 1$ and $\mathrm{D}(\mathsf{acc}) = 1$.

- *Efficiency:* The cost of running the accumulation verifier n times plus the cost of running the accumulation decider once should be lower than the cost of running the argument verifier n times.

Accumulation schemes can be generalized to handle multiple inputs and accumulators in each step. For example, the prover's syntax would be $\mathrm{P}([\mathbb{x}_i, \pi_i]_{i=1}^{m_1}, [\mathsf{acc}_i]_{i=1}^{m_2})$, where m_1 and m_2 are the arities; see Section 4 for a comprehensive definition.

Prior constructions. All prior accumulation schemes [BCMS20; BCLMS21; KST22; BC23; EG23; KS23b; KS23a] crucially use *additively homomorphic vector commitment schemes*. Informally, a vector commitment scheme allows one to construct a succinct commitment cm to a vector $\mathbf{v} \in \mathbb{F}^n$. The scheme is additively homomorphic if, given $\mathsf{cm}_1 = \mathsf{Commit}(\mathbf{v}_1)$ and $\mathsf{cm}_2 = \mathsf{Commit}(\mathbf{v}_2)$, $\mathsf{cm}_3 = \alpha \cdot \mathsf{cm}_1 + \beta \cdot \mathsf{cm}_2$ is a commitment to $\alpha\mathbf{v}_1 + \beta\mathbf{v}_2$. We remark that all known additively homomorphic vector commitment schemes, e.g. Pedersen commitments [Ped92], rely on public-key assumptions.

The general blueprint for an accumulation scheme is as follows. An accumulator witness $\mathsf{acc}.\mathbb{w} \in \mathbb{F}^n$ is a vector, and the corresponding instance $\mathsf{acc}.\mathbb{x}$ is a commitment to $\mathsf{acc}.\mathbb{w}$. For simplicity, suppose the prover claims that acc_1 and acc_2 accumulate into acc_3. Roughly speaking, we want to guarantee that the output

[5] We restrict our presentation to *split* accumulation schemes, as defined in [BCLMS21].

accumulator is a random linear combination of the input accumulators. The verifier checks this by computing the linear combination of the input commitments $\mathsf{acc}_1.\mathbb{x}$ and $\mathsf{acc}_2.\mathbb{x}$, and checking that the result equals the output commitment $\mathsf{acc}_3.\mathbb{x}$ [KST22; BC23]. Later, the decider will check that $\mathsf{acc}_3.\mathbb{w}$ is a "good" vector, and that $\mathsf{acc}_3.\mathbb{x}$ commits to $\mathsf{acc}_3.\mathbb{w}$. Since commitments are binding, acc_3 must be the correct linear combination of acc_1 and acc_2.

We have omitted many details, most notably how to accumulate argument proofs. However, from this description alone we can observe two key properties of the vector commitment scheme. First, it has succinct commitments; this allows the verifier to be efficient. Second, it is additively homomorphic; this allows the verifier to perform meaningful checks. As noted earlier, this combination of properties unfortunately seems to require public-key assumptions.

2.1 Checking linearity

To overcome the foregoing limitation, we make a key observation: to verify that the output accumulator is a linear combination of the input accumulators, it is not necessary to directly compute a linear combination of the input commitments. Instead, it suffices to *check* that the output accumulator commits to a vector that is a linear combination of the vectors committed by the input accumulators. This idea is a natural one, and has appeared before under the name of "linear combination schemes" [BDFG21].

Recall that we want to check that $\mathsf{acc}_3.\mathbb{w} = \alpha \cdot \mathsf{acc}_1.\mathbb{w} + \beta \cdot \mathsf{acc}_2.\mathbb{w}$, where $\alpha, \beta \in \mathbb{F}$ are previously chosen scalars. More precisely, we want to check that $\mathbf{v}_3 = \alpha\mathbf{v}_1 + \beta\mathbf{v}_2$, where $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$ are the underlying committed vectors of $\mathsf{acc}_1.\mathbb{x}$, $\mathsf{acc}_2.\mathbb{x}$, and $\mathsf{acc}_3.\mathbb{x}$ (and since commitments are binding, these vectors correspond with the accumulator witnesses). A *linearity check* is a protocol between the prover and the verifier that convinces the verifier of this claim. Assuming the verifier is public-coin, this can be made non-interactive using random oracles.

Our goal is to construct a linearity check which does not require homomorphic vector commitments. Instead, we require vector commitments with *local openings*. Informally, these allow the prover to generate a succinct proof that the underlying committed vector's i-th element is some claimed value. For example, Merkle tree commitments support local openings with proof size $O(\lambda \log n)$.

Distance spot checks. The key tool that we will be relying on is a protocol for convincing the verifier that two committed vectors are at most a constant distance apart. Concretely, let cm_1 and cm_2 be commitments to vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ respectively. We say that $\mathbf{v}_1$ and $\mathbf{v}_2$ are δ-far apart if they differ in at most δn locations. To show that $\mathbf{v}_1$ and $\mathbf{v}_2$ are at most δ-far apart, the prover and verifier engage in the following protocol:

1. The verifier uniformly samples an index $i \in [n]$ and sends it to the prover.
2. The prover responds with the purported i-th elements of $\mathbf{v}_1$ and $\mathbf{v}_2$, along with opening proofs.
3. The verifier accepts if these opening proofs are valid and the claimed elements are equal.

Clearly, if $\mathbf{v}_1$ and $\mathbf{v}_2$ are δ-far apart, then the verifier will reject with probability δ. For any constant $\delta > 0$, this soundness error can be made negligible with $\Theta(\lambda)$ parallel repetitions.

Linearity spot checks. This protocol easily generalizes to testing any kind of element-wise property, and in particular we can use it to check that $\mathbf{v}_3$ is δ-close to the "virtual vector" $\alpha\mathbf{v}_1 + \beta\mathbf{v}_2$. Unfortunately, we need to ensure that the two vectors are equal at all locations. Suppose a cheating prover commits to a vector that only differs from $\alpha\mathbf{v}_1 + \beta\mathbf{v}_2$ at a *single* location j. Detecting this would require $\Theta(\lambda n)$ repetitions (essentially opening the entire commitment), which violates the accumulation verifier's efficiency requirement.

2.1.1 Error-resilient linearity checks from codes

It seems that we are at an impasse: our spot check can only guarantee that two vectors are δ-close, but the accumulation scheme requires exact agreement. To overcome this issue, we need to make our accumulation scheme resilient to a constant δ-fraction of corruptions. We do so by relying on linear codes, and in particular those which enjoy good distance properties, such as the Reed–Solomon code [RSM60].

At a high level, we make the following changes to the accumulation scheme blueprint. Let C be a linear code, and let δ be a constant which is smaller than the unique decoding radius of C. The accumulator witness is a codeword $C(\mathbf{w})$, and the corresponding instance $\mathsf{acc}.\mathbb{x}$ is a commitment to $\mathsf{acc}.\mathbb{w}$. The accumulation verifier checks that the output accumulator is δ-close to a random linear combination of the input accumulators by running the linearity spot check. Later, the decider will check that $\mathsf{acc}_3.\mathbb{w}$ is the encoding of a good vector, and that $\mathsf{acc}_3.\mathbb{x}$ commits to $\mathsf{acc}_3.\mathbb{w}$.

Knowledge soundness. We would like our linearity spot check to satisfy the following knowledge soundness property. Suppose a (possibly malicious) prover outputs commitments cm_1, cm_2, and cm_3 which pass the check. Furthermore, suppose that cm_3 commits to a vector $\mathbf{v}_3$. Then an efficient extractor can find vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ such that (a) $\alpha\mathbf{v}_1 + \beta\mathbf{v}_2$ is δ-close to $\mathbf{v}_3$; and (b) cm_1 and cm_2 commit to $\mathbf{v}_1$ and $\mathbf{v}_2$. Notice that if we can extract vectors that satisfy (b), then our previous analysis of the spot check implies (a). Extraction turns out to be fairly straightforward: if we use Merkle commitments with a random oracle as the hash function, then we can find $\mathbf{v}_1$ and $\mathbf{v}_2$ by observing the prover's random oracle queries [Val08].[6]

Returning to accumulation, suppose that a (possibly malicious) prover outputs $\mathsf{acc}_1.\mathbb{x}, \mathsf{acc}_2.\mathbb{x}, \mathsf{acc}_3$ which pass the verifier's and decider's checks. Since the verifier runs the spot check, we can extract accumulator witnesses $\mathsf{acc}_1.\mathbb{w}$ and $\mathsf{acc}_2.\mathbb{w}$ such that $\alpha \cdot \mathsf{acc}_1.\mathbb{w} + \beta \cdot \mathsf{acc}_2.\mathbb{w}$ is δ-close to $\mathsf{acc}_3.\mathbb{w}$. Since the decider accepts $\mathsf{acc}_3.\mathbb{w}$, we know that $\mathsf{acc}_3.\mathbb{w}$ is a codeword $C(\mathbf{w}_3)$. Similarly, we need $\mathsf{acc}_1.\mathbb{w}$ and $\mathsf{acc}_2.\mathbb{w}$ to be codewords in order for the decider to accept $\mathsf{acc}_1.\mathbb{w}$ and $\mathsf{acc}_2.\mathbb{w}$. Unfortunately, this is simply not the case. For example, a cheating prover can always choose $\mathsf{acc}_1.\mathbb{w}$ which agrees with a codeword at all but one location, and this will almost certainly go undetected.

Can we still say something meaningful about the extracted witnesses? We argue that intuitively, since α and β are (possibly correlated) random scalars, with high probability $\mathsf{acc}_1.\mathbb{w}$ and $\mathsf{acc}_2.\mathbb{w}$ are themselves δ-close to codewords. Moreover, $\mathsf{acc}_1.\mathbb{w}$ and $\mathsf{acc}_2.\mathbb{w}$ decode to $\mathbf{w}_1$ and $\mathbf{w}_2$ such that $\alpha \cdot \mathbf{w}_1 + \beta \cdot \mathbf{w}_2 = \mathbf{w}_3$. This intuition can be formally proven using a suitable "proximity gap" result [BCIKS23], which exists for a variety of parameter regimes.

The upshot is that we extract accumulators acc_1 and acc_2 which are only accepted by a *relaxed* decider. Namely, given an accumulator acc, this decider checks that $\mathsf{acc}.\mathbb{x}$ commits to $\mathsf{acc}.\mathbb{w}$, and moreover that $\mathsf{acc}.\mathbb{w}$ is δ-close to the code.

Recursive extraction. The foregoing analysis suffices for a single step of accumulation. However, in order to construct PCD, we will have to recursively extract from old accumulators. It is straightforward to see that a recursively extracted accumulator is only guaranteed to be 2δ-close to the code, since we are extracting from an accumulator that may already be δ-far from the code. More generally, k steps of recursion will only guarantee accumulators that are $k\delta$-close to the code. We will see that once $k\delta$ is larger than the unique decoding radius, extraction is no longer meaningful. In particular, this leads to the following concrete attack: a cheating prover can start with a bad codeword (rejected by the decider) and, over the k accumulation steps, incrementally move it to a good codeword (accepted by the decider). This motivates our notion of "bounded-depth" accumulation, which is not captured by existing definitions [BCLMS21].

[6] To be precise, we must also consider a malicious prover that does not commit to a full vector; see Remark 3.5.

2.2 Defining bounded-depth accumulation

To describe our construction which only supports accumulation up to a certain (constant) depth, we introduce a new, relaxed knowledge soundness property; the key differences are highlighted in blue. We say that an accumulation scheme has *bounded-depth knowledge soundness* (with maximum depth d) if there exists a *family of deciders* $\{\mathrm{D}_s\}_{s=0}^{d}$, where D is equivalent to D_0, such that the following holds. If $\mathrm{D}_{s-1}(\mathsf{acc}') = 1$ and $\mathrm{V}(\mathbb{x}, \pi.\mathbb{x}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}'.\mathbb{x}) = 1$, then an efficient extractor can find $\pi.\mathbb{w}$ and $\mathsf{acc}.\mathbb{w}$ such that $\mathcal{V}(\mathbb{x}, \pi) = 1$ and $\mathrm{D}_s(\mathsf{acc}) = 1$.

This is a meaningful definition. In addition to generalizing standard knowledge soundness, which can be recovered by setting $d = \infty$ and using a single decider D, it captures our construction based on error-resilient linearity checks: D_s is the decider that only accepts if the accumulator is at most $s\delta$-far from the code, and in particular D_0 only accepts codewords. The depth bound d is the maximum number of recursive extractions that we can perform before $d\delta$ exceeds the unique decoding radius of the code.

2.3 Bounded-depth PCD from bounded-depth accumulation

Existing theorems that build PCD from accumulation [BCLMS21; BDFG21; KST22] do not immediately translate to the bounded-depth setting. To see why, let us recall a simplified version of the construction from [BCLMS21]. Suppose we have an accumulation scheme for a *non-interactive argument of knowledge* (NARK). Informally, a PCD proof for z_i, which consists of a NARK proof π_i and accumulator acc_i, certifies that $z_i = F^i(z_0)$, where z_0 is some initial value. We maintain the invariant that if π_i and acc_i are valid, then the computation is correct up to the i-th step.

- The PCD prover receives a proof (π_i, acc_i) for z_i, and wants to output a proof for z_{i+1}. First, it accumulates π_i and acc_i into a new accumulator acc_{i+1}, generating an accumulation proof pf_{i+1}. Next, it generates a NARK proof π_{i+1} for the following claim, expressed as a circuit R (see Figure 1): "$z_{i+1} = F(z_i)$, and there exists a NARK proof π_i, old accumulator acc_i, and accumulation proof pf_{i+1} which correctly accumulate into acc_{i+1}." The proof for z_{i+1} is $(\pi_{i+1}, \mathsf{acc}_{i+1})$.
- The PCD verifier checks a proof (π_i, acc_i) for z_i by running the NARK verifier on π_i and the decider on acc_i.

$R(\mathbb{x} = (z_{i+1}, \mathsf{acc}_{i+1}.\mathbb{x}), \mathbb{w} = (z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}, \mathsf{pf}_{i+1})) :$
1. Check that $z_{i+1} = F(z_i)$.
2. Set $\mathbb{x}_i = (z_i, \mathsf{acc}_i.\mathbb{x})$.
3. Check that $\mathrm{V}(\mathbb{x}_i, \pi_i, \mathsf{acc}_i.\mathbb{x}, \mathsf{acc}_{i+1}.\mathbb{x}, \mathsf{pf}_{i+1}) = 1$.

Figure 1: Recursion circuit for PCD.

Now suppose we replace the accumulation scheme with one that only has bounded-depth knowledge soundness. The construction remains the same, but we must provide a new soundness analysis.

PCD knowledge soundness. We need to construct an extractor which, given an accepting proof (π_T, acc_T) for z_T, extracts a sequence of values $z_0, \ldots, z_T$ such that $z_{i+1} = F(z_i)$ for all i. [BCLMS21] gives the following strategy, which *interleaves* the NARK extractor and accumulation extractor. Suppose we have z_{i+1}, π_{i+1}, and acc_{i+1}. First, we invoke the NARK extractor to obtain $(z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}, \mathsf{pf}_{i+1})$. Second, we invoke

the accumulation extractor to obtain $(\pi_i.\mathbb{w}, \mathsf{acc}_i.\mathbb{w})$. This gives us π_i and acc_i, and the process continues. We maintain the invariant that in the i-th step, π_i and acc_i are valid.

With bounded-depth accumulation, we need to maintain a slightly weaker invariant: in the i-th step, instead of requiring that acc_i is accepted by the *strict* decider D, we only require that it is accepted by the i-th *relaxed* decider D_i. This discussion only provides a high-level overview of the proof strategy, and only describes an *IVC* construction; we describe the full PCD construction that supports arbitrary (bounded-depth) computation graphs, along with a full soundness analysis, in Section 5.

Remark 2.1 (bounded-depth PCD suffices)**.** As presented, our PCD scheme supports up to d steps of computation, where d is the maximum depth of the accumulation scheme. We call this *bounded-depth PCD*. Since d will realistically be a small constant, this seems to be of limited use: most computations require more than a constant number of steps! Fortunately, even such a limited PCD scheme can be used to construct IVC for *any* polynomial-length computation [BCCT13]. The idea is for the PCD prover to receive multiple proofs in each step, yielding a *computation tree*. In particular, if we can accumulate m inputs and m accumulators in a single step, then our PCD scheme can support computation trees of size m^d. Setting $m = \lambda$ and $d = O(1)$ allows us to support polynomial-size computations.

Remark 2.2 (bounded-depth vs. constant-depth PCD)**.** Perhaps surprisingly, even with standard (unbounded) accumulation, [BCLMS21] is only able to construct *constant-depth PCD*.[7] This is because the size of the PCD extractor grows exponentially in the computation's depth, regardless of the accumulation scheme's knowledge soundness property. We remark that this limitation is largely theoretical: there is no known attack which exploits unbounded recursive proof composition. In contrast, the depth bound in bounded-depth PCD is not merely an artifact of the analysis: there exists a concrete attack that can be mounted against our construction when the depth exceeds d. This means that the tree-based strategy described in Remark 2.1 is *necessary* for real-world implementations, unlike in prior work.

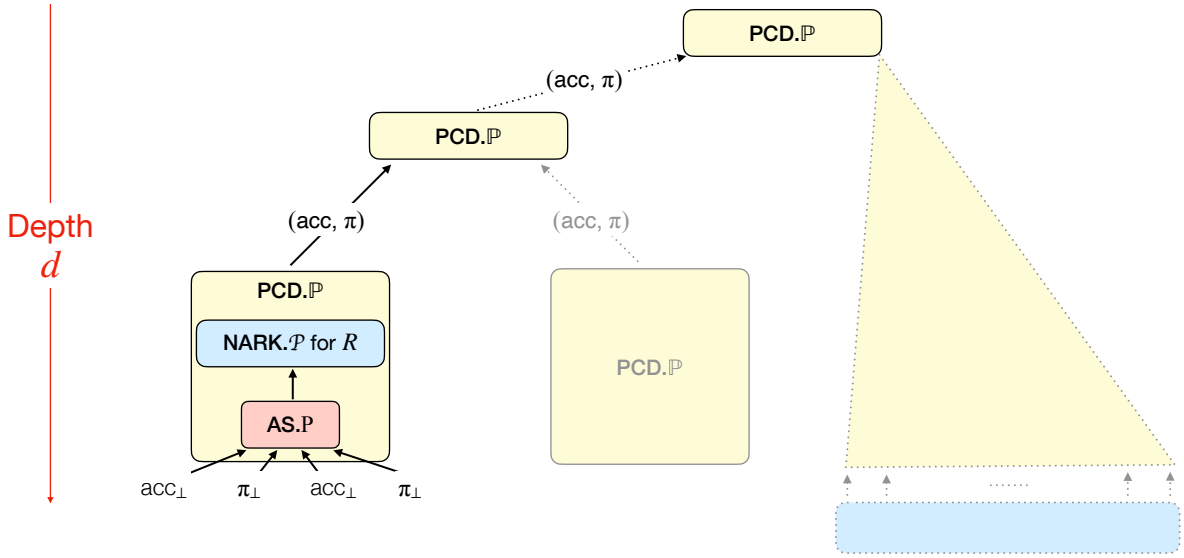

Figure 2: Construction of PCD from bounded-depth accumulation.

[7] This is slightly different from bounded-depth PCD, where the maximum depth must be fixed in advance.

2.4 Constructing bounded-depth accumulation

Our starting point is the ProtoStar and ProtoGalaxy accumulation schemes [BC23; EG23]. They support a general class of non-interactive arguments whose verifier consists of three steps: (a) compute Fiat–Shamir challenges; (b) open vector commitments; and (c) evaluate a polynomial over the instance, challenges, and openings. The key insight of ProtoStar [BC23] is that the accumulation verifier can cheaply perform the first step, while batching the remaining steps and deferring it to the decider.

Let us recall ProtoGalaxy's strategy [EG23]. An accumulator witness is a vector $w \in \mathbb{F}^n$, and an accumulator instance consists of a (homomorphic) vector commitment to w and an error term $e \in \mathbb{F}$. The decider accepts an accumulator if $p(w) = e$, where p is a polynomial in n variables. Now suppose the prover wants to accumulate m accumulators $\mathsf{acc}_1, \ldots, \mathsf{acc}_m$. Consider the univariate polynomial

$$p\left(\sum_{i=1}^{m} L_{i,H}(X) \cdot w_i\right) - \sum_{i=1}^{m} L_{i,H}(X) \cdot e_i,$$

where $L_{i,H}$ is the i-th Lagrange polynomial of some m-sized set $H \subset \mathbb{F}$. Notice that this polynomial is zero on H, and hence factors into $v_H(X) \cdot q(X)$, where $v_H(X)$ is the vanishing polynomial on H and q is some quotient polynomial. The prover sends q, and the verifier tests this equality at a random point $\alpha \in \mathbb{F}$. In particular, the prover constructs a new accumulator acc where the new vector is $w := \sum_i L_i(\alpha) \cdot w_i$ and the new error is $e := v(\alpha) \cdot q(\alpha) + \sum_i L_i(\alpha) \cdot e_i$. The verifier checks that acc was computed correctly by homomorphically computing the new vector commitment, and directly computing the new error term. The decider finishes the test by checking that $p(w) = e$.

Our construction. We follow the same strategy, except we replace homomorphic computations with error-resilient linearity checks. Concretely, we make the following changes. An accumulator witness is a codeword $f \in C$, and an accumulator instance consists of a vector commitment to f and an error term $e \in \mathbb{F}$. The decider accepts an accumulator if f is the encoding of a vector $w \in \mathbb{F}^n$ such that $p(w) = e$. Finally, as discussed in Section 2.1, the verifier uses a linearity check to ensure that the new accumulator acc is sufficiently consistent with the old accumulators $\mathsf{acc}_1, \ldots, \mathsf{acc}_m$.

Overall, our construction inherits many desirable properties from ProtoStar and ProtoGalaxy, including support for arbitrary arity $m = \mathrm{poly}(\lambda)$, which is crucial for constructing PCD (see Remark 2.1), and efficient support for degree d gates.

Security. Our construction naturally corresponds with an interactive oracle proof (IOP) [BCS16], where the codewords are now given as oracles instead of vector commitments. Indeed, we prove knowledge soundness of the accumulation scheme by proving soundness of the underlying IOP, and then applying the BCS transformation [BCS16] (with some technical subtleties).

Remark 2.3 (choosing the linear code)**.** A key parameter in our construction is the linear code. We use Reed–Solomon codes because they display a proximity gap when the coefficients are Lagrange evaluations $L_1(\alpha), \ldots, L_m(\alpha)$. However, our construction can be adapted to support arbitrary linear codes which display any proximity gap (e.g. uniformly random coefficients). See Section 6.4 for details.

Efficiency. The cost of the accumulation verifier is dominated by that of the linearity checker. Recall that to achieve negligible knowledge soundness error at depth d, the latter checks that two code words are $d \cdot \delta$ close via $k = O(d \cdot \lambda)$ spot checks, where δ is such that $d \cdot \delta$ is less than the unique decoding radius of the code. Overall, when instantiated with the Merkle tree-based vector commitment, the accumulation verifier checks $O(d \cdot \lambda)$ Merkle tree paths.

When applying this construction to the PCD scheme in Section 2.3, the latter cost becomes the *recursive overhead* of the PCD prover. As noted in Table 1, this cost is asymptotically better than the $O(\log n \cdot \lambda)$ cost of prior SNARK-based PCD schemes [COS20].

A keen reader may notice that a disadvantage of our construction is that recursive overhead scales with the depth of the PCD computation graph. We now present several optimizations that reduce the depth of the PCD tree and significantly improve the efficiency of the resulting PCD scheme in practice.

2.5 Optimization: Batch commitments

Recall from Remarks 2.1 and 2.2 that for our PCD construction, reducing the depth of the computation graph is essential for achieving provable security guarantees, and the standard depth-reduction technique for the case of IVC [BCCT13] works by constructing a *PCD tree* whose leaves comprise the actual computation being performed. To achieve constant computation depth, Bitansky et al. [BCCT13] set the arity m of this tree to be super-constant (i.e., $m = \lambda$). The per-node recursive overhead of our PCD construction in this setting is the cost of performing linearity checks on m codewords of size n, which costs $O(m \cdot \lambda \cdot \log n)$ hashes when using Merkle trees. We now describe an optimization that reduces this to just $O(\lambda(\log n + m))$ hashes.

Recall that the linearity checker opens all m codewords at the same locations. This means that for each spot-check, each of the m Merkle trees is opened at the same leaf. We take advantage of this repetitive structure by committing to all m codewords using a *single* Merkle tree whose i-th leaf is the concatenation of the i-th symbols of the codewords. Each spot-check now requires opening only a single Merkle tree path (and checking the leaf hash), leading to a cost of $O(\lambda(\log n + m))$ hashes for $O(\lambda)$ spot checks, as required.

However, this modification comes at a cost: it requires us to commit to all codewords together at the same time. While this is straightforward at the leaf layer, it gets more complex at higher layers. For example, even committing to a new accumulator now requires waiting for the batch of "sibling" new accumulators, which in turn means that we must wait for m accumulations (each of size m) to complete before we can compute the commitments to the m new accumulators. Overall, across the entire tree, this requires the PCD prover to maintain a state of $O(m^2)$ "pending" accumulators. The resulting PCD scheme is illustrated in Figure 3, and we provide more details in Section 7.1.

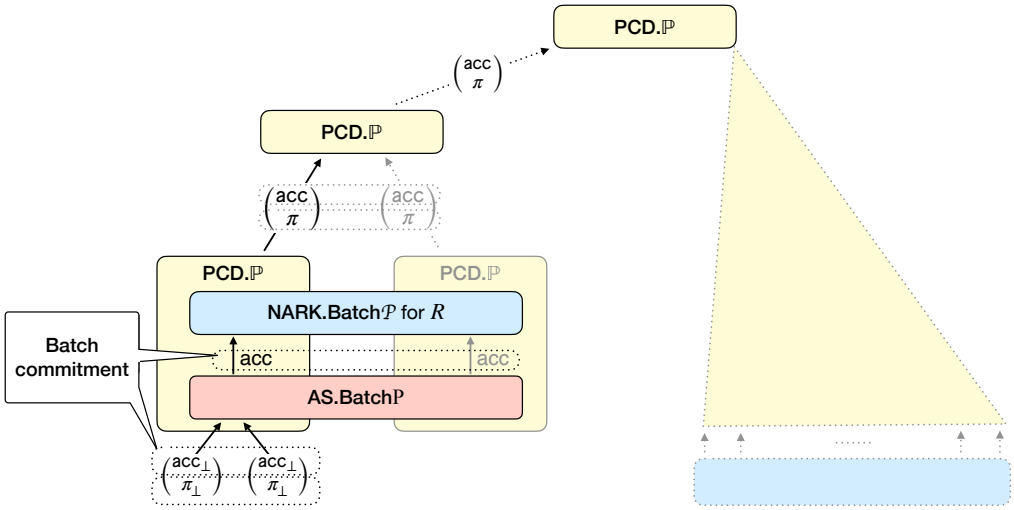

Figure 3: PCD with batch commitments.

2.6 Optimization: Low-overhead IVC from accumulation

We give a generic optimization which improves on the PCD-to-IVC compiler of Bitansky et al. [Val08; BCCT13]. They construct a (polynomial-length) IVC scheme for a function F by using a (constant-depth) PCD scheme for a related function F'. In particular, F' computes F (at leaf nodes) and performs consistency checks (at internal nodes). This is wasteful: even though we do not need to prove anything about F at internal nodes, the PCD prover still generates a proof for F' (which is dominated by F).

We improve this compiler by constructing an IVC scheme with minimal overhead. The core idea is that we first construct a tree of accumulators for F, i.e. a tree the leaf nodes are proofs for F, and each parent accumulates its children. Then, we construct a PCD tree which proves that the accumulation tree was constructed correctly. When the PCD scheme is instantiated with our accumulation-based construction, the PCD circuit now checks *two* accumulation verifiers: one that checks the correctness of the accumulation tree, and one that helps check the correctness of the PCD tree.

Accumulating separately means that we no longer have to generate NARK proofs for F at internal nodes. Additionally, because we only need to show that internal nodes of the accumulation tree were constructed correctly, our PCD tree has one fewer layer than before. This further reduces cost, in particular for higher arities (as m grows, the leaf layer of a tree contains a higher fraction of nodes). The resulting IVC scheme is illustrated in Figure 4, and we provide more details in Section 7.2.

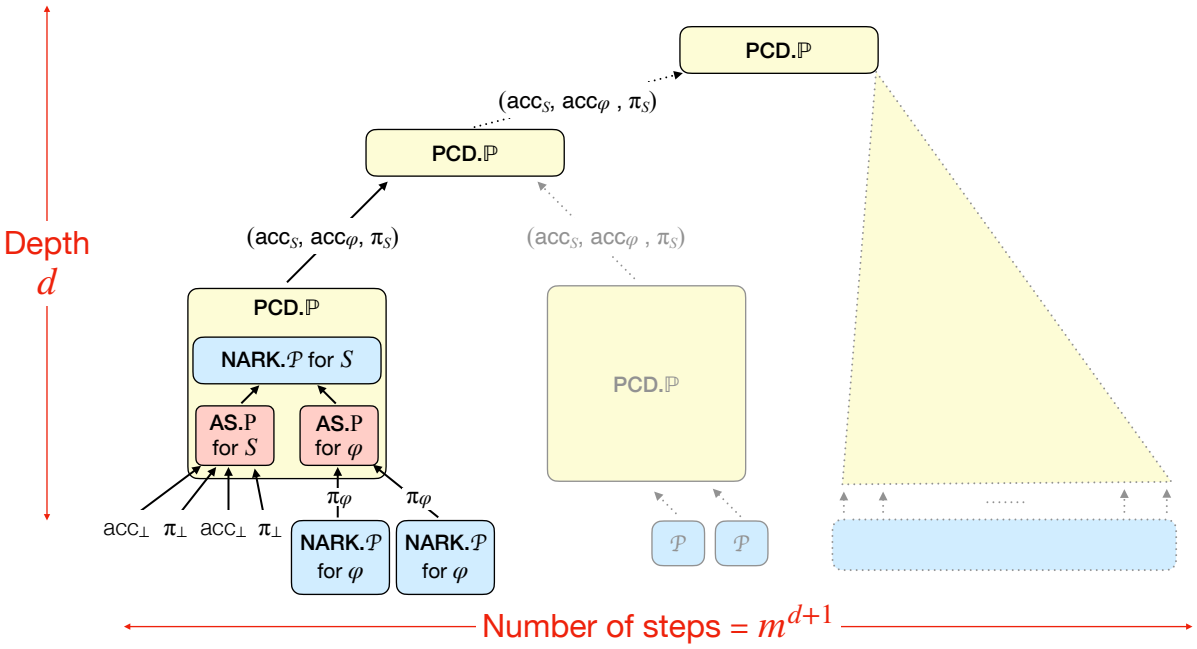

Figure 4: PCD-to-IVC compiler that checks the correctness of an accumulation tree.

2.7 Optimization: PCD composition

The recursive overhead of our PCD scheme grows linearly with the maximum supported depth. This is contrast with SNARK-based PCD schemes like Fractal [COS20], which do not suffer from such an efficiency loss. However, these schemes pay a higher per-step cost anyway, and are thus asymptotically less efficient than our PCD scheme for low recursion depths.

We provide a generic optimization to combine SNARK-based PCD schemes with our PCD scheme to achieve a scheme that is achieves better efficiency than either scheme alone. The core idea is to first use our accumulation-based PCD up to some depth d_1, and then prove the PCD verifier for the latter with a SNARK-based PCD scheme on top. When invoked with tree PCD, this means that the SNARK-based PCD scheme is invoked only every m^{d_1} steps. By choosing d_1 appropriately, the resulting scheme achieves better efficiency than either scheme alone. Furthermore, the resulting scheme supports *arbitrary* constant depth, as opposed to our accumulation-based PCD scheme, which only supports an *a priori* fixed depth. See Figure 5 for an illustration of this idea, and Section 7.3 for more details.

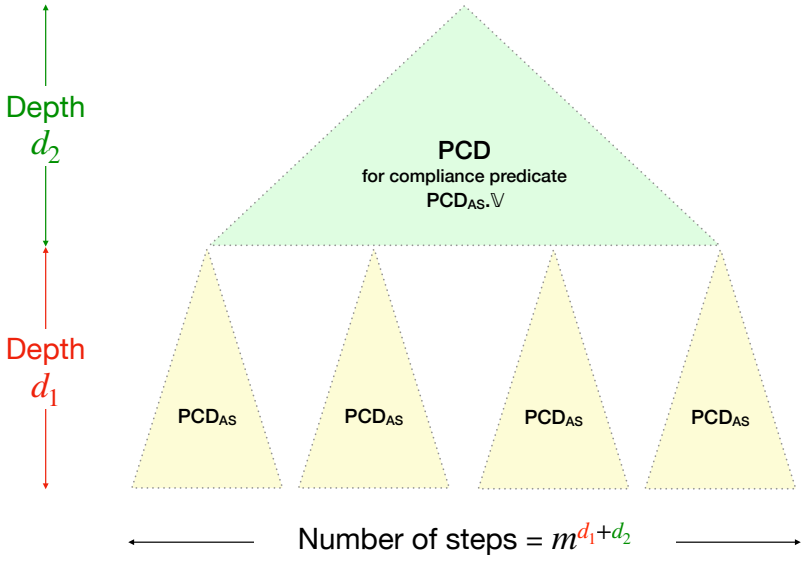

Figure 5: PCD composition.

3 Preliminaries

Indexed relations. An *indexed relation* $\mathscr{R}$ is a set of triples $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ where $\mathtt{i}$ is the index, $\mathtt{x}$ is the instance, and $\mathtt{w}$ is the witness; the corresponding *indexed language* $\mathscr{L}(\mathscr{R})$ is the set of pairs $(\mathtt{i}, \mathtt{x})$ for which there exists a witness $\mathtt{w}$ such that $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathscr{R}$. For example, the indexed relation of satisfiable boolean circuits consists of triples where $\mathtt{i}$ is the description of a boolean circuit, $\mathtt{x}$ is a partial assignment to its input wires, and $\mathtt{w}$ is an assignment to the remaining wires that makes the boolean circuit output 0.

Security parameters. For simplicity of notation, we assume that all public parameters have length at least λ, so that algorithms which receive such parameters can run in time $\mathrm{poly}(\lambda)$.

Random oracles. We denote by $\mathcal{U}(\lambda)$ the set of all functions that map $\{0,1\}^*$ to $\{0,1\}^\lambda$. A *random oracle* with security parameter λ is a function $\rho\colon \{0,1\}^* \to \{0,1\}^\lambda$ sampled uniformly at random from $\mathcal{U}(\lambda)$. In our random oracle definitions, we assume that all algorithms (except generators and setup algorithms), adversaries, and extractors have access to the random oracle.

Adversaries. All of the definitions in this paper should be taken to refer to non-uniform adversaries. An adversary (or extractor) running in *expected polynomial time* is then a Turing machine provided with a *polynomial-size* non-uniform advice string and access to an infinite random tape, whose expected running time for all choices of advice is polynomial. We sometimes write $(\mathsf{o}; r) \leftarrow A(x)$ when A is an expected polynomial-time algorithm, where o is the output of A and r is the randomness used by A (i.e. up to the rightmost position of the head on the randomness tape).

Hamming distance. Let Σ be an alphabet, typically $\mathbb{F} \cup \{\perp\}$. The relative Hamming distance between two vectors $f, g \in \Sigma^n$, denoted $\Delta(f, g)$, is the number of locations where f and g disagree, divided by n. The distance between a vector $f \in \Sigma^n$ and a subset $S \subset \Sigma^n$, denoted $\Delta(f, S)$, is equal to $\min_{g \in S} \Delta(f, g)$.

Polynomials. For any field $\mathbb{F}$ and subset $H = \{a_1, \ldots, a_k\} \subset \mathbb{F}$, let $L_{i,H}$ denote the i-th Lagrange polynomial, i.e. the unique polynomial of degree less than k such that $L_{i,H}(a_i) = 1$ and $L_{i,H}(a_j) = 0$ for all $j \neq i$. Let v_H denote the vanishing polynomial on H, i.e. $v_H(X) = \prod_{i=1}^{m}(X - a_i)$. When clear from context, we omit the set H.

3.1 Non-interactive arguments of knowledge

In the standard definition of a non-interactive argument of knowledge (NARK), completeness and knowledge soundness hold for the same verifier. We introduce a "relaxed" verifier, for which only knowledge soundness must hold. In other words, an extractor must be able to extract a witness for any proof accepted by the relaxed verifier, but completeness only needs to hold for the original verifier. Concretely, a tuple of algorithms $\mathsf{ARG} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a *non-interactive argument of knowledge* in the random oracle model for an indexed relation family $\{\mathscr{R}_{\mathsf{pp}}\}_{\mathsf{pp}}$ if the following properties hold.

Completeness. For every (unbounded) adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c|r} (\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathscr{R}_{\mathsf{pp}} & \rho \leftarrow \mathcal{U}(\lambda) \\ \Downarrow & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ \mathcal{V}(\mathsf{pp}, \mathtt{i}, \mathtt{x}, \pi) = 1 & (\mathtt{i}, \mathtt{x}, \mathtt{w}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ & \pi \leftarrow \mathcal{P}(\mathsf{pp}, \mathtt{i}, \mathtt{x}, \mathtt{w}) \end{array}\right] = 1.$$

Knowledge soundness. We say that ARG has knowledge soundness for a *relaxed verifier* $\hat{\mathcal{V}}$, i.e. $\mathcal{V}$ accepting implies $\hat{\mathcal{V}}$ accepting, if there exists an expected polynomial time extractor $\mathcal{E}$ such that for every expected

polynomial time adversary $\tilde{\mathcal{P}}$, and auxiliary input distribution $\mathcal{D}$, the following probability is negligibly close to 1:

$$\Pr\left[\begin{array}{c|r} \hat{\mathcal{V}}(\mathsf{pp}, \mathbb{i}, \mathbb{x}, \pi) = 1 & \rho \leftarrow \mathcal{U}(\lambda) \\ \Downarrow & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathscr{R}_{\mathsf{pp}} & \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ & (\mathbb{i}, \mathbb{x}, \pi; r) \leftarrow \tilde{\mathcal{P}}(\mathsf{pp}, \mathsf{ai}) \\ & \mathbb{w} \leftarrow \mathcal{E}^{\tilde{\mathcal{P}}}(\mathsf{pp}, \mathsf{ai}, r) \end{array}\right]$$

Remark 3.1. Clearly, any standard NARK satisfies our definition by setting the relaxed verifier to be the original verifier. However, our accumulation construction will require a non-trivial relaxation.

Multi-instance extraction. As in [BCLMS21], we also define a weaker notion of knowledge soundness which is implied by the earlier definition. For every expected polynomial time adversary $\tilde{\mathcal{P}}$ and auxiliary input distribution $\mathcal{D}$, there exists an expected polynomial time extractor $\mathcal{E}_{\tilde{\mathcal{P}}}$ such that for every set Z,

$$\begin{aligned} &\Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{ai}, \vec{\mathbb{i}}, \vec{\mathbb{x}}, \mathsf{ao}) \in Z & \rho \leftarrow \mathcal{U}(\lambda) \\ \forall j \in [\ell], (\mathbb{i}_j, \mathbb{x}_j, \mathbb{w}_j) \in \mathscr{R}_{\mathsf{pp}} & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ & \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ & (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\mathbb{w}}, \mathsf{ao}) \leftarrow \mathcal{E}_{\tilde{\mathcal{P}}}(\mathsf{pp}, \mathsf{ai}) \end{array}\right] \\ \geq &\Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{ai}, \vec{\mathbb{i}}, \vec{\mathbb{x}}, \mathsf{ao}) \in Z & \rho \leftarrow \mathcal{U}(\lambda) \\ \forall j \in [\ell], \mathcal{V}(\mathbb{i}_j, \mathbb{x}_j, \pi_j) = 1 & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ & \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ & (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{ao}) \leftarrow \tilde{\mathcal{P}}(\mathsf{pp}, \mathsf{ai}) \end{array}\right] - \mathrm{negl}(\lambda). \end{aligned}$$

3.2 Proof-carrying data

Before we define proof-carrying data (PCD), we recall some terminology. A *transcript* T is a directed acyclic graph where each vertex $u \in V(\mathsf{T})$ is labeled by local data $z^{(u)}_{\mathsf{loc}}$ and each edge $e \in E(\mathsf{T})$ is labeled by a message $z^{(e)} \neq \bot$. The *output* of a transcript T, denoted $\mathsf{o}(\mathsf{T})$, is $z^{(e)}$ where $e = (u, v)$ is the lexicographically-first edge such that v is a sink.

Compliance. A vertex $u \in V(\mathsf{T})$ is φ-compliant for some *compliance predicate* $\varphi : \{0,1\}^* \to \{0,1\}$ if for all outgoing edges $e = (u, v) \in E(\mathsf{T}) \in E(\mathsf{T})$, one of the following holds. If u has no incoming edges, $\varphi(z^{(e)}, z^{(u)}_{\mathsf{loc}}, \bot, \ldots, \bot)$ accepts. If u has incoming edges $e_1, \ldots, e_m$, $\varphi(z^{(e)}, z^{(u)}_{\mathsf{loc}}, z^{(e_1)}, \ldots, z^{(e_m)})$ accepts. We say that a transcript is φ-compliant if all of its vertices are φ-compliant.

Depth. The depth of a transcript T is the largest number of nodes on a source-to-sink path in T, minus two (to ignore the source and sink). The depth of a compliance predicate φ, denoted $\mathsf{d}(\varphi)$, is defined to be the maximum depth over *all* φ-compliant transcripts.

A tuple of algorithms $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ is a *proof-carrying data scheme* for a class of compliance predicates F if the following properties hold.

Completeness. For every (unbounded) adversary $\mathcal{A}$,

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

Knowledge soundness. For every expected polynomial time adversary $\tilde{\mathbb{P}}$ there exists an expected polynomial-time extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ such that for every auxiliary input distribution $\mathcal{D}$ and set Z,

$$\Pr\left[\begin{array}{c|r} \varphi \in \mathsf{F} & \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (\mathbb{pp}, \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}), \mathsf{ao}) \in Z & \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ \mathsf{T} \text{ is } \varphi\text{-compliant} & (\varphi, \mathsf{T}, \mathsf{ao}) \leftarrow \mathbb{E}_{\tilde{\mathbb{P}}}(\mathbb{pp}, \mathsf{ai}) \end{array}\right]$$
$$\geq \Pr\left[\begin{array}{c|r} \varphi \in \mathsf{F} & \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (\mathbb{pp}, \mathsf{ai}, \varphi, \mathsf{o}, \mathsf{ao}) \in Z & \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ \mathbb{V}(\mathsf{ivk}, \mathsf{o}, \pi) = 1 & (\varphi, \mathsf{o}, \pi, \mathsf{ao}) \leftarrow \tilde{\mathbb{P}}(\mathbb{pp}, \mathsf{ai}) \\ & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathbb{pp}, \varphi) \end{array}\right] - \mathrm{negl}(\lambda) \ .$$

Efficiency. The generator $\mathbb{G}$, prover $\mathbb{P}$, indexer $\mathbb{I}$, and verifier $\mathbb{V}$ run in polynomial time. A proof π has size $\mathrm{poly}(\lambda, |\varphi|)$; in particular, it is not permitted to grow with each application of $\mathbb{P}$.

Remark 3.2. In this paper, we are interested in PCD for *bounded-depth* compliance predicates. Concretely, pick an arbitrary constant $d \in \mathbb{N}$. We construct PCD for a class $\{\varphi : \mathsf{d}(\varphi) < d\}$. In contrast, prior work construct PCD for the class of *constant-depth* compliance predicates $\{\varphi : \mathsf{d}(\varphi) = O(1)\}$. Intuitively, this is a change in the order of quantifiers; instead of saying "there exists a PCD scheme for predicates of arbitrary (constant) depth," we say "for any c, there exists a PCD scheme for predicates of depth at most c." In general, compliance predicates can be engineered to have bounded depth, e.g. by incrementing a counter in the transcript.

3.3 Instantiating the random oracle

As in prior work [BCMS20; BCLMS21], almost all definitions and constructions in this paper are in the *random oracle model*. The sole exception is our PCD definition, as we do not know how to build secure PCD schemes in the random oracle model. Instead, we show how to construct PCD in the standard (CRS) model, assuming we have a accumulation-compatible NARK (for circuit satisfiability) in the standard (CRS) model. This can be heuristically realized from our constructions by instantiating the random oracle as a hash function.

3.4 Reed–Solomon codes

A linear code of blocklength n over a field $\mathbb{F}$ is a linear subspace $C \subset \mathbb{F}^n$. The dimension of the code is the dimension of the subspace. The rate of the code is $R = \dim(C)/n$. Given an evaluation domain $L \subset \mathbb{F}$ and degree bound $k < |L|$, the Reed-Solomon code $\mathsf{RS}[\mathbb{F}, L, k]$ is defined to be the set of all evaluations over L of polynomials of degree at most k:

$$\mathsf{RS}[\mathbb{F}, L, k] := \left\{ \hat{f}(L) : \hat{f} \in \mathbb{F}[X], \deg(\hat{f}) \leq k \right\}$$

This is a linear code with blocklength $n = |L|$ and dimension $k + 1$. Given a codeword f, we let $\hat{f}$ denote the corresponding polynomial. We will often interpret $\hat{f}$ as a coefficient vector in $\mathbb{F}^{k+1}$, and vice versa.

Decoding. Let $C = \mathsf{RS}[\mathbb{F}, L, k]$ be a Reed-Solomon code with rate $R = (k + 1)/n$. There exists a polynomial-time decoding algorithm which, given a vector $f \in (\mathbb{F} \cup \{\perp\})^n, \Delta(f, C) \leq (1 - R)/2$, finds the unique codeword $g \in C$ such that $\Delta(f, g) \leq (1 - R)/2$. If f does not satisfy the closeness condition, the algorithm rejects. We refer to $(1 - R)/2$ as the unique decoding radius of the code.

Proximity gaps. Reed-Solomon codes enjoy a number of so-called proximity gap results. Informally, these say the following. Suppose you have vectors $f_1, \ldots, f_m \in \mathbb{F}^n$, of which at least one is δ-far from the code, i.e. $\Delta(f_i, C) \geq \delta$. Then with high probability, a random linear combination of these vectors will also be δ-far from the code. The exact kind of random linear combination is somewhat flexible. Besides uniformly random coefficients, [BCIKS23] show that one can sample a single field element $\alpha \leftarrow \mathbb{F}$ and set the coefficients to be the monomial evaluations $1, \alpha, \alpha^2, \ldots, \alpha^{m-1}$. In our accumulation scheme, we set the coefficients to be the evaluations $L_1(\alpha), \ldots, L_m(\alpha)$ of the Lagrange polynomials of some set of size m. Although the corresponding proximity gap result is not proven in [BCIKS23], it is a straightforward generalization as illustrated in Theorem 3.3.

Theorem 3.3. *Let $C = \mathsf{RS}[\mathbb{F}, L, k]$ be a Reed–Solomon code with rate R and blocklength n, and suppose $\delta \leq (1-R)/2$. Let $L_0, \ldots, L_\ell$ be the Lagrange polynomials for an arbitrary set of $\ell + 1$ points in $\mathbb{F}$. Let $u_0, \ldots, u_\ell : L \to \mathbb{F}$ be functions. Define the set*

$$S = \left\{ z \in \mathbb{F} : \Delta\left(\textstyle\sum_{i=0}^{\ell} L_i(z) \cdot u_i, C\right) \leq \delta \right\}$$

and suppose $|S| > \ell \cdot n$. Then for all $z \in \mathbb{F}$ we have

$$\Delta\left(\textstyle\sum_{i=0}^{\ell} L_i(z) \cdot u_i, C\right) \leq \delta,$$

and furthermore there exist $v_0, \ldots, v_\ell \in C$ such that for all $z \in \mathbb{F}$,

$$\Delta\left(\textstyle\sum_{i=0}^{\ell} L_i(z) \cdot u_i, \sum_{i=0}^{\ell} L_i(z) \cdot v_i\right) \leq \delta,$$

and in fact $|\{x \in L : (u_0(x), \ldots, u_\ell(x)) \neq (v_0(x), \ldots, v_\ell(x))\}| \leq \delta |L|$.

Proof. This is a direct adaption of Theorem 6.1 from [BCIKS23]. The only difference between the statement of Theorem 3.3 and theirs is the choice of parameterized curve. In particular, their theorem statement is for the curve $u_0 + zu_1 + \cdots + z^\ell u_\ell$, whereas ours is for the curve $L_0(z) \cdot u_0 + L_1(z) \cdot u_1 + \cdots + L_\ell(z) \cdot u_\ell$. Their proof essentially goes through, since it only depends on the degree of x and z; these are identical in both curves. The only change is how we interpret the final polynomial $P(X, Z)$, which recovers the candidate codewords $v_0, \ldots, v_\ell$. In particular, instead of writing $P(X, Z)$ as $v_0(X) + Zv_1(X) + \cdots + Z^\ell v_\ell(X)$, we use a change of basis: $P(X, Z) = L_0(Z) \cdot v_0(X) + L_1(Z) \cdot v_1(X) + \cdots + L_\ell(Z) \cdot v_\ell(X)$. □

The following lemma is immediately implied by Theorem 3.3.

Lemma 3.4. *Let $C = \mathsf{RS}[\mathbb{F}, L, k]$ be a Reed-Solomon code with rate R and blocklength n, and suppose $\delta \leq (1-R)/2$. Let $L_1, \ldots, L_m$ be the Lagrange polynomials for an arbitrary set of m points in $\mathbb{F}$. Consider arbitrary vectors $f_1, \ldots, f_m \in \mathbb{F}^n$. If*

$$\Pr_{\alpha \leftarrow \mathbb{F}}\left[\Delta\left(\sum_{i=1}^{m} L_i(\alpha) \cdot f_i, C\right) \geq \delta\right] > \frac{n(m-1)}{\mathbb{F}},$$

then there exists a subdomain $L' \subset L$ and codewords $g_1, \ldots, g_m \in C$ such that the following holds. First, $|L'|/|L| \geq 1 - \delta$. Second, for all i, f_i and g_i agree on L'.

3.5 Vector commitments

An *extractable vector commitment scheme* in the random oracle model is a tuple of algorithms $\mathsf{VC} = (\mathsf{VC.Setup}, \mathsf{VC.Commit}, \mathsf{VC.Open}, \mathsf{VC.Answer})$ with the following syntax and properties.

- $\mathsf{VC.Setup}(1^\lambda, \Sigma) \to \mathsf{vp}$: On input a security parameter λ and alphabet Σ, outputs public parameters vp which allow for committing to arbitrarily-length vectors over Σ.
- $\mathsf{VC.Commit}(\mathsf{vp}, m) \to (\mathsf{cm}, \mathsf{aux})$: On input public parameters vp, message $m \in \Sigma^\ell$, outputs a commitment cm and auxiliary data aux.
- $\mathsf{VC.Open}(\mathsf{vp}, \mathsf{aux}, \mathcal{Q}) \to \mathsf{op}$: On input public parameters vp, auxiliary data aux, and a query set $\mathcal{Q} \subseteq [\ell]$, outputs a partial opening op for the commitment.
- $\mathsf{VC.Answer}(\mathsf{vp}, \mathsf{cm}, \mathsf{op}, \mathcal{Q}) \to \mathsf{ans}$: On input public parameters vp, commitment cm, partial opening op, and query set $\mathcal{Q} \subset [\ell]$, outputs an answer $\mathsf{ans} : \mathcal{Q} \to \Sigma \cup \{\perp\}$, which can also be interpreted as a vector of length $|\mathcal{Q}|$. If $\mathsf{ans}[i] = \perp$ for some $i \in \mathcal{Q}$, this implies that op does not contain an opening for that location.

Completeness. For every alphabet Σ, message length ℓ, message $m \in \Sigma^\ell$, and query set $\mathcal{Q} \subset [\ell]$,

$$\Pr\left[\, \mathsf{VC.Answer}(\mathsf{vp}, \mathsf{cm}, \mathsf{op}, \mathcal{Q}) = m[\mathcal{Q}] \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{vp} \leftarrow \mathsf{VC.Setup}(1^\lambda) \\ (\mathsf{cm}, \mathsf{aux}) \leftarrow \mathsf{VC.Commit}(\mathsf{vp}, m) \\ \mathsf{op} \leftarrow \mathsf{VC.Open}(\mathsf{vp}, \mathsf{aux}, \mathcal{Q}) \end{array} \right] = 1.$$

Extractability. There exists a polynomial time extractor E such that for every alphabet Σ, message length ℓ, polynomial time adversaries $\mathcal{A}, \mathcal{B}$, and auxiliary input distribution $\mathcal{D}$, the following is negligibly close to 1:

$$\Pr\left[\begin{array}{c} \mathcal{Q} \subset [\ell] \\ \Downarrow \\ \forall i \in \mathcal{Q}, \mathsf{ans}'[i] \in \{\mathsf{ans}[i], \perp\} \end{array} \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{vp} \leftarrow \mathsf{VC.Setup}(1^\lambda, \Sigma) \\ \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ (\mathsf{cm}; r) \leftarrow \mathcal{A}(\mathsf{vp}, \mathsf{ai}) \\ \mathsf{op} \leftarrow \mathrm{E}^{\mathcal{A}}(\mathsf{vp}, \mathsf{ai}, r) \\ (\mathsf{op}', \mathcal{Q}) \leftarrow \mathcal{B}(\mathsf{vp}, \mathsf{ai}) \\ \mathsf{ans} \leftarrow \mathsf{VC.Answer}(\mathsf{vp}, \mathsf{cm}, \mathsf{op}, \mathcal{Q}) \\ \mathsf{ans}' \leftarrow \mathsf{VC.Answer}(\mathsf{vp}, \mathsf{cm}, \mathsf{op}', \mathcal{Q}) \end{array} \right]$$

The extractor implicitly receives Σ and ℓ as input.

Remark 3.5. Informally, extractability says that the extractor outputs a "maximal" opening, in the sense that no adversary can open to a value outside or inconsistent with the extractor's opening. This subsumes the standard *position binding* property of vector commitments.

Remark 3.6. We assume that the expected vector length ℓ is implicitly provided to $\mathsf{VC.Answer}$. In our constructions, we assume that $\mathsf{VC.Answer}$ accepts auxiliary data aux and interprets it as a full opening to the vector; this can always be done by first calling $\mathsf{VC.Open}(\mathsf{vp}, \mathsf{aux}, [\ell])$. In this case, we omit the query set $\mathcal{Q}$.

Extractable vector commitments can be realized with Merkle trees which use the random oracle as a hash function. Valiant's extractor [Val08] satisfies the extractability property.

4 Bounded-depth accumulation

Let $\mathsf{ARG} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ be a non-interactive argument with knowledge soundness for a relaxed verifier $\hat{\mathcal{V}}$. Suppose proofs can be split into two parts, i.e. $\pi = (\pi.\mathbb{x}, \pi.\mathbb{w})$. Let qx denote $(\mathbb{x}, \pi.\mathbb{x})$, where $\mathbb{x}$ is an instance of the relation; call this the *verifier input instance*. Let qw denote $\pi.\mathbb{w}$; call this the *verifier input witness*. We write $\mathcal{V}(\mathsf{pp}, \mathbb{i}, \mathsf{qx}_i, \mathsf{qw}_i)$ as shorthand for the verifier running on $\mathbb{x}$ and π. We write acc as shorthand for the tuple $(\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$. An *accumulation scheme* in the random oracle model for ARG is a tuple of algorithms $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ with the following syntax and properties.

- $\mathrm{G}(1^\lambda) \to \mathsf{pp}_{\mathsf{AS}}$: On input a security parameter λ, the generator G samples and outputs accumulation parameters $\mathsf{pp}_{\mathsf{AS}}$.
- $\mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathbb{i}) \to \mathsf{pp}_{\mathsf{AS}}$: On input accumulation parameters $\mathsf{pp}_{\mathsf{AS}}$ and argument parameters pp, the indexer I deterministically computes and outputs a proving key apk, verification key avk, and decision key dk.
- $\mathrm{P}(\mathsf{apk}, [\mathsf{qx}_i, \mathsf{qw}_i]_{i=1}^{m_1}, [\mathsf{acc}_i]_{i=1}^{m_2}) \to (\mathsf{acc}, \mathsf{pf})$: On input the proving key apk, verifier inputs $[\mathsf{qx}_i, \mathsf{qw}_i]_{i=1}^{m_1}$, and accumulators $[\mathsf{acc}_i]_{i=1}^{m_2}$, the accumulation prover P outputs a new accumulator acc and proof pf. Here, m_1 and m_2 are fixed arities which may be functions of λ.
- $\mathrm{V}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}.\mathbb{x}_i]_{i=1}^{m_2}, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) \to \{0,1\}$: On input the verifying key avk, verifier input instances $[\mathsf{qx}_i]_{i=1}^{m_1}$, accumulator instances $[\mathsf{acc}.\mathbb{x}_i]_{i=1}^{m_2}$, new accumulator instance $\mathsf{acc}.\mathbb{x}$, and proof pf, the accumulation verifier V outputs a bit indicating whether $\mathsf{acc}.\mathbb{x}$ correctly accumulates the other instances.
- $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) \to \{0,1\}$: On input the decision key dk and accumulator acc, the decider outputs a bit indicating whether acc is a valid accumulator.

Completeness. For every (unbounded) adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} \forall i \in [m_1], \mathcal{V}(\mathsf{pp}, \mathbb{i}, \mathsf{qx}_i, \mathsf{qw}_i) = 1 \\ \forall i \in [m_2], \mathrm{D}(\mathsf{dk}, \mathsf{acc}_i) = 1 \\ \Downarrow \\ \mathrm{V}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2}, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1 \\ \mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 1 \end{array} \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{G}(1^\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (\mathbb{i}, [\mathsf{qx}_i, \mathsf{qw}_i]_{i=1}^{m_1}, [\mathsf{acc}_i]_{i=1}^{m_2}) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathbb{i}) \\ (\mathsf{acc}, \mathsf{pf}) \leftarrow \mathrm{P}(\mathsf{apk}, [\mathsf{qx}_i, \mathsf{qw}_i]_{i=1}^{m_1}, [\mathsf{acc}_i]_{i=1}^{m_2}) \end{array}\right] = 1.$$

To bootstrap accumulation, we also assume the prover can efficiently construct a dummy accumulator and proof, denoted $\mathsf{acc} = \mathrm{P}(\mathsf{apk}, \perp)$, which the decider accepts.

Knowledge soundness. We say that AS has bounded-depth knowledge soundness (with maximum depth d_s) if there exists a *family of deciders* $\{\mathrm{D}_s\}_{s=0}^{d_s}$, where D is equivalent to D_0, such that the following holds. There exists an expected polynomial time extractor E such that for every depth parameter $s \in [d_s]$, expected polynomial time adversary $\tilde{\mathrm{P}}$, and auxiliary input distribution $\mathcal{D}$, the following probability is negligibly close to 1:

$$\Pr\left[\begin{array}{c} \mathrm{V}^\rho(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2}, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1 \\ \mathrm{D}_{s-1}(\mathsf{dk}, \mathsf{acc}) = 1 \\ \Downarrow \\ \forall i \in [m_1], \hat{\mathcal{V}}(\mathsf{pp}, \mathbb{i}, \mathsf{qx}_i, \mathsf{qw}_i) = 1 \\ \forall i \in [m_2], \mathrm{D}_s(\mathsf{dk}, \mathsf{acc}_i) = 1 \end{array} \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{G}(1^\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ (\mathbb{i}, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2}, \mathsf{acc}, \mathsf{pf}; r) \leftarrow \tilde{\mathrm{P}}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}) \\ ([\mathsf{qw}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{w}]_{i=1}^{m_2}) \leftarrow \mathrm{E}^{\tilde{\mathrm{P}}}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}, r) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathbb{i}) \end{array}\right]$$

The extractor implicitly receives s as input.

Remark 4.1. We can also define a bounded-depth version of completeness with a separate family of deciders. In the completeness definition, valid accumulators for the i-th decider accumulate to a valid accumulator for the $(i+1)$-th decider. This is important in settings where even an honest prover can only perform a bounded number of accumulations.

Multi-instance extraction. As in [BCLMS21], we also define a weaker notion of knowledge soundness which is implied by the earlier definition. For every expected polynomial time adversary $\tilde{\mathrm{P}}$ and auxiliary input distribution $\mathcal{D}$, there exists an expected polynomial time extractor $\mathrm{E}_{\tilde{\mathrm{P}}}$ such that for every set Z,

$$\Pr\left[\begin{array}{c|r} (\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}, \vec{\mathbb{i}}, \vec{\mathbf{qx}}, \vec{\mathbf{ax}}, \vec{\mathsf{acc}}, \mathsf{ao}) \in Z & \rho \leftarrow \mathcal{U}(\lambda) \\ \forall j \in [\ell], \mathrm{V}^{\rho}(\mathsf{avk}, \mathbf{qx}_j, \mathbf{ax}_j, \mathsf{acc}_j.\mathbb{x}, \mathsf{pf}_j) = 1 & \mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{G}(1^{\lambda}) \\ \forall j \in [\ell], \mathrm{D}_{s-1}(\mathsf{dk}_j, \mathsf{acc}_j) = 1 & \mathsf{pp} \leftarrow \mathcal{G}(1^{\lambda}) \\ & \mathsf{ai} \leftarrow \mathcal{D}(1^{\lambda}) \\ & (\vec{\mathbb{i}}, \vec{\mathbf{qx}}, \vec{\mathbf{ax}}, \vec{\mathsf{acc}}, \vec{\mathsf{pf}}, \mathsf{ao}) \leftarrow \tilde{\mathrm{P}}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}) \\ & \forall j \in [\ell], (\mathsf{apk}_j, \mathsf{avk}_j, \mathsf{dk}_j) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathbb{i}_j) \end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c|r} (\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}, \vec{\mathbb{i}}, \vec{\mathbf{qx}}, \vec{\mathbf{ax}}, \vec{\mathsf{acc}}, \mathsf{ao}) \in Z & \rho \leftarrow \mathcal{U}(\lambda) \\ \forall j \in [\ell], \forall i \in [m_1], \hat{\mathcal{V}}(\mathsf{pp}, \mathbb{i}_j, \mathsf{qx}_i^{(j)}, \mathsf{qw}_i^{(j)}) = 1 & \mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{G}(1^{\lambda}) \\ \forall j \in [\ell], \forall i \in [m_2], \mathrm{D}_s(\mathsf{dk}_j, \mathsf{acc}_i^{(j)}) = 1 & \mathsf{pp} \leftarrow \mathcal{G}(1^{\lambda}) \\ & \mathsf{ai} \leftarrow \mathcal{D}(1^{\lambda}) \\ & (\vec{\mathbb{i}}, \vec{\mathbf{qx}}, \vec{\mathbf{qw}}, \vec{\mathbf{ax}}, \vec{\mathbf{qw}}, \vec{\mathsf{acc}}, \mathsf{ao}) \leftarrow \mathrm{E}_{\tilde{\mathrm{P}}}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}) \\ & \forall j \in [\ell], (\mathsf{apk}_j, \mathsf{avk}_j, \mathsf{dk}_j) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathbb{i}_j) \end{array}\right] - \mathrm{negl}(\lambda).$$

Here, we write $\mathbf{qx}_j$ as shorthand for $[\mathsf{qx}_i^{(j)}]_{i=1}^{m_1}$, and similarly for $\mathbf{qw}_j = [\mathsf{qw}_i^{(j)}]_{i=1}^{m_1}$, $\mathbf{ax}_j = [\mathsf{acc}_i^{(j)}.\mathbb{x}]_{i=1}^{m_2}$, and $\mathbf{aw}_j = [\mathsf{acc}_i^{(j)}.\mathbb{w}]_{i=1}^{m_2}$.

5 PCD from bounded-depth accumulation

We construct PCD from bounded-depth accumulation (Section 5.1), and analyze its knowledge soundness (Section 5.2). To simplify our analysis, we only consider compliance predicates which correspond with regular m-ary trees, i.e. $\varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m)$ only accepts if either $\forall i \in [m], z_i = \bot$ or $\forall i \in [m], z_i \neq \bot$. We omit an analysis of completeness and efficiency, as these follow with minimal or no change from the analyses in prior work [BCLMS21].

5.1 Construction

Let $\mathcal{H}(\lambda)$ be a family of collision-resistant hash functions which map to λ bits. Let $\{\mathscr{R}_{\mathsf{pp}}\}_{\mathsf{pp}}$ be an indexed relation family which encodes circuit satisfiability, e.g. R1CS over a field specified by the public parameters. Let $\mathsf{ARG} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ be a NARK for $\{\mathscr{R}_{\mathsf{pp}}\}_{\mathsf{pp}}$. Let $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ be an accumulation scheme for ARG, with bounded-depth knowledge soundness for maximum depth d. We construct a PCD scheme $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$, shown in Figure 7, for the class of bounded-depth compliance predicates $\mathsf{F} = \{\varphi : \mathsf{d}(\varphi) < d\}$.

$R^{(\lambda,N)}_{\mathrm{V},\varphi,\mathsf{H}}\big(h, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}, z_{\mathsf{loc}}, [z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}]^m_{i=1}, \mathsf{pf})\big)$:

1. Check that $h = \mathsf{H}(\mathsf{avk}, z, \mathsf{acc}.\mathbb{x})$.
2. Check that the compliance predicate $\varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m)$ accepts.
3. If there exists $i \in [m]$ such that $z_i \neq \bot$, check that the NARK accumulation verifier accepts:

   $$\mathrm{V}^{(\lambda,m,N)}(\mathsf{avk}, [\mathsf{qx}_i]^m_{i=1}, [\mathsf{acc}_i.\mathbb{x}]^m_{i=1}, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1 \quad \text{where} \quad \mathsf{qx}_i := \big((\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i.\mathbb{x}\big).$$

   Here, $\mathrm{V}^{(\lambda,m,N)}$ refers to the circuit representation of V with input size appropriate for security parameter λ, arity m, and circuit size N.
4. If the above checks hold, output 1; otherwise, output 0.

Figure 6: Recursion circuit for PCD.

Remark 5.1 (accumulator instance size)**.** As noted in [BCLMS21, Remark 5.4], the size of an accumulator instance must be *independent* of the size of an argument instance. This is not the true for the construction presented in Section 6; to avoid this issue, they suggest converting it into a new scheme, where the accumulator instance is hashed and appended to the accumulator witness/proof. We take an alternative approach (also in [KST22]) where the circuit's instance is hashed and appended to the witness. This means that the argument instance is always λ bits, irrespective of the accumulation scheme.

5.2 Knowledge soundness

The extracted transcript T will be a tree, so each vertex u has a unique outgoing edge e. For convenience, we associate the label $z^{(e)}$ with the vertex u itself by writing $z^{(u)} = z^{(e)}$. In our extractors, we will also label a vertex u with a NARK proof $\pi^{(u)}$ and accumulator $\mathsf{acc}^{(u)}$. We recursively construct a sequence of extractors $\mathbb{E}_0, \ldots, \mathbb{E}_d$, where $\mathbb{E}_j$ outputs a tree of depth j. The overall PCD extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ is (essentially) $\mathbb{E}_d$.

- $\mathbb{G}(1^\lambda)$:
  1. Sample $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$.
  2. Sample $\mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{G}(1^\lambda)$.
  3. Sample $\mathsf{H} \leftarrow \mathcal{H}(\lambda)$.
  4. Output $\mathbb{pp} := (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}, \mathsf{H})$.
- $\mathbb{I}(\mathbb{pp}, \varphi)$:
  1. Compute the integer $N := N(\lambda, |\varphi|, m, \ell)$, as defined in [BCLMS21, Lemma 5.5].
  2. Construct the circuit $R := R^{(\lambda,N)}_{\mathrm{V},\varphi,\mathsf{H}}$, as defined in Figure 6.
  3. Compute the accumulation keys $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) := \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathbb{i} = R)$.
  4. Output the proving key $\mathsf{ipk} := (\mathsf{pp}, \mathbb{i}, \mathsf{apk})$ and verification key $\mathsf{ivk} := (\mathsf{pp}, \mathbb{i}, \mathsf{dk}, \mathsf{avk})$.
- $\mathbb{P}(\mathsf{ipk}, z, z_{\mathsf{loc}}, [z_i, (\pi_i, \mathsf{acc}_i)]_{i=1}^m)$:
  1. If $z_i = \bot$ for all $i \in [m]$ then set $\mathsf{acc} = \mathrm{P}(\mathsf{apk}, \bot)$ and $\mathsf{pf} = \bot$.
  2. If $z_i \neq \bot$ for some $i \in [m]$ then:
     (a) set predicate input instance $\mathsf{qx}_i := ((\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i.\mathbb{x})$;
     (b) set predicate input witness $\mathsf{qw}_i := (\mathsf{acc}_i.\mathbb{w}, \pi_i.\mathbb{w})$;
     (c) let $(\mathsf{acc}, \mathsf{pf}) \leftarrow \mathrm{P}(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^m, [\mathsf{acc}_i]_{i=1}^m)$.
  3. Sample $\pi \leftarrow \mathcal{P}\big(\mathsf{pp}, \mathbb{i}, \mathsf{H}(\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), (z_{\mathsf{loc}}, [z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \mathsf{pf})\big)$.
  4. Output (π, acc).
- $\mathbb{V}(\mathsf{ivk}, z, (\pi, \mathsf{acc}))$: Accept if both $\mathcal{V}(\mathsf{pp}, \mathbb{i}, \mathsf{H}(\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), \pi)$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ accept.

Figure 7: PCD algorithms.

We first construct $\mathbb{E}_0$ using the PCD adversary $\tilde{\mathbb{P}}$:

$\mathbb{E}_0(\mathbb{pp}, \mathsf{ai}_{\mathsf{PCD}} = \mathsf{ai})$:
1. Get the prover's output $(\varphi, \mathsf{o}, (\pi, \mathsf{acc}), \mathsf{ao}) \leftarrow \tilde{\mathbb{P}}(\mathbb{pp}, \mathsf{ai})$.
2. Initialize a transcript T with two nodes u, v and an edge (u, v), labeled with $z^{(u)} := \mathsf{o}$.
3. Label u with $\pi^{(u)}$ and $\mathsf{acc}^{(u)}$.
4. Output $(\varphi, \mathsf{T}, \mathsf{ao}_{\mathsf{PCD}} = \mathsf{ao})$.

Suppose we have the extractor $\mathbb{E}_{j-1}$. We show how to construct $\mathbb{E}_j$ in several steps. First, we construct an adversary for ARG:

$\tilde{\mathcal{P}}_j(\mathsf{pp}, \mathsf{ai}_{\mathsf{ARG}} = (\mathsf{pp}_{\mathsf{AS}}, \mathsf{H}, \mathsf{ai}))$:
1. Run the extractor $(\varphi, \mathsf{T}, \mathsf{ao}_{\mathsf{PCD}} = \mathsf{ao}) \leftarrow \mathbb{E}_{j-1}(\mathbb{pp} = (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}, \mathsf{H}), \mathsf{ai}_{\mathsf{PCD}} = \mathsf{ai})$.
2. Construct the circuit $R := R^{(\lambda,N)}_{\mathrm{V},\varphi,\mathsf{H}}$.
3. Run the accumulator indexer $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) := \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, R)$.
4. Initialize an empty set S_{ARG}.
5. For each vertex v in the $(j+1)$-th layer of T:
   - Compute the hash $h := \mathsf{H}(\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x})$.
   - Set $\mathbb{i}_v := R, \mathbb{x}_v := h, \pi_v := \pi^{(v)}$.
6. Output $(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{ao}_{\mathsf{ARG}} = (\varphi, \mathsf{T}, \mathsf{ao}, S_{\mathsf{ARG}}))$.

Next, letting $\mathcal{E}_{\tilde{\mathcal{P}}_j}$ be the extractor for $\tilde{\mathcal{P}}_j$ according to the multi-instance knowledge soundness property of ARG, we construct an adversary for AS:

$\tilde{\mathrm{P}}_j(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}_{\mathsf{AS}} = (\mathsf{H}, \mathsf{ai}))$:

1. Run the extractor $(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\mathbb{w}}, \mathsf{ao}_{\mathsf{ARG}} = (\varphi, \mathsf{T}, \mathsf{ao}, S_{\mathsf{ARG}})) \leftarrow \mathcal{E}_{\tilde{\mathcal{P}}_j}(\mathsf{pp}, \mathsf{ai}_{\mathsf{ARG}} = (\mathsf{pp}_{\mathsf{AS}}, \mathsf{H}, \mathsf{ai}))$.
2. Construct the circuit $R := R^{(\lambda,N)}_{\mathrm{V},\varphi,\mathsf{H}}$.
3. Run the accumulator indexer $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) := \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, R)$.
4. For each vertex v in the $(j+1)$-th layer of T:
   - Parse avk', z', $\mathsf{acc}'.\mathbb{x}$, local data z_{loc}, incoming data $[z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}]_{i\in[m]}$, and accumulation proof pf from the witness $\mathbb{w}^{(v)}$.
   - If $(\mathsf{avk}', z', \mathsf{acc}'.\mathbb{x}) \neq (\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x})$, this constitutes a hash collision; abort.
   - Label v with $z^{(v)}_{\mathsf{loc}} := z_{\mathsf{loc}}$ in T.
   - If there exists $i \in [m], z_i \neq \bot$:
     - Add vertices $u_1, \ldots, u_m$ to T.
     - For each $i \in [m]$, add an edge (u_i, v), labeled with $z^{(u_i)} := z_i$, to T; this is the i-th incoming edge to v.
     - For each $i \in [m]$, compute the hash $h_i := \mathsf{H}(\mathsf{avk}, z^{(u_i)}, \mathsf{acc}_i.\mathbb{x})$ and set $\mathsf{qx}_i := (h_i, \pi_i.\mathbb{x})$.
     - Set $\mathbf{qx}_v = [\mathsf{qx}_i]^m_{i=1}, \mathbf{ax}_v := [\mathsf{acc}_i.\mathbb{x}]^m_{i=1}, \mathsf{acc}_v := \mathsf{acc}^{(v)}, \mathsf{pf}_v := \mathsf{pf}$.
5. Output $(\vec{\mathbb{i}}, \vec{\mathbf{qx}}, \vec{\mathbf{ax}}, \vec{\mathsf{acc}}, \vec{\mathsf{pf}}, \mathsf{ao}_{\mathsf{AS}} = (\varphi, \mathsf{T}, \mathsf{ao}))$.

Let $\mathrm{E}_{\tilde{\mathrm{P}}_j}$ be the extractor for $\tilde{\mathrm{P}}_j$ according to the multi-instance knowledge soundness property of AS. We use the latter to construct the PCD extractor $\mathbb{E}_j$:

$\mathbb{E}_j(\mathbb{pp} = (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}, \mathsf{H}), \mathsf{ai}_{\mathsf{PCD}} = \mathsf{ai})$:

1. Run the extractor $(\vec{\mathbb{i}}, \vec{\mathbf{qx}}, \vec{\mathbf{qw}}, \vec{\mathbf{ax}}, \vec{\mathbf{aw}}, \vec{\mathsf{acc}}, \mathsf{ao}_{\mathsf{AS}} = (\varphi, \mathsf{T}, \mathsf{ao})) \leftarrow \mathrm{E}_j(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}_{\mathsf{AS}} = (\mathsf{H}, \mathsf{ai}))$.
2. For each vertex v in the $(j+1)$-th layer of T:
   - Parse $[\mathsf{qx}_i = (h_i, \pi_i.\mathbb{x})]^m_{i=1}$ from $\mathbf{qx}^{(v)}$.
   - Parse $[\mathsf{qw}_i = \pi_i.\mathbb{w}]^m_{i=1}$ from $\mathbf{qw}^{(v)}$.
   - Parse $[\mathsf{acc}_i.\mathbb{x}]^m_{i=1}$ from $\mathbf{ax}^{(v)}$.
   - Parse $[\mathsf{acc}_i.\mathbb{w}]^m_{i=1}$ from $\mathbf{aw}^{(v)}$.
   - Let $u_1, \ldots, u_m$ be the children of v.
   - For each $i \in [m]$:
     - Combine $\pi_i.\mathbb{x}$ and $\pi_i.\mathbb{w}$ into a NARK proof π.
     - Combine $\mathsf{acc}_i.\mathbb{x}$ and $\mathsf{acc}_i.\mathbb{w}$ into an accumulator acc.
     - Label u_i with $\pi^{(u_i)} := \pi$ and $\mathsf{acc}^{(u_i)} := \mathsf{acc}$.
3. Output $(\varphi, \mathsf{T}, \mathsf{ao}_{\mathsf{PCD}} = \mathsf{ao})$.

Finally, we construct the overall PCD extractor:

$\mathbb{E}_{\tilde{\mathbb{P}}}(\mathbb{pp}, \mathsf{ai})$:

1. Run the extractor $(\varphi, \mathsf{T}, \mathsf{ao}_{\mathsf{PCD}} = \mathsf{ao}) \leftarrow \mathbb{E}_{d+1}(\mathbb{pp}, \mathsf{ai}_{\mathsf{PCD}} = \mathsf{ai})$.
2. Remove any unnecessary labeling, such as $\pi^{(v)}$ and $\mathsf{acc}^{(v)}$, from each vertex v in T.
3. Output $(\varphi, \mathsf{T}, \mathsf{ao})$.

Running time of the extractor. It follows from the extraction guarantees of ARG and AS that $\mathbb{E}_j$ runs in expected time polynomial in the expected running time of $\mathbb{E}_{j-1}$. Since d is constant, the overall extractor runs in expected polynomial time.

Correctness of the extractor. Fix a set Z, and suppose

$$\Pr\left[\begin{array}{c|r} \varphi \in \mathsf{F} & \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (\mathbb{pp}, \mathsf{ai}, \varphi, \mathsf{o}, \mathsf{ao}) \in Z & \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ \mathbb{V}(\mathsf{ivk}, \mathsf{o}, (\pi, \mathsf{acc})) = 1 & (\varphi, \mathsf{o}, (\pi, \mathsf{acc}), \mathsf{ao}) \leftarrow \tilde{\mathbb{P}}(\mathbb{pp}, \mathsf{ai}) \\ & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathbb{pp}, \varphi) \end{array}\right] = \mu. \tag{1}$$

Throughout our proof, we will reference the index R, verification key avk, and decision key dk with the understanding that these can be deterministically derived from $\mathbb{pp}$ and φ via the PCD indexer. We show by induction that, for all $j \in \{0, \ldots, d\}$, the extractor $\mathbb{E}_j(\mathbb{pp}, \mathsf{ai})$ outputs $(\varphi, \mathsf{T}, \mathsf{ao}_{\mathsf{PCD}} = \mathsf{ao})$ such that the following holds with probability $\mu - \mathrm{negl}(\lambda)$ (over the public parameters, auxiliary input, and the extractor's randomness):

- $\varphi \in \mathsf{F}$ and $(\mathbb{pp}, \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}), \mathsf{ao}) \in Z$.
- T is a transcript tree of depth at most j (and hence there are at most $j+2$ layers). Moreover, the vertices in the first $j+1$ layers of T are φ-compliant.
- For all u in the $(j+2)$-th layer of T, both $\hat{\mathcal{V}}(R, h, \pi^{(u)})$ and $\mathrm{D}_j(\mathsf{dk}, \mathsf{acc}^{(u)})$ accept, where $h := \mathsf{H}(\mathsf{avk}, z^{(u)}, \mathsf{acc}^{(u)}.\mathbb{x})$.

For the base case $\mathbb{E}_0$, this is implied by Equation (1). In particular, the sink v is trivially φ-compliant, since it has no outgoing edges. Since the PCD verifier accepts, the strict decider D and verifier $\mathcal{V}$ accept. Hence, D_0 and $\hat{\mathcal{V}}$ accept. For the inductive step, suppose that $\mathbb{E}_{j-1}$ satisfies the inductive hypothesis.

Correctness of $\mathcal{E}_j$. The set Z_{ARG} is defined as follows: $(\mathsf{pp}, \mathsf{ai}_{\mathsf{ARG}} = (\mathsf{pp}_{\mathsf{AS}}, \mathsf{H}, \mathsf{ai}), \vec{\mathbb{i}}, \vec{\mathbb{x}}, \mathsf{ao}_{\mathsf{ARG}} = (\varphi, \mathsf{T}, \mathsf{ao}))$ is an element of Z_{ARG} if and only if the following holds:

- $\varphi \in \mathsf{F}$ and $(\mathbb{pp} = (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}, \mathsf{H}), \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}), \mathsf{ao}) \in Z$.
- T is a transcript tree of depth at most $j-1$ (and hence there are $j+1$ layers). Moreover, the vertices in the first j layers of T are φ-compliant.
- For all v in the $(j+1)$-th layer of T, $\mathrm{D}_{j-1}(\mathsf{dk}, \mathsf{acc}^{(v)})$ accepts. Moreover, $\mathbb{i}^{(v)} = R$ and $\mathbb{x}_v = \mathsf{H}(\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x})$.

By construction, with probability $\mu - \mathrm{negl}(\lambda)$, the adversary $\tilde{\mathcal{P}}_j(\mathsf{pp}, \mathsf{ai}_{\mathsf{ARG}})$ outputs $(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{ao}_{\mathsf{ARG}})$ such that $(\mathsf{pp}, \mathsf{ai}_{\mathsf{ARG}}, \vec{\mathbb{i}}, \vec{\mathbb{x}}, \mathsf{ao}_{\mathsf{ARG}}) \in Z_{\mathsf{ARG}}$ and for all v, $\hat{\mathcal{V}}(\mathbb{i}_v, \mathbb{x}_v, \pi_v)$ accepts. By (multi-instance) knowledge soundness of ARG, with probability $\mu - \mathrm{negl}(\lambda)$, the extractor $\mathcal{E}_{\tilde{\mathcal{P}}_j}(\mathsf{pp}, \mathsf{ai}_{\mathsf{ARG}})$ outputs $(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\mathbb{w}}, \mathsf{ao}_{\mathsf{ARG}})$ and we have that $(\mathsf{pp}, \mathsf{ai}_{\mathsf{ARG}}, \vec{\mathbb{i}}, \vec{\mathbb{x}}, \mathsf{ao}_{\mathsf{ARG}}) \in Z_{\mathsf{ARG}}$ and for all v, $(\mathbb{i}_v, \mathbb{x}_v, \mathbb{w}_v) \in \mathscr{R}_{\mathsf{pp}}$.

Correctness of E_j. Define the set Z_{AS} as follows:

$$(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}_{\mathsf{AS}} = (\mathsf{H}, \mathsf{ai}), \vec{\mathbb{i}}, \vec{\mathbf{qx}}, \vec{\mathbf{ax}}, \vec{\mathsf{acc}}, \vec{\mathsf{pf}}, \mathsf{ao}_{\mathsf{AS}} = (\varphi, \mathsf{T}, \mathsf{ao})) \in Z_{\mathsf{AS}}$$

if and only if the following holds:

- $\varphi \in \mathsf{F}$ and $(\mathbb{pp} = (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}, \mathsf{H}), \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}), \mathsf{ao}) \in Z$.
- T is a transcript tree of depth at most j (and hence there are $j+2$ layers). Moreover, the vertices in the first $j+1$ layers of T are φ-compliant.
- For each non-source vertex v in the $(j+1)$-th layer of T, v has children $u_1, \ldots, u_m$ such that $\mathbb{i}_v = R$ and $\mathbf{ax}_v = [\mathsf{acc}^{u_i}.\mathbb{x}]_{i=1}^m$ such that $\mathbf{qx}_v = [h_i, \pi_i.\mathbb{x}]_{i=1}^m$ for $h_i = \mathsf{H}(\mathsf{avk}, z^{(u_i)}, \mathsf{acc}_i.\mathbb{x})$.

Suppose E_j obtains $(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\mathbb{w}}, \mathsf{ao}_{\mathsf{ARG}})$ such that $(\mathsf{pp}, \mathsf{ai}_{\mathsf{ARG}}, \vec{\mathbb{i}}, \vec{\mathbb{x}}, \mathsf{ao}_{\mathsf{ARG}}) \in Z_{\mathsf{ARG}}$ and for each v, $(\mathbb{i}_v, \mathbb{x}_v, \mathbb{w}_v) \in \mathscr{R}_{\mathsf{pp}}$. By membership in Z_{ARG}, for each v in the $(j+1)$-th layer of the tree, we have the following:

- $\mathsf{H}(\mathsf{avk}', z', \mathsf{acc}'.\mathbb{x}) = \mathbb{x}_v$.
- If v has no children $\varphi(z', z^{(v)}_{\mathsf{loc}}, \bot, \ldots, \bot)$ accepts.
- If v has children $u_1, \ldots, u_m$, $\varphi(z', z^{(v)}_{\mathsf{loc}}, z^{(u_1)}, \ldots, z^{(u_m)})$ accepts and $\mathrm{V}(\mathsf{avk}', \mathbf{qx}_v, \mathbf{ax}_v, \mathsf{acc}_v, \mathsf{pf}_v)$ accepts.

By membership in Z_{ARG}, we also have $\mathsf{H}(\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x}) = \mathbb{x}_v$. Since $\mathcal{H}$ is collision-resistant, we conclude that $\mathsf{avk} = \mathsf{avk}'$, $z^{(v)} = z'$, and $\mathsf{acc}^{(v)}.\mathbb{x} = \mathsf{acc}'.\mathbb{x}$. The rest of the argument follows from construction: with probability $\mu - \mathrm{negl}(\lambda)$, the adversary $\tilde{\mathrm{P}}_j(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}_{\mathsf{AS}})$ outputs $(\vec{\mathbb{i}}, \vec{\mathbf{qx}}, \vec{\mathbf{ax}}, \vec{\mathsf{acc}}, \vec{\mathsf{pf}}, \mathsf{ao}_{\mathsf{AS}})$ such that $(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}_{\mathsf{AS}}, \vec{\mathbb{i}}, \vec{\mathbf{qx}}, \vec{\mathbf{ax}}, \vec{\mathsf{acc}}, \vec{\mathsf{pf}}, \mathsf{ao}_{\mathsf{AS}}) \in Z_{\mathsf{AS}}$, and for all v, both $\mathrm{V}(\mathsf{avk}_v, \mathbf{qx}_v, \mathbf{ax}_v, \mathsf{acc}_v, \mathsf{pf}_v)$ and $\mathrm{D}_{j-1}(\mathsf{dk}_v, \mathsf{acc}_v)$ accept. Here, avk_v and dk_v are derived from $\mathbb{i}_v$. By (multi-instance) knowledge soundness of AS, with probability $\mu - \mathrm{negl}(\lambda)$, the extractor $\mathrm{E}_j(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}_{\mathsf{AS}})$ outputs $(\vec{\mathbb{i}}, \vec{\mathbf{qx}}, \vec{\mathbf{qw}}, \vec{\mathbf{ax}}, \vec{\mathbf{aw}}, \vec{\mathsf{acc}}, \mathsf{ao}_{\mathsf{AS}})$ such that $(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}_{\mathsf{AS}}, \vec{\mathbb{i}}, \vec{\mathbf{qx}}, \vec{\mathbf{ax}}, \vec{\mathsf{acc}}, \vec{\mathsf{pf}}, \mathsf{ao}_{\mathsf{AS}}) \in Z_{\mathsf{AS}}$ and for all v and $i \in [m]$, both $\hat{\mathcal{V}}(\mathbb{i}_v, \mathsf{qx}^{(v)}_i, \mathsf{qw}^{(v)}_i)$ and $\mathrm{D}_j(\mathsf{dk}_v, \mathsf{acc}^{(v)}_i)$ accept. Here, dk_v is derived from $\mathbb{i}$.

Correctness of $\mathbb{E}_j$. By construction, with probability $\mu - \mathrm{negl}(\lambda)$, the extractor $\mathbb{E}_j$ satisfies the inductive hypothesis. The key requirement is that the NARK proof $\pi^{(u)}$ and accumulator $\mathsf{acc}^{(u)}$ are related by the hash, which is guaranteed by membership in Z_{AS}. This concludes our proof by induction.

Correctness of $\mathbb{E}_{\tilde{\mathbb{P}}}$. With probability $\mu - \mathrm{negl}(\lambda)$, $\mathbb{E}_{\tilde{\mathbb{P}}}$ outputs $(\phi, \mathsf{T}, \mathsf{ao})$ such that $\varphi \in \mathsf{F}$, $(\mathbb{pp}, \mathsf{ai}, \mathsf{o}(\mathsf{T}), \mathsf{ao}) \in Z$, and the first $d+2$ layers of T are φ-compliant. Since the depth of φ is at most d, we conclude that all of the vertices in the $(d+2)$-th layer are trivially accepted by the compliance predicate, and their incoming edges must all be labeled with $\bot$. Hence, T is φ-compliant.

6 Constructing bounded-depth accumulation

We construct a bounded-depth accumulation scheme, which supports (possibly distinct) arities $m_1 = \mathrm{poly}(\lambda)$ and $m_2 = \mathrm{poly}(\lambda)$, for a general class of non-interactive arguments. Across all of our constructions, we fix the following global constants: depth bound d_s, code rate $R \in (0, 1)$, and distance $\delta \leq (1 - R)/2d_s$; this guarantees that $d_s\delta$ is smaller than the unique decoding radius of a Reed–Solomon code with rate R. Additionally, we use domain separation on the random oracle ρ to model three disjoint oracles: ρ_H for Merkle trees and hashing, ρ_{ARG} for the argument verifier's randomness, and ρ_{AS} for the accumulation verifier's randomness. When querying ρ_{ARG} and ρ_{AS}, we assume the random oracle's output is used to sample from the verifier's challenge set.

Notation. If x, y, z are vectors, we will often interpret tuples like $(x, (y, z))$ as a vector consisting of x, y, z concatenated together. Given a codeword $f \in C$, let $\hat{f}$ denote the corresponding message which encodes to f. We write $\mathbf{f}$ as shorthand for the tuple $(f^{(1)}, \ldots, f^{(\mu)})$ and $\mathbf{C}$ as shorthand for the Cartesian product $C^{(1)} \times \cdots \times C^{(\mu)}$. Similarly, let $\hat{\mathbf{f}} = (\hat{f}^{(1)}, \ldots, \hat{f}^{(\mu)})$ and $\Delta(\mathbf{f}, \mathbf{g}) = \max_{j \in [\mu]} \Delta(f^{(j)}, g^{(j)})$. When querying locations in a codeword, let $\mathbf{Q} = (\mathcal{Q}^{(1)}, \ldots, \mathcal{Q}^{(\mu)})$ and $\mathbf{f}[\mathbf{Q}] = (f^{(1)}[\mathcal{Q}^{(1)}], \ldots, f^{(\mu)}[\mathcal{Q}^{(\mu)}])$. We use arrow notation as shorthand for tuples of commitment data, e.g. $\vec{\mathsf{cm}} = (\mathsf{cm}^{(1)}, \ldots, \mathsf{cm}^{(\mu)})$. Vector commitment functions map over tuples, e.g. $(\vec{\mathsf{cm}}, \vec{\mathsf{aux}}) \leftarrow \mathsf{VC.Commit}(\mathsf{vp}, \mathbf{f})$ should be interpreted as saying "for each $j \in [\mu]$, let $(\vec{\mathsf{cm}}^{(j)}, \vec{\mathsf{aux}}^{(j)}) \leftarrow \mathsf{VC.Commit}(\mathsf{vp}, f^{(j)})$."

6.1 Non-interactive argument

Our starting point is any special-sound interactive proof with an algebraic verifier. By this, we mean that the verifier's check can be expressed as a sequence of degree d polynomials (derived from the index) which take the transcript as input. The verifier accepts if all of the polynomials evaluate to zero.

In more detail, we require an interactive proof for some indexed relation $\mathscr{R}(\mathbb{F})$. Let μ be the number of rounds in the protocol; this may be a function of the index. For simplicity, we assume that the instance $\mathbb{x}$, the prover's messages $m^{(1)}, \ldots, m^{(\mu)}$, and the verifier's challenges $r^{(1)}, \ldots, r^{(\mu)}$ are all vectors over $\mathbb{F}$; their lengths may be a function of the index. From the index, the verifier derives degree d polynomials $p_1, \ldots, p_\ell$ over $\mathbb{F}$. It accepts if, for all i, $p_i(\mathbb{x}, \mathbf{r}, \mathbf{m}) = 0$.

Compressing the verifier. Without loss of generality, we can assume that the algebraic verifier consists of a single polynomial. This is because multiple polynomial checks can be compressed into a single check, e.g. by using the following technique due to [EG23; BC23].

Let Π be an interactive proof where the verifier's check consists of ℓ polynomials $p_0, \ldots, p_{\ell-1}$, each of degree d. We transform this into an interactive proof $\mathsf{CV}[\Pi]$ for the same relation, where the verifier's check is a single polynomial

$$p(\vec{X}, \vec{Y}) = \sum_{i=0}^{\ell-1} \mathsf{pow}_i(\vec{Y}) \cdot p_i(\vec{X}).$$

Here, pow_i is the unique degree $\log \ell$ polynomial satisfying $\mathsf{pow}_i(1, \beta, \beta^2, \beta^4, \ldots, \beta^{\ell/2}) = \beta^i$ for all $\beta \in \mathbb{F}$. It follows that p is of degree $d + \log \ell$. The interactive protocol is the same as before, except that the verifier additionally samples a challenge $y = (1, \beta, \beta^2, \beta^4, \ldots, \beta^{\ell/2})$ where $\beta \leftarrow \mathbb{F}$. The verifier accepts if $p(x, y) = 0$, where x is the transcript from the original proof.

If Π is $(k_1, \ldots, k_\mu)$-special-sound, then $\mathsf{CV}[\Pi]$ is $(k_1, \ldots, k_\mu, \ell)$-special-sound. To see why, suppose we have a tree T of accepting transcripts for $\mathsf{CV}[\Pi]$. Consider an arbitrary node in the penultimate layer of the tree. Its children correspond with transcripts of the form $(x, y_1), \ldots, (x, y_\ell)$, each y_i distinct. Since

the transcripts are accepting, the degree $\ell - 1$ univariate polynomial $\sum_{i=0}^{\ell-1} Z^i \cdot p_i(x)$ is zero at ℓ points. It follows that f is the zero polynomial and, for all i, $p_i(x) = 0$. Let T' denote T with its bottom layer removed. We have shown that T' is a tree of accepting transcripts for Π. Hence, we construct an extractor for $\mathsf{CV}[\Pi]$ by running the extractor for Π on T'.

- $\mathcal{G}(1^\lambda)$:
  1. Choose a suitable field $\mathbb{F}, \log|\mathbb{F}| = \Omega(\lambda)$.
  2. Let $\mathsf{vp} \leftarrow \mathsf{VC.Setup}(1^\lambda, \mathbb{F})$.
  3. Output $\mathsf{pp} = (\mathbb{F}, \mathsf{vp})$.
- $\mathcal{I}^\rho(\mathsf{pp} = (\mathbb{F}, \mathsf{vp}), \mathbb{i})$:
  1. Query $\tau \leftarrow \rho_H(\mathbb{i})$.
  2. From $\mathbb{F}$ and $\mathbb{i}$, derive the following parameters, collected into $\mathbb{p}$:
     - The number of rounds, denoted μ.
     - The length of the instance.
     - For each $j \in [\mu]$, the length, denoted $\ell^{(j)}$, of the prover's j-th message.
     - For each $j \in [\mu]$, the format of the verifier's j-th challenge.
  3. From $\mathbb{F}$ and $\mathbb{i}$, derive the verifier's check p.
  4. For each $j \in [\mu]$, let $C^{(j)}$ be a Reed–Solomon code over $\mathbb{F}$ with dimension $\ell^{(j)}$, rate R, and evaluation domain $L^{(j)}$.
  5. Output $\mathrm{ipk} = (\mathbb{F}, \mathsf{vp}, \mathbb{i}, \tau, \mathbb{p}, \mathbf{C})$ and $\mathrm{ivk} = (\mathbb{F}, \mathsf{vp}, \tau, \mathbb{p}, p, \mathbf{C})$.
- $\mathcal{P}^\rho(\mathsf{pp}, \mathbb{i}, \mathbb{x}, \mathbb{w})$:
  1. Compute the proving key $\mathrm{ipk} = (\mathbb{F}, \mathsf{vp}, \mathbb{i}, \tau, \mathbb{p}, \mathbf{C})$ according to $\mathcal{I}^\rho(\mathsf{pp}, \mathbb{i})$.
  2. For $j \in [\mu]$:
     - Compute the prover's j-th message $m^{(j)} \leftarrow P(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathbb{w}, [m^{(i)}, r^{(i)}]_{i=1}^{j-1})$.
     - Encode $m^{(j)}$ to $f^{(j)} \in C^{(j)}$.
     - Let $(\mathsf{cm}^{(j)}, \mathsf{aux}^{(j)}) \leftarrow \mathsf{VC.Commit}^{\rho_H}(\mathsf{vp}, f^{(j)})$.
     - Query $r^{(j)} \leftarrow \rho_{\mathsf{ARG}}(\tau, \mathbb{x}, \mathsf{cm}^{(1)}, \ldots, \mathsf{cm}^{(j)})$.
  3. Output $\pi = (\vec{\mathsf{cm}}, \vec{\mathsf{aux}})$.
- $\mathcal{V}^\rho(\mathsf{pp}, \mathbb{i}, \mathbb{x}, \pi = (\vec{\mathsf{cm}}, \vec{\mathsf{aux}}))$:
  1. Compute the verification key $\mathrm{ivk} = (\mathbb{F}, \mathsf{vp}, \tau, \mathbb{p}, p, \mathbf{C})$ according to $\mathcal{I}^\rho(\mathsf{pp}, \mathbb{i})$.
  2. For each $j \in [\mu]$, query $r^{(j)} \leftarrow \rho_{\mathsf{ARG}}(\tau, \mathbb{x}, \mathsf{cm}^{(1)}, \ldots, \mathsf{cm}^{(j)})$.
  3. Let $\mathbf{f} \leftarrow \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}, \vec{\mathsf{aux}})$.
  4. Verify that $\mathbf{f} \in \mathbf{C}$.
  5. Accept if $p(\mathbb{x}, \mathbf{r}, \hat{\mathbf{f}}) = 0$.
- $\hat{\mathcal{V}}^\rho(\mathsf{pp}, \mathbb{i}, \mathbb{x}, \pi = (\vec{\mathsf{cm}}, \vec{\mathsf{aux}}))$:
  1. Compute the verification key $\mathrm{ivk} = (\mathbb{F}, \mathsf{vp}, \tau, \mathbb{p}, p, \mathbf{C})$ according to $\mathcal{I}^\rho(\mathsf{pp}, \mathbb{i})$.
  2. For each $j \in [\mu]$, query $r^{(j)} \leftarrow \rho_{\mathsf{ARG}}(\tau, \mathbb{x}, \mathsf{cm}^{(1)}, \ldots, \mathsf{cm}^{(j)})$.
  3. Let $\mathbf{f} \leftarrow \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}, \vec{\mathsf{aux}})$.
  4. Find $\mathbf{g} \in \mathbf{C}$ by uniquely decoding $\mathbf{f}$. If no such codeword exists, reject.
  5. Accept if $p(\mathbb{x}, \mathbf{r}, \hat{\mathbf{g}}) = 0$.

Figure 8: Our NARK construction.

Committing to messages. In order to achieve efficient accumulation, we will instead have the prover send commitments to messages. Only in the final move of the protocol does the prover send openings to all of the commitments. Strictly speaking, this is only special-sound for an "augmented relation"; namely, there exists an extractor which, given a tree of accepting transcripts, outputs either a witness or a break of the commitment scheme. Nonetheless, assuming the scheme is computationally binding, applying the Fiat-Shamir transformation yields a non-interactive argument of knowledge for the original relation. We refer to [BC23] for a more detailed analysis.

Removing interaction. Given a special-sound interactive proof (P, V), we apply the Fiat–Shamir transformation (with commitments) to get a non-interactive argument of knowledge [AFK23]. In order to achieve efficient accumulation, we use a standard variant of the transformation where the index is first hashed to a succinct value τ. The Fiat–Shamir transform outputs a non-interactive argument $\mathsf{ARG} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$, shown in Figure 8, for the indexed relation family $\{\mathscr{R}_{\mathsf{pp}} = \mathscr{R}(\mathbb{F})\}$. We also define an indexer $\mathcal{I}$ as a helper algorithm.

Attema et al. [AFK23] prove that multi-round public-coin protocols can be compiled into non-interactive arguments. However, their definition of knowledge-soundness is slightly different from ours: they require that the extractor succeeds with non-negligible probability if the adversary succeeds with non-negligible probability. Our definition, on the other hand, requires that the extractor succeeds with all but negligible probability whenever the adversary succeeds. However, these definitions are equivalent as one can boost the extractor's success probability by running it until it succeeds. This works as the extractor is only required to run in *expected* polynomial time.

Relaxed verifier. We use a specific commitment scheme to support accumulation: the prover sends a vector commitment to a codeword of the message, along with the full auxiliary data. We relax the verifier in two different ways to get $\hat{\mathcal{V}}$ (also shown in Figure 8). First, we allow it to decode noisy codewords. Second, we allow it to accept partial openings to commitments; the missing positions correspond with erasure errors. These changes do not affect knowledge soundness, since the prover is still bound to (decoded) messages.

6.2 Accumulation scheme

Fix a subset $H \subset \mathbb{F}$ of size $m = m_1 + m_2$; this may be a function of λ. We construct an accumulation scheme $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ for ARG. The argument verifier inputs are split into $\mathsf{qx} = (\mathbb{x}, \vec{\mathsf{cm}})$ and $\mathsf{qw} = \vec{\mathsf{aux}}$. Accumulators have a nearly identical structure; in fact, any verifier input $(\mathsf{qx}, \mathsf{qw})$ can be converted into an accumulator acc by setting $\mathsf{acc}.\mathbb{w} = \mathsf{qw}$ and $\mathsf{acc}.\mathbb{x} = \textsc{CastInput}^{\rho}(\tau, \mathsf{qx})$, which is defined below.

$\textsc{CastInput}^{\rho}(\tau, \mathsf{qx} = (\mathbb{x}, \vec{\mathsf{cm}}))$:
1. For each $j \in [\mu]$, query $r^{(j)} \leftarrow \rho_{\mathsf{ARG}}(\tau, \mathbb{x}, \mathsf{cm}^{(1)}, \ldots, \mathsf{cm}^{(j)})$.
2. Collect $\mathbb{x}$ and $\mathbf{r}$ into a vector x.
3. Output $\mathsf{acc}.\mathbb{x} = (0, x, \vec{\mathsf{cm}})$.

We also define a helper function which casts $[\mathsf{qx}_i]_{i=1}^{m_1}$ and $[\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2}$ into a single list of $m = m_1 + m_2$ accumulator instances.

$\textsc{Cast}^{\rho}(\tau, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2})$:
1. Output $[\textsc{CastInput}(\tau, \mathsf{qx}_1), \ldots, \textsc{CastInput}(\tau, \mathsf{qx}_{m_1}), \mathsf{acc}_1.\mathbb{x}, \ldots, \mathsf{acc}_{m_2}.\mathbb{x}]$.

Finally, we define helper functions which perform the bulk of proving and verification.

$\textsc{Prove}^{\rho}(\mathsf{apk}, [e_i, x_i, \vec{\mathsf{cm}}_i, \vec{\mathsf{aux}}_i]_{i=1}^m)$:

1. For each $i \in [m]$, let $\mathbf{f}_i \leftarrow \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}_i, \vec{\mathsf{aux}}_i)$.
2. Compute the univariate polynomial $q \in \mathbb{F}[X]$ of degree at most $d(m-1) - m$ such that

$$p\left(\sum_{i=1}^{m} L_{i,H}(X) \cdot (x_i, \hat{\mathbf{f}}_i)\right) = v_H(X) \cdot q(X) + \sum_{i=1}^{m} L_{i,H}(X) \cdot e_i.$$

3. Query $\alpha \leftarrow \rho_{\mathsf{AS}}(\tau, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, q)$, where $\alpha \in \mathbb{F}$ is a uniformly sampled field element.
4. Compute $e = v(\alpha) \cdot q(\alpha) + \sum_i L_i(\alpha) \cdot e_i$.
5. Compute $(x, \hat{\mathbf{f}}) = \sum_i L_i(\alpha) \cdot (x_i, \hat{\mathbf{f}}_i)$.
6. Let $(\vec{\mathsf{cm}}, \vec{\mathsf{aux}}) \leftarrow \mathsf{VC.Commit}^{\rho_H}(\mathsf{vp}, \mathbf{f})$.
7. Set $\mathsf{acc}.\mathbb{x} = (e, x, \vec{\mathsf{cm}})$ and $\mathsf{acc}.\mathbb{w} = \vec{\mathsf{aux}}$.
8. Query $\mathbf{Q} \leftarrow \rho_{\mathsf{AS}}(\tau, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, q, \mathsf{acc}.\mathbb{x})$, where $\mathcal{Q}^{(j)}$ is a uniformly sampled t-sized subset of $L^{(j)}$.
9. For each $i \in [m]$, let $\vec{\mathsf{op}}_i \leftarrow \mathsf{VC.Open}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{aux}}_i, \mathbf{Q})$.
10. Let $\vec{\mathsf{op}} \leftarrow \mathsf{VC.Open}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{aux}}, \mathbf{Q})$.
11. Output acc and $\mathsf{pf} = (q, [\vec{\mathsf{op}}_i]_{i=1}^m, \vec{\mathsf{op}})$.

$\textsc{Verify}^{\rho}(\mathsf{avk}, [e_i, x_i, \vec{\mathsf{cm}}_i]_{i=1}^m, (e, x, \vec{\mathsf{cm}}), (q, [\vec{\mathsf{op}}_i]_{i=1}^m, \vec{\mathsf{op}}))$:

1. Query $\alpha \leftarrow \rho_{\mathsf{AS}}(\tau, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, q)$.
2. Query $\mathbf{Q} \leftarrow \rho_{\mathsf{AS}}(\tau, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, q, \mathsf{acc}.\mathbb{x})$.
3. For each $i \in [m]$, let $\mathbf{v}_i = \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}_i, \vec{\mathsf{op}}_i)$. If $\mathbf{v}_i[\mathbf{Q}]$ contains $\perp$, reject.
4. Let $\mathbf{v} = \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}, \vec{\mathsf{op}}, \mathbf{Q})$. If $\mathbf{v}[\mathbf{Q}]$ contains $\perp$, reject.
5. Verify that $e = v_H(\alpha) \cdot q(\alpha) + \sum_{i=1}^n L_{i,H}(\alpha) \cdot e^{(i)}$.
6. Verify that $(x, \mathbf{v}) = \sum_{i=1}^m L_{i,H}(\alpha) \cdot (x_i, \mathbf{v}_i)$.

See Figure 9 for a full description of AS, along with the family of deciders.

- $\mathrm{G}(1^\lambda)$:
  1. Choose a suitable spot check parameter $t = \Omega(\lambda)$.
  2. Output $\mathsf{pp}_{\mathsf{AS}} = t$.
- $\mathrm{I}^{\rho}(\mathsf{pp}_{\mathsf{AS}} = t, \mathsf{pp} = (\mathbb{F}, \mathsf{vp}), \mathbb{i})$:
  1. Obtain $\tau, \mathbb{p}, p, \mathbf{C}$ from $\mathcal{I}^{\rho}(\mathsf{pp}, \mathbb{i})$.
  2. Output $\mathsf{apk} = (\mathbb{F}, \mathsf{vp}, t, \tau, \mathbb{p}, p, \mathbf{C})$, $\mathsf{avk} = (\mathbb{F}, \mathsf{vp}, t, \tau, \mathbb{p}, \mathbf{C})$, and $\mathsf{dk} = (\mathbb{F}, \mathsf{vp}, \mathbb{p}, p, \mathbf{C})$.
- $\mathrm{P}^{\rho}(\mathsf{apk}, [\mathsf{qx}_i, \mathsf{qw}_i]_{i=1}^{m_1}, [\mathsf{acc}]_{i=1}^{m_2})$:
  1. Let $[e_i, x_i, \vec{\mathsf{cm}}_i]_{i=1}^m \leftarrow \textsc{Cast}(\tau, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2})$.
  2. Parse $[\vec{\mathsf{aux}}_i]_{i=1}^m = [\mathsf{qw}_1, \ldots, \mathsf{qw}_{m_1}, \mathsf{acc}_1.\mathbb{w}, \ldots, \mathsf{acc}_{m_2}.\mathbb{w}]$.
  3. Output $\textsc{Prove}^{\rho}(\mathsf{apk}, [e_i, x_i, \vec{\mathsf{cm}}_i, \vec{\mathsf{aux}}_i]_{i=1}^m)$.
- $\mathrm{V}^{\rho}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2}, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$:
  1. Let $[e_i, x_i, \vec{\mathsf{cm}}_i]_{i=1}^m \leftarrow \textsc{Cast}(\tau, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2})$.
  2. Accept if $\textsc{Verify}(\mathsf{avk}, [e_i, x_i, \vec{\mathsf{cm}}_i]_{i=1}^m, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$ accepts.
- $\mathrm{D}_s^{\rho}(\mathsf{dk}, \mathsf{acc}.\mathbb{x} = (e, x, \vec{\mathsf{cm}}), \mathsf{acc}.\mathbb{w} = \vec{\mathsf{aux}})$:
  1. Let $\mathbf{f} \leftarrow \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}, \vec{\mathsf{aux}})$.
  2. Find $\mathbf{g} \in \mathbf{C}, \Delta(\mathbf{f}, \mathbf{g}) \leq s\delta$ by decoding $\mathbf{f}$. If no such codeword exists, reject.
  3. Accept if $p(x, \hat{\mathbf{g}}) = e$.

Figure 9: Accumulation scheme algorithms.

Completeness. Suppose $\mathcal{A}$ outputs $\mathtt{i}$, $[\mathsf{qx}_i, \mathsf{qw}_i]_{i=1}^{m_1}$, and $[\mathsf{acc}_i]_{i=1}^{m_2}$, where the inputs are accepted by $\mathcal{V}$ and the accumulators are accepted by D. After casting, the prover holds $[e_i, x_i, \mathsf{cm}_i, \vec{\mathsf{aux}}_i]_{i=1}^{m}$ such that, for all i, the auxiliary data $\vec{\mathsf{aux}}_i$ opens to codewords $\mathbf{f}_i \in \mathbf{C}, p(x_i, \hat{\mathbf{f}}_i) = e_i$. Consider the degree $d(m-1)$ polynomial

$$p\left(\sum_{i=1}^{m} L_{i,H}(X) \cdot (x_i, \hat{\mathbf{f}}_i)\right) - \sum_{i=1}^{n} L_{i,H}(X) \cdot e_i.$$

At each evaluation point $a_j \in H$, $\sum_i L_i(a_j) \cdot (x_i, \hat{\mathbf{f}}_i) = (x_j, \hat{\mathbf{f}}_j)$, since $L_j(a_j) = 1$ and $L_i(a_j) = 0$ for $i \neq j$. Similarly, $\sum_i L_i(a_j) \cdot e_i = e_j$. It follows that the polynomial is zero at all points in H, and thus factors into $v_H(X) \cdot q(X)$. Since v_H has degree m, q has degree $d(m-1) - m$. It remains to be shown that the spot check succeeds. Since the accumulators are valid, the prover can open the codewords at any location. Next, recall that the prover computes $\hat{\mathbf{f}} = \sum_i L_i(\alpha) \cdot \hat{\mathbf{f}}_i$. By linearity of the code, this relationship holds over the entire codeword.

6.3 Soundness analysis

Our accumulation scheme can be viewed as a compiled interactive oracle proof, where the prover corresponds with PROVE and the verifier corresponds with VERIFY plus D_{s-1}. The relation corresponds with D_s; in particular, for field $\mathbb{F}$ and depth parameter $s \in [d_s]$, define the indexed language

$$\mathscr{L}(\mathbb{F}, s) = \{(\mathtt{i}, (e, x, \mathbf{f})) : \exists \mathbf{g} \in \mathbf{C}, \Delta(\mathbf{f}, \mathbf{g}) \leq s\delta, p(x, \hat{\mathbf{g}}) = e\}.$$

Here, $\mathtt{i}$ is an index of $\mathscr{R}(\mathbb{F})$ from which p and $\mathbf{C}$ are implicitly derived. We construct a public-coin interactive oracle proof IOP for the multi-instance language

$$\mathscr{L}(\mathbb{F}, s, m) = \{(\mathtt{i}, [e_i, x_i, \mathbf{f}_i]_{i=1}^{m}) : \forall i \in [m], (\mathtt{i}, (e_i, x_i, \mathbf{f}_i)) \in \mathscr{L}(\mathbb{F}, s)\}.$$

Note that all algorithms in IOP implicitly take as input the following protocol parameters: field $\mathbb{F}$, depth parameter s, arity m, and spot check parameter t. Since we are only interested in proving soundness, we omit the prover's description (it essentially matches the accumulation prover). The protocol is as follows.

1. The prover sends $q \in \mathbb{F}[X]$, a polynomial of degree at most $d(m-1) - m$.
2. The verifier uniformly samples a field element $\alpha \in \mathbb{F}$.
3. The prover sends $(e, x, \mathbf{f})$.
4. For each $j \in [\mu]$, the verifier uniformly samples a query set $\mathcal{Q}^{(j)} \subset L^{(j)}$ of size t.

The verifier then performs two sets of checks. The first set corresponds with the accumulation verifier:

- Verify that $e = v(\alpha) \cdot q(\alpha) + \sum_i L_i(\alpha) \cdot e_i$.
- Verify that $x = \sum_i L_i(\alpha) \cdot x_i$.
- Verify that $\mathbf{f}[\mathbf{Q}] = \sum_i L_i(\alpha) \cdot \mathbf{f}_i[\mathbf{Q}]$.

The second set corresponds with the decider:

- Find $\mathbf{g} \in \mathbf{C}$ by decoding $\mathbf{f}$. If no such codewords exist, reject.
- Verify that $\Delta(\mathbf{f}, \mathbf{g}) \leq (s-1)\delta$.
- Verify that $p(x, \hat{\mathbf{g}}) = e$.

Theorem 6.1. IOP *has round-by-round soundness error*

$$\epsilon_{\mathsf{rbr}}(\mathbb{F}, m, t, n) = \max\left(\frac{n(m-1)}{|\mathbb{F}|}, \frac{d(m-1)}{|\mathbb{F}|}, (1-\delta)^t\right),$$

where $n = \max_j \ell^{(j)}/R$ *is the maximum blocklength.*

Proof. We define a doomed set $\mathcal{D}$ as follows.

- $([e_i, x_i, \mathbf{f}_i]_{i=1}^m, \varnothing) \in \mathcal{D}$ if the instance is not in the language, i.e. there exists $i \in [m]$ such that for all $\mathbf{g} \in \mathbf{C}$, $\Delta(\mathbf{f}_i, \mathbf{g}) > s\delta$ or $p(x_i, \hat{\mathbf{g}}) \neq e_i$. Here, $\varnothing$ denotes the empty transcript.
- $([e_i, x_i, \mathbf{f}_i]_{i=1}^m, q|\alpha) \in \mathcal{D}$ if for all $\mathbf{g} \in \mathbf{C}$,

$$\Delta\left(\sum_{i=1}^m L_{i,H}(\alpha) \cdot \mathbf{f}_i, \mathbf{g}\right) > s\delta \quad \text{or} \quad p(x, \hat{\mathbf{g}}) \neq v_H(\alpha) \cdot q(\alpha) + \sum_{i=1}^m L_{i,H}(\alpha) \cdot e_i.$$

- $([e_i, x_i, \mathbf{f}_i]_{i=1}^m, q|\alpha|(e, x, \mathbf{f})|\mathbf{Q}) \in \mathcal{D}$ if the verifier rejects the transcript.

The statement follows from Claims 6.2 and 6.3. □

Claim 6.2. *Suppose* $([e_i, x_i, \mathbf{f}_i]_{i=1}^m, \varnothing)$ *is in the doomed set. Then*

$$\Pr_\alpha[(e_i, x_i, \mathbf{f}_i)_{i=1}^m, q|\alpha) \notin \mathcal{D}] \leq \max\left(\frac{n(m-1)}{|\mathbb{F}|}, \frac{d(m-1)}{|\mathbb{F}|}\right).$$

Proof. Suppose not. Let $\epsilon_1 = n(m-1)/|\mathbb{F}|, \epsilon_2 = d(m-1)/|\mathbb{F}|$, and define the events

$$E_1(\mathbf{g}, \alpha) : \Delta\left(\sum_{i=1}^m L_{i,H}(\alpha) \cdot \mathbf{f}_i, \mathbf{g}\right) \leq s\delta,$$

$$E_2(\mathbf{g}, \alpha) : p(x, \hat{\mathbf{g}}) = v_H(\alpha) \cdot q(\alpha) + \sum_{i=1}^m L_{i,H}(\alpha) \cdot e_i.$$

Our supposition is that $\Pr_\alpha[\exists \mathbf{g} \in \mathbf{C}, E_1(\mathbf{g}, \alpha) \wedge E_2(\mathbf{g}, \alpha)] > \max(\epsilon_1, \epsilon_2)$. It follows that

$$\Pr_\alpha[E_1(\mathbf{g}, \alpha)] = \Pr_\alpha\left[\forall j \in [\mu], \Delta\left(\sum_{i=1}^m L_{i,H}(\alpha) \cdot \mathbf{f}_i^{(j)}, C^{(j)}\right)\right] > \epsilon_1.$$

For each j, by Lemma 3.4, there exist codewords $g_1^{(j)}, \ldots, g_m^{(j)} \in C^{(j)}$ and subdomain $L' \subset L^{(j)}, |L'|/|L^{(j)}| \geq 1 - s\delta$, such that, for all i, $f_i^{(j)}$ and $g_i^{(j)}$ agree on L'. This implies the following.

- $\Delta(\mathbf{f}_i, \mathbf{g}_i) \leq s\delta$. Since we start in the doomed set, this implies that, for some i, $p(x_i, \hat{\mathbf{g}}_i) \neq e_i$. Hence, the degree $d(m-1)$ polynomial

$$z(X) = p\left(\sum_{i=1}^m L_{i,H}(X) \cdot (x_i, \hat{\mathbf{g}}_i)\right) - v_H(X) \cdot q(X) - \sum_{i=1}^m L_{i,H} \cdot e_i$$

  is non-zero. By the Schwartz-Zippel lemma, $\Pr_\alpha[z(\alpha) = 0] \leq \epsilon_2$.

- For all $\alpha \in \mathbb{F}$, $\Delta(\sum_i L_i(\alpha) \cdot \mathbf{f}_i, \sum_i L_i(\alpha) \cdot \mathbf{g}_i) \leq s\delta$. In other words, $E_1(\mathbf{g}, \alpha)$ holds for $\mathbf{g} = \mathbf{g}(\alpha) = \sum_i L_i(\alpha) \cdot \mathbf{g}_i$; moreover, since $s\delta$ is smaller than the unique decoding radius, this is the only satisfying assignment. It follows that $\Pr_\alpha[E_2(\mathbf{g}(\alpha), \alpha)] > \epsilon_2$.

Notice that the events $z(\alpha) = 0$ and $E_2(\mathbf{g}(\alpha), \alpha)$ are equivalent. Thus, we have shown a contradiction. □

Claim 6.3. *Suppose* $([e_i, x_i, \mathbf{f}_i]_{i=1}^m, q|\alpha)$ *is in the doomed set. Then*

$$\Pr_\alpha[((e_i, x_i, \mathbf{f}_i)_{i=1}^m, q|\alpha|(e, x, \mathbf{f})|\mathbf{Q}) \notin \mathcal{D}] \leq (1-\delta)^t.$$

Proof. In order for the verifier to accept, $\mathbf{f}$ must decode to codewords $\mathbf{g} \in \mathbf{C}, \Delta(\mathbf{f}, \mathbf{g}) \leq (s-1)\delta$ such that $p(x, \hat{\mathbf{g}}) = e = v(\alpha) \cdot q(\alpha) + \sum_i L_i(\alpha) \cdot e_i$. Since we start in the doomed set, this implies that, for some j, $\Delta(\sum_i L_i(\alpha) \cdot f_i^{(j)}, g^{(j)}) > s\delta$. Hence, $\Delta(\sum_i L_i(\alpha) \cdot f_i^{(j)}, f^{(j)}) > \delta$. The probability that the vectors are consistent at a random index is $1 - \delta$, and the claim follows. □

From IOP to Accumulation. We apply the BCS transformation [BCS16] to get a non-interactive argument $(\mathbf{P}, \mathbf{V})$ which matches our accumulation scheme. At a high level, the transformation replaces each IOP message with a vector commitment, and each challenge with a query to the random oracle. The prover then includes partial openings to all of the verifier's oracle accesses in the proof.

However, this does not immediately work. The most glaring issue is that the accumulation verifier does not have full access to the IOP instance $[e_i, x_i, \mathbf{f}_i]_{i=1}^m$. We will instead treat the IOP instance as an oracle, in the sense that $\mathbf{P}$ outputs a vector commitment to some instance and provides partial openings in the proof. Recall that the BCS transformation uses Valiant's extractor [Val08] to extract IOP messages from each vector commitment in the proof. Similarly, we should be able to extract an instance from the proof, which is valid if $\mathbf{V}$ accepts. Listed below are some additional modifications to the BCS transformation that we require. These are relatively minor, and follow from a close reading of [BCS16].

1. Our IOP prover sends certain values which will always be read by the verifier, namely q, e, and x. Therefore, $\mathbf{P}$ should include these values directly in the proof. Similarly, $\mathbf{P}$ should output parts of the instance without committing, namely $[e_i, x_i]_{i=1}^m$.

2. Our IOP prover sends many codewords $f^{(1)}, \ldots, f^{(\mu)}$ in the same round. Therefore, $\mathbf{P}$ should output separate commitments for each codeword. Similarly, $\mathbf{P}$ should output separate commitments for each codeword $f_i^{(j)}$ in the instance.

3. Our IOP verifier accesses the codewords $\mathbf{f}$ twice: first for the spot check, and second for decoding to $\mathbf{g}$. Since these checks will be separated across the accumulation verifier and decider, $\mathbf{P}$ should send separate openings for each access.

4. Suppose (a potentially malicious) $\mathbf{P}$ provides a partial opening which does not contain one of the locations in the query set. Instead of rejecting, $\mathbf{V}$ should fill any missing locations with a default symbol $\perp$ using $\mathsf{VC.Answer}$. This affords more flexibility to the IOP verifier, in particular when it attempts to decode $\mathbf{f}$.

Items 1 and 2 guarantees that the committed IOP instance can be parsed as m accumulator instances $[\mathsf{acc}_i.\mathbb{x}]_{i=1}^m$. Items 1 to 3 guarantee that a proof output by $\mathbf{P}$ can be parsed as an accumulator acc and accumulation proof pf. Items 3 and 4 guarantees that $\mathbf{V}$ can be decomposed into $\textsc{Verify}$ and D_{s-1}. Our requirements are formally summarized in Claim 6.4.

Claim 6.4. *There exists a transformation* BCS *such that* $\mathsf{BCS}[\mathsf{IOP}, \mathsf{VC}] = (\mathbf{P}, \mathbf{V})$ *satisfies the following soundness property. There exists a polynomial time extractor* $\mathbf{E}$ *such that for every choice of protocol parameters* $(\mathbb{F}, s, m, t)$*, polynomial time adversary* $\tilde{\mathbf{P}}$*, and auxiliary input distribution* $\mathcal{D}$*,*

$$\Pr\left[\begin{array}{c} \mathbf{V}^{\rho}(\mathsf{vp}, \mathbb{i}, \bar{\mathbb{x}}, \pi) = 1 \\ \Downarrow \\ (\mathbb{i}, [e_i, x_i, \mathbf{f}_i]_{i=1}^m) \in \mathscr{L}(\mathbb{F}, s, m) \end{array} \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{vp} \leftarrow \mathsf{VC.Setup}^{\rho_H}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ (\mathbb{i}, \bar{\mathbb{x}} = [e_i, x_i, \vec{\mathsf{cm}}_i]_{i=1}^m, \pi; r) \leftarrow \tilde{\mathbf{P}}^{\rho}(\mathsf{vp}, \mathsf{ai}) \\ [\vec{\mathsf{op}}_i]_{i=1}^m \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}, \rho}(\mathsf{vp}, \mathsf{ai}, r) \\ \forall i \in [m], \mathbf{f}_i \leftarrow \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}_i, \vec{\mathsf{op}}_i) \end{array} \right]$$

is negligibly close to $1 - \epsilon_{\mathsf{rbr}}(\mathbb{F}, m, t, n) \cdot Q$.[8] *Here,* n *is an upper bound on the blocklengths derived from any index output by* $\tilde{\mathbf{P}}$*, and* Q *is an upper bound on the number of random oracle queries made by* $\tilde{\mathbf{P}}$*. The extractor implicitly receives the protocol parameters as input. Additionally,* $\mathbf{V}$ *can be implemented as follows.*

$\mathbf{V}^{\rho}(\mathsf{vp}, \mathbb{i}, \bar{\mathbb{x}} = [e_i, x_i, \vec{\mathsf{cm}}_i]_{i=1}^m, \pi = (q, (e, x, \vec{\mathsf{cm}}), (\vec{\mathsf{op}}_1, \ldots, \vec{\mathsf{op}}_m, \vec{\mathsf{op}}, \vec{\mathsf{op}}')))$*:*

1. *Let* $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^{\rho}(\mathsf{pp}_{\mathsf{AS}} = t, \mathsf{pp} = (\mathbb{F}, \mathsf{vp}), \mathsf{i}_{\Phi} = \mathbb{i})$.
2. *Verify that* $\textsc{Verify}^{\rho}(\mathsf{avk}, [e_i, x_i, \vec{\mathsf{cm}}_i]_{i=1}^m, (e, x, \vec{\mathsf{cm}}), (q, [\vec{\mathsf{op}}_i]_{i=1}^m, \vec{\mathsf{op}}))$ *accepts.*
3. *Verify that* $\mathrm{D}_{s-1}^{\rho}(\mathsf{dk}, (e, x, \vec{\mathsf{cm}}), \vec{\mathsf{op}}')$ *accepts.*

Theorem 6.5. AS *has bounded-depth knowledge soundness for depth* d_s*.*

Proof. Let $\tilde{\mathrm{P}}$ be an adversary and $\mathcal{D}$ be an auxiliary input distribution for the accumulation scheme. Let $\mathbf{E}$ be the BCS extractor. We first construct an auxiliary input distribution $\mathcal{D}'$ and adversary $\tilde{\mathbf{P}}$ for $\mathsf{BCS}[\mathsf{IOP}, \mathsf{VC}]$.

$\mathcal{D}'(1^\lambda)$:

1. Sample $\mathsf{ai} \leftarrow \mathcal{D}(1^\lambda)$.
2. Sample t according to $\mathrm{G}(1^\lambda)$.
3. Sample $\mathbb{F}$ according to $\mathcal{G}(1^\lambda)$.
4. Output $\mathsf{ai}' = (\mathsf{ai}, \mathbb{F}, t)$.

$\tilde{\mathbf{P}}^{\rho}(\mathsf{vp}, \mathsf{ai}' = (\mathsf{ai}, \mathbb{F}, t))$:

1. Let $(\mathbb{i}, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2}, \mathsf{acc}, \mathsf{pf} = (q, [\vec{\mathsf{op}}_i]_{i=1}^m, \vec{\mathsf{op}})) \leftarrow \tilde{\mathrm{P}}^{\rho}(\mathsf{pp}_{\mathsf{AS}} = t, \mathsf{pp} = (\mathbb{F}, \mathsf{vp}), \mathsf{ai})$.
2. Query $\tau \leftarrow \rho_H(\mathbb{i})$.
3. Let $\bar{\mathbb{x}} \leftarrow \textsc{Cast}^{\rho}(\tau, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2})$.
4. Output $\mathbb{i}$, $\bar{\mathbb{x}}$, and $\pi = (q, \mathsf{acc}.\mathbb{x}, (\vec{\mathsf{op}}_1, \ldots, \vec{\mathsf{op}}_m, \vec{\mathsf{op}}, \mathsf{acc}.\mathbb{w}))$.

By Claim 6.4, the following is negligibly close to $1 - \epsilon_{\mathsf{rbr}}(\mathbb{F}, m, t, n) \cdot Q$:

$$\Pr\left[\begin{array}{c} \mathbf{V}^{\rho}(\mathsf{vp}, \mathbb{i}, \bar{\mathbb{x}}, \pi) = 1 \\ \Downarrow \\ (\mathbb{i}, [e_i, x_i, \mathbf{f}_i]_{i=1}^m) \in \mathscr{L}(\mathbb{F}, s, m) \end{array} \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{vp} \leftarrow \mathsf{VC.Setup}^{\rho_H}(1^\lambda) \\ \mathsf{ai}' \leftarrow \mathcal{D}'(1^\lambda) \\ (\mathbb{i}, \bar{\mathbb{x}} = [e_i, x_i, \vec{\mathsf{cm}}_i]_{i=1}^m, \pi; r) \leftarrow \tilde{\mathbf{P}}^{\rho}(\mathsf{vp}, \mathsf{ai}') \\ [\vec{\mathsf{op}}_i]_{i=1}^m \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}, \rho}(\mathsf{vp}, \mathsf{ai}', r) \\ \forall i \in [m], \mathbf{f}_i \leftarrow \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}_i, \vec{\mathsf{op}}_i) \end{array} \right]$$

Since the adversary is polynomial-size, we have $n = \mathrm{poly}(\lambda)$ and $Q = \mathrm{poly}(\lambda)$. Setting $|\mathbb{F}| = \Omega(\lambda)$, $m = \mathrm{poly}(\lambda)$, and $t = \Omega(\lambda)$, we find that the probability is negligibly close to 1. Finally, we show that this

[8] If an IOP has round-by-round soundness error ϵ_{rbr}, then it has state-restoration soundness error $\epsilon_{\mathsf{rbr}} \cdot Q$ against a Q-round prover [Can+19; Hol19; COS20].

probabilistic experiment is essentially equivalent to that of the accumulation scheme's knowledge soundness guarantee. Notice that calling VC.Answer and testing language membership corresponds with running the relaxed decider; rewriting, we get

$$\Pr\left[\begin{array}{c|r} \mathbf{V}^{\rho}(\mathsf{vp}, \mathbb{i}, [e_i, x_i, \vec{\mathsf{cm}}_i]_{i=1}^{m}, \pi) = 1 \\ \Downarrow \\ \forall i \in [m], \mathrm{D}_s^{\rho}(\mathsf{dk}, (e_i, x_i, \vec{\mathsf{cm}}_i), \vec{\mathsf{op}}_i) = 1 \end{array} & \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{vp} \leftarrow \mathsf{VC.Setup}^{\rho_H}(1^\lambda) \\ \mathsf{ai}' \leftarrow \mathcal{D}'(1^\lambda) \\ (\mathbb{i}, [e_i, x_i, \vec{\mathsf{cm}}_i]_{i=1}^{m}, \pi; r) \leftarrow \tilde{\mathbf{P}}^{\rho}(\mathsf{vp}, \mathsf{ai}') \\ [\vec{\mathsf{op}}_i]_{i=1}^{m} \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}, \rho}(\mathsf{vp}, \mathsf{ai}', r) \end{array}\right]$$

Since $\tilde{\mathbf{P}}$ does not use any additional randomness, the randomness output by $\tilde{\mathbf{P}}$ is the same as the randomness output by $\tilde{\mathrm{P}}$. Unwrapping the implementations of $\tilde{\mathbf{P}}$, $\mathcal{D}'$, and $\mathbf{V}$, we get

$$\Pr\left[\begin{array}{c|r} \begin{array}{c} \textsc{Verify}^{\rho}(\mathsf{avk}, [(e_i, x_i, \vec{\mathsf{cm}}_i)]_{i=1}^{m}, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1 \\ \mathrm{D}_{s-1}^{\rho}(\mathsf{dk}, \mathsf{acc}) \\ \Downarrow \\ \forall i \in [m], \mathrm{D}_s^{\rho}(\mathsf{dk}, (e_i, x_i, \vec{\mathsf{cm}}_i), \vec{\mathsf{op}}_i) = 1 \end{array} & \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{G}(1^\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ (\mathbb{i}, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2}, \mathsf{acc}, \mathsf{pf}; r) \leftarrow \tilde{\mathrm{P}}^{\rho}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}) \\ \tau \leftarrow \rho_H(\mathbb{i}) \\ [(e_i, x_i, \vec{\mathsf{cm}}_i)]_{i=1}^{m} \leftarrow \textsc{Cast}^{\rho}(\tau, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2}) \\ [\vec{\mathsf{op}}_i]_{i=1}^{m} \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}, \rho}(\mathsf{vp}, (\mathsf{ai}, \mathbb{F}, t), r) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^{\rho}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathbb{i}) \end{array}\right]$$

Notice that if the verification helper accepts the cast inputs, then the accumulation verifier accepts the original inputs. Rewriting, we get

$$\Pr\left[\begin{array}{c|r} \begin{array}{c} \mathrm{V}^{\rho}(\mathsf{avk}, [\mathsf{qx}]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2}, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1 \\ \mathrm{D}_{s-1}^{\rho}(\mathsf{dk}, \mathsf{acc}) \\ \Downarrow \\ \forall i \in [m], \mathrm{D}_s^{\rho}(\mathsf{dk}, (e_i, x_i, \vec{\mathsf{cm}}_i), \vec{\mathsf{op}}_i) = 1 \end{array} & \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{G}(1^\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ (\mathbb{i}, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2}, \mathsf{acc}, \mathsf{pf}; r) \leftarrow \tilde{\mathrm{P}}^{\rho}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}) \\ \tau \leftarrow \rho_H(\mathbb{i}) \\ [(e_i, x_i, \vec{\mathsf{cm}}_i)]_{i=1}^{m} \leftarrow \textsc{Cast}^{\rho}(\tau, [\mathsf{qx}_i)]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2}) \\ [\vec{\mathsf{op}}_i]_{i=1}^{m} \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}, \rho}(\mathsf{vp}, (\mathsf{ai}, \mathbb{F}, t), r) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^{\rho}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}_{\Phi}, \mathbb{i}) \end{array}\right]$$

Similarly, if the relaxed decider accepts the cast inputs, then the relaxed verifier accepts the original inputs. Hence, the following probability is negligibly close to 1:

$$\Pr\left[\begin{array}{c|r} \begin{array}{c} \mathrm{V}^{\rho}(\mathsf{avk}, [\mathsf{qx}]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2}, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1 \\ \mathrm{D}_{s-1}^{\rho}(\mathsf{dk}, \mathsf{acc}) \\ \Downarrow \\ \forall i \in [m_1], \hat{\mathcal{V}}^{\rho}(\mathsf{pp}, \mathbb{i}, \mathsf{qx}_i, \mathsf{qw}_i) = 1 \\ \forall i \in [m_2], \mathrm{D}_s^{\rho}(\mathsf{dk}, \mathsf{acc}) = 1 \end{array} & \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{G}(1^\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ (\mathbb{i}, [\mathsf{qx}_i]_{i=1}^{m_1}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^{m_2}, \mathsf{acc}, \mathsf{pf}; r) \leftarrow \tilde{\mathrm{P}}^{\rho}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}) \\ [\vec{\mathsf{op}}_i]_{i=1}^{m} \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}, \rho}(\mathsf{vp}, (\mathsf{ai}, \mathbb{F}, t), r) \\ [\mathsf{qw}_1, \ldots, \mathsf{qw}_{m_1}, \mathsf{acc}_1.\mathbb{w}, \ldots, \mathsf{acc}_{m_2}.\mathbb{w}] = [\vec{\mathsf{op}}_i]_{i=1}^{m} \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^{\rho}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathbb{i}) \end{array}\right]$$

We conclude that the following accumulation extractor satisfies knowledge soundness.

$\mathrm{E}^{\tilde{\mathrm{P}}, \rho}(\mathsf{pp} = t, \mathsf{pp}_{\Phi} = (\mathbb{F}, \mathsf{vp}), \mathsf{ai}, r)$:

1. Let $[\vec{\mathsf{op}}_i]_{i=1}^{m} \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}, \rho}(\mathsf{vp}, (\mathsf{ai}, \mathbb{F}, t), r)$, where $\mathbf{E}$ implicitly receives $(\mathbb{F}, s, m, t)$ as input.
2. Parse $[\mathsf{qw}_1, \ldots, \mathsf{qw}_{m_1}, \mathsf{acc}_1.\mathbb{w}, \ldots, \mathsf{acc}_{m_2}.\mathbb{w}] = [\vec{\mathsf{op}}_i]_{i=1}^{m}$.
3. Output $[\mathsf{qw}_i]_{i=1}^{m_1}$ and $[\mathsf{acc}_i.\mathbb{w}]_{i=1}^{m_2}$.

□

6.4 Using arbitrary linear codes

At first glance, our accumulation scheme does not use any special properties of the Reed–Solomon code. Because other codes might have desirable properties, e.g. linear-time encoding, this motivates the following question: can we instantiate it with an arbitrary linear code, so long as it has good distance? Recall that the accumulator's vectors are a random linear combination of the input vectors. In particular, the coefficients are Lagrange evaluations of a random field element; this is necessary for compressing the polynomial evaluation claims. Unfortunately, existing proximity gaps for arbitrary linear codes [RVW13; AHIV17; DP23a] do not support this specific distribution of coefficients.

Separating the proximity claim. In some sense, our construction accumulates two distinct claims for the same vector: a polynomial evaluation claim, and a proximity claim. To overcome the foregoing issue, we modify our accumulation scheme to separate these claims. Specifically, each accumulator now holds *two* codewords: the first is a linear combination using Lagrange coefficients, and the second is a linear combination using proximity gap coefficients. Accordingly, the accumulation verifier uses the first codeword to compress evaluation claims, and the second codeword to maintain proximity. This will be roughly twice as expensive as before.

Definition 6.6 (proximity gaps for linear codes)**.** *Let C be a linear code with relative distance d and blocklength n. Let $(r_1, \ldots, r_\ell) \leftarrow \mathsf{Coeffs}(\mathbf{\Gamma})$ be a function that takes in randomness $\mathbf{\Gamma}$. We say that C has a proximity gap with respect to distribution Coeffs, error $0 \leq \varepsilon(\ell) \leq 1$, slack γ, relative error bound $e < d/2$, if the following holds:*

For any $\delta < e$ and vectors $u_1, \ldots, u_\ell \in \mathbb{F}^n$, if

$$\Pr_{\mathbf{\Gamma}}\left[\Delta\left(\sum_{i=1}^{\ell} r_i \cdot u_i, C\right) \leq \delta \;:\; (r_1, \ldots, r_\ell) \leftarrow \mathsf{Coeffs}(\mathbf{\Gamma})\right] > \varepsilon(\ell)$$

Then, for all $(r_1, \ldots, r_\ell)$ in the support of the distribution $\mathsf{Coeffs}(\mathbf{\Gamma})$, we must have

$$\Delta\left(\sum_{i=1}^{\ell} r_i \cdot u_i, C\right) \leq \delta.$$

and furthermore there exists $v_1, \ldots, v_\ell \in C$ such that for all $(r_1, \ldots, r_\ell)$ in the support,

$$\Delta\left(\sum_{i=1}^{\ell} r_i \cdot u_i, \sum_{i=1}^{\ell} r_i \cdot v_i\right) \leq \delta,$$

and in fact $|\{i \in [n] : (u_{1,i}, \ldots, u_{\ell,i}) \neq (v_{1,i}, \ldots, v_{\ell,i})\}| \leq \gamma\delta n$.

Construction. We use linear codes with efficient decoding up to $e = O(1)$, the error bound of the proximity gap. Fix $\delta = e/\sum_{i=0}^{d_s-1} \gamma^i$, where γ is the slack of the proximity gap. The modified construction is shown below; all changes are highlighted in blue.

$\textsc{CastInput}^{\rho}(\tau, \mathsf{qx} = (\mathbb{x}, \vec{\mathsf{cm}}))$:

1. For each $j \in [\mu]$, query $r^{(j)} \leftarrow \rho_{\mathsf{ARG}}(\tau, \mathbb{x}, \mathsf{cm}^{(1)}, \ldots, \mathsf{cm}^{(j)})$.
2. Collect $\mathbb{x}$ and $\mathbf{r}$ into a vector x.
3. Output $\mathsf{acc}.\mathbb{x} = (0, x, \vec{\mathsf{cm}}, \vec{\mathsf{cm}}')$, where $\vec{\mathsf{cm}}'$ are commitments to zero vectors.

$\text{PROVE}^{\rho}(\mathsf{apk}, [e_i, x_i, \vec{\mathsf{cm}}_i, \vec{\mathsf{aux}}_i, \vec{\mathsf{cm}}'_i, \vec{\mathsf{aux}}'_i]_{i=1}^m)$:

1. For each $i \in [m]$, let $\mathbf{f}_i \leftarrow \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}_i, \vec{\mathsf{aux}}_i)$, and $\mathbf{f}'_i \leftarrow \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}'_i, \vec{\mathsf{aux}}'_i)$.
2. Compute the univariate polynomial $q \in \mathbb{F}[X]$ of degree at most $d(m-1) - m$ such that

$$p\left(\sum_{i=1}^{m} L_{i,H}(X) \cdot (x_i, \hat{\mathbf{f}}_i)\right) = v_H(X) \cdot q(X) + \sum_{i=1}^{n} L_{i,H}(X) \cdot e_i.$$

3. Query $\alpha, \mathbf{\Gamma} \leftarrow \rho_{\mathsf{AS}}(\tau, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, q)$, where $\alpha \in \mathbb{F}$ and $\mathbf{\Gamma}$ are sampled uniformly.
4. Compute $e = v(\alpha) \cdot q(\alpha) + \sum_i L_i(\alpha) \cdot e_i$.
5. Compute $(x, \hat{\mathbf{f}}) = \sum_i L_i(\alpha) \cdot (x_i, \hat{\mathbf{f}}_i)$.
6. Let $(r_1, \ldots, r_m, r'_1, \ldots, r'_m) \leftarrow \mathsf{Coeffs}(\mathbf{\Gamma})$. Compute $\mathbf{f}' = \sum_{i=1}^m r_i \cdot \mathbf{f}_i + r'_i \cdot \mathbf{f}'_i$.
7. Let $(\vec{\mathsf{cm}}, \vec{\mathsf{aux}}) \leftarrow \mathsf{VC.Commit}^{\rho_H}(\mathsf{vp}, \mathbf{f})$, and $(\vec{\mathsf{cm}}', \vec{\mathsf{aux}}') \leftarrow \mathsf{VC.Commit}^{\rho_H}(\mathsf{vp}, \mathbf{f}')$.
8. Set $\mathsf{acc}.\mathbb{x} = (e, x, \vec{\mathsf{cm}}, \vec{\mathsf{cm}}')$ and $\mathsf{acc}.\mathbb{w} = (\vec{\mathsf{aux}}, \vec{\mathsf{aux}}')$.
9. Query $\mathbf{Q} \leftarrow \rho_{\mathsf{AS}}(\tau, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, q, \mathsf{acc}.\mathbb{x})$, where $\mathcal{Q}^{(j)}$ is a uniformly sampled t-sized subset of $[n]$.
10. For each $i \in [m]$, let $\vec{\mathsf{op}}_i \leftarrow \mathsf{VC.Open}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{aux}}_i, \mathbf{Q})$, and $\vec{\mathsf{op}}'_i \leftarrow \mathsf{VC.Open}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{aux}}'_i, \mathbf{Q})$.
11. Let $\vec{\mathsf{op}} \leftarrow \mathsf{VC.Open}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{aux}}, \mathbf{Q})$, and $\vec{\mathsf{op}}' \leftarrow \mathsf{VC.Open}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{aux}}', \mathbf{Q})$.
12. Output acc and $\mathsf{pf} = (q, [\vec{\mathsf{op}}_i, \vec{\mathsf{op}}'_i]_{i=1}^m, \vec{\mathsf{op}}, \vec{\mathsf{op}}')$.

$\text{VERIFY}^{\rho}(\mathsf{avk}, [e_i, x_i, \vec{\mathsf{cm}}_i, \vec{\mathsf{cm}}'_i]_{i=1}^m, (e, x, \vec{\mathsf{cm}}, \vec{\mathsf{cm}}'), (q, [\vec{\mathsf{op}}_i, \vec{\mathsf{op}}'_i]_{i=1}^m, \vec{\mathsf{op}}, \vec{\mathsf{op}}'))$:

1. Query $\alpha, \mathbf{\Gamma} \leftarrow \rho_{\mathsf{AS}}(\tau, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, q)$.
2. Query $\mathbf{Q} \leftarrow \rho_{\mathsf{AS}}(\tau, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, q, \mathsf{acc}.\mathbb{x})$.
3. For each $i \in [m]$, let $\mathbf{v}_i = \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}_i, \vec{\mathsf{op}}_i)$. If $\mathbf{v}_i[\mathbf{Q}]$ contains $\perp$, reject.
4. Let $\mathbf{v} = \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}, \vec{\mathsf{op}}, \mathbf{Q})$. If $\mathbf{v}[\mathbf{Q}]$ contains $\perp$, reject.
5. For each $i \in [m]$, let $\mathbf{v}'_i = \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}'_i, \vec{\mathsf{op}}'_i)$. If $\mathbf{v}'_i[\mathbf{Q}]$ contains $\perp$, reject.
6. Let $\mathbf{v}' = \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}', \vec{\mathsf{op}}', \mathbf{Q})$. If $\mathbf{v}'[\mathbf{Q}]$ contains $\perp$, reject.
7. Verify that $e = v_H(\alpha) \cdot q(\alpha) + \sum_{i=1}^n L_{i,H}(\alpha) \cdot e^{(i)}$.
8. Verify that $(x, \mathbf{v}) = \sum_{i=1}^m L_{i,H}(\alpha) \cdot (x_i, \mathbf{v}_i)$.
9. Let $(r_1, \ldots, r_m, r'_1, \ldots, r'_m) \leftarrow \mathsf{Coeffs}(\mathbf{\Gamma})$. Verify that $\mathbf{v}' = \sum_{i=1}^m r_i \cdot v_i + r'_i \cdot v'_i$.

$\text{D}_s^{\rho}(\mathsf{dk}, \mathsf{acc}.\mathbb{x} = (e, x, \vec{\mathsf{cm}}, \vec{\mathsf{cm}}'), \mathsf{acc}.\mathbb{w} = (\vec{\mathsf{aux}}, \vec{\mathsf{aux}}'))$:

1. Let $\mathbf{f} \leftarrow \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}, \vec{\mathsf{aux}})$.
2. Let $\mathbf{f}' \leftarrow \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}', \vec{\mathsf{aux}}')$. If $\Delta(\mathbf{f}', \mathbf{C}) > (\sum_{i=0}^{s-1} \gamma^i)\delta$, reject.
3. Find $\mathbf{g} \in \mathbf{C}, \Delta(\mathbf{f}, \mathbf{g}) \leq (\sum_{i=0}^{s-1} \gamma^i)\delta$ by decoding $\mathbf{f}$. If no such codeword exists, reject.
4. Accept if $p(x, \hat{\mathbf{g}}) = e$.

7 Optimizations

7.1 Batch commitments

The accumulation scheme in Section 6 enables accumulating m accumulators (or proofs) into one. Each accumulator consists of a short commitment, e.g. a Merkle tree, and a long witness, the vector inside the commitment. The most expensive step of the accumulation verifier requires opening a vector commitment, at k locations of the accumulator's codewords. Naively, this requires opening $m \cdot k$ Merkle proofs, which have total size $k \cdot m \cdot \log n$, where n is the size of one accumulator witness. We can optimize the construction by committing to *all* m codewords inside a *single* vector commitment. To do this we commit to a tuple of m entries, one per codeword, at each position of the vector commitment. This enables opening all m vectors at the same position using only a single Merkle proof. When performing the linearity check, we open the same position for all codewords. This batch accumulator construction, now only requires a single Merkle proof per opening. The total opening proof size is, thus, reduced from $k \cdot m \cdot \log n$ to just $k \cdot (m + \log n)$. This is a very significant saving, especially for larger values of m, which we use to build PCD trees with higher arity m and lower depth.

Batch commitments and PCD. Using batch commitments naively is incompatible with our PCD construction from Section 5.1. The reason is that each node containing an accumulator or proof is constructed independently, thus committing to m accumulator witnesses or proof witnesses in one commitment isn't possible. Fortunately, there are many instances of PCD where it is possible to still take advantage of batch commitments and batch accumulation. For example, consider the IVC construction from Section 7.2 with uniform m-ary PCD and accumulation trees. Instead of constructing the PCD tree node by node, we can construct m new PCD nodes and accumulators at the same time and use a batch commitment to commit to the accumulator and proof witnesses. More formally, this is a PCD tree where the predicate φ_{big} is m concatenations of the original PCD predicate φ_{small}. Each accumulator now is a single batch commitment with m witnesses and similarly, each proof consists of a batch proof with m witnesses. Each batch proof proves φ_{big} and the accumulation of m batch accumulators (m^2 witnesses) into a single output batch accumulator (m witnesses). More specifically, the m proof witnesses each prove one φ_{small} and the accumulation of a batch accumulator (m witnesses) into the same output batch accumulator (localized to one index). The construction is displayed in Figure 3. The big advantage is that the number of Merkle tree openings in the recursive circuit is reduced by a factor of m. The downside is that the PCD prover's memory (and the amount of data needed to construct a new PCD node), now consists of at least m^2 accumulator witness, a factor m more than in the construction without batch commitments.

7.2 Low-overhead IVC from accumulation

As mentioned in Remark 2.2, constructing polynomial-length IVC from bounded-depth PCD requires a tree-based strategy. Let us recall a simplified version of the construction from [Val08; BCCT13]. The idea is to use a PCD scheme for the following compliance predicate.

$\varphi_F(z, z_{\mathsf{loc}}, [z_i]_{i=1}^m)$:

1. *Leaf node.* If $z = (0, \mathsf{in}, \mathsf{out})$ and each $z_i = \bot$:
   - Parse $z_{\mathsf{loc}} = w$.
   - Check that $\mathsf{out} = F(\mathsf{in}, w)$.
2. *Internal node.* If $z = (k, \mathsf{in}, \mathsf{out})$ for some $k \in [d]$ and each $z_i = (k-1, \mathsf{in}_i, \mathsf{out}_i)$:
   - Check that $\mathsf{in} = \mathsf{in}_1$ and $\mathsf{out} = \mathsf{out}_m$.
   - Check that $\mathsf{out}_1 = \mathsf{in}_2, \mathsf{out}_2 = \mathsf{in}_3, \ldots, \mathsf{out}_{m-1} = \mathsf{in}_m$.

Suppose a transcript with output $(k, \mathsf{in}, \mathsf{out})$ is φ_F-compliant. Then out must be the result of m^k applications of F to in. Moreover, the leaf nodes of the transcript are labeled with the witness values. Using a PCD scheme for φ_F, we construct IVC for computations of length at most m^d as follows. The IVC prover begins with an empty m-ary tree of depth d. In each step, it adds a leaf node, which consists of the message $(0, \mathsf{in}, \mathsf{out})$ and a PCD proof σ. Whenever there exists a full set of m nodes in a layer, the IVC prover creates a parent node by running the PCD prover. This means that it only needs to keep track of the tree's "frontier," denoted τ, which consists of at most md nodes. At any step, the IVC verifier can check the computation by running the PCD verifier on each node in the frontier.

In more detail, let $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ be a PCD scheme for the class of compliance predicates $\{\varphi_F\}$. A full description of the IVC scheme is given below.

$\mathsf{IVC.Generate}(1^\lambda)$:
1. Sample $\mathbb{pp} \leftarrow \mathbb{G}(1^\lambda)$.
2. Output $\mathbb{pp}$.

$\mathsf{IVC.Index}(\mathbb{pp}, F)$:
1. Compute $(\mathsf{ipk}, \mathsf{ivk}) := \mathbb{I}(\mathbb{pp}, \varphi_F)$.
2. Output ipk and ivk.

$\mathsf{IVC.Prove}(\mathsf{ipk}, \tau, \mathsf{in}, w, \mathsf{out})$:
1. Generate a PCD proof $\sigma \leftarrow \mathbb{P}(\mathsf{ipk}, (0, \mathsf{in}, \mathsf{out}), w, [\bot]_{i=1}^m)$.
2. Append $(0, \mathsf{in}, \mathsf{out}, \sigma)$ to τ.
3. For each $k \in [d]$: if there are m nodes in the $(k-1)$-th layer of the frontier, i.e. τ contains $[k-1, \mathsf{in}_i, \mathsf{out}_i, \sigma_i]_{i=1}^m$:
   - Remove the nodes from τ.
   - Generate a PCD proof $\sigma \leftarrow \mathbb{P}(\mathsf{ipk}, (k, \mathsf{in}_1, \mathsf{out}_m), \bot, [(k-1, \mathsf{in}_i, \mathsf{out}_i), \sigma_i]_{i=1}^m)$.
   - Append $(k, \mathsf{in}_1, \mathsf{out}_m, \sigma)$ to τ.
4. Output τ.

$\mathsf{IVC.Verify}(\mathsf{ivk}, \tau, x, T)$:
1. If τ is empty, x is the initial value, and $T = 0$, accept.
2. Parse τ as a list of nodes $[k_j, \mathsf{in}_j, \mathsf{out}_j, \sigma_j]_{j=1}^\ell$.
3. For each $j \in [\ell]$, check that $\mathbb{V}(\mathsf{ivk}, (k_j, \mathsf{in}_j, \mathsf{out}_j), \sigma_j)$ accepts.
4. Check that in_1 is the initial value and $\mathsf{out}_\ell = x$.
5. Check that $\mathsf{out}_1 = \mathsf{in}_2, \mathsf{out}_2 = \mathsf{in}_3, \ldots, \mathsf{out}_{\ell-1} = \mathsf{in}_\ell$.
6. Check that $T = \sum_{j=1}^\ell m^{k_j}$.

This IVC scheme is somewhat wasteful, in the sense that the internal nodes of the tree do not use F. If we were to instantiate PCD with the construction from Section 5, then the internal NARK proofs would be proving F for nothing.

Accumulating separately. Our idea is to accumulate computations of F separately, and use PCD to verify the accumulations. In particular, let $\mathsf{ARG} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ be a NARK for the indexed relation $\mathscr{R} = \{(F, (x, x'), w) : x' = F(x, w)\}$. Let $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ be an accumulation scheme for ARG. For simplicity, we will assume that NARK proofs can be "cast" into accumulators (this is true for the construction in Section 6). Finally, let $\mathsf{PCD}' = (\mathbb{G}', \mathbb{I}', \mathbb{P}', \mathbb{V}')$ be a (not necessarily accumulation-based) PCD scheme for the class of compliance predicates $\{\varphi'_{\mathsf{avk}}\}$, which is defined below.

$\varphi'_{\mathsf{avk}}(z, z_{\mathsf{loc}}, [z_i]_{i=1}^m)$:

1. *Leaf node.* If $z = (0, \mathsf{in}, \mathsf{out}, \pi.\mathbb{x})$ and each $z_i = \bot$, accept.
2. *Internal node.* If $z = (k, \mathsf{in}, \mathsf{out}, \mathsf{acc}.\mathbb{x})$ and each $z_i = (k-1, \mathsf{in}_i, \mathsf{out}_i, \mathsf{acc}_i.\mathbb{x})$ for some $k \in [d]$:
   - If $k = 1$, check that each $\mathsf{acc}_i.\mathbb{x}$ is cast from some $\pi.\mathbb{x}$ with instance $(\mathsf{in}_i, \mathsf{out}_i)$.
   - Check that $\mathsf{in} = \mathsf{in}_1$ and $\mathsf{out} = \mathsf{out}_m$.
   - Check that $\mathsf{out}_1 = \mathsf{in}_2, \mathsf{out}_2 = \mathsf{in}_3, \ldots, \mathsf{out}_{m-1} = \mathsf{in}_m$.
   - Pare $z_{\mathsf{loc}} = \mathsf{pf}$.
   - Check that $\mathrm{V}(\mathsf{avk}, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$ accepts.

Similar to before, the IVC prover begins with an empty m-ary tree of depth d. In the T-th step, it adds a leaf node, which consists of the message $(0, \mathsf{in}, \mathsf{out})$, a NARK proof certifying the computation $\mathsf{out} = F(\mathsf{in}, w)$ for some w, and a PCD proof σ. Whenever there exists a full set of m nodes in a layer, the IVC prover creates a parent node by running the accumulation and PCD provers. A full description of the IVC scheme is given below.

$\mathsf{IVC.Generate}(1^\lambda)$:

1. Sample $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$.
2. Sample $\mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{G}(1^\lambda)$.
3. Sample $\mathbb{pp} \leftarrow \mathbb{G}'(1^\lambda)$.
4. Output $(\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}, \mathbb{pp})$.

$\mathsf{IVC.Index}((\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}, \mathbb{pp}), F)$:

1. Compute $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, F)$.
2. Compute $(\mathsf{ipk}, \mathsf{ivk}) := \mathbb{I}'(\mathbb{pp}, \varphi_{\mathsf{avk}})$.
3. Output $(\mathsf{pp}, F, \mathsf{apk}, \mathsf{ipk})$ and $(\mathsf{pp}, F, \mathsf{dk}, \mathsf{ivk})$.

$\mathsf{IVC.Prove}((\mathsf{pp}, F, \mathsf{apk}, \mathsf{ipk}), \tau, \mathsf{in}, w, \mathsf{out})$:

1. Generate a NARK proof $\pi \leftarrow \mathcal{P}(\mathsf{pp}, F, (\mathsf{in}, \mathsf{out}), w)$ and cast it into an accumulator acc.
2. Generate a dummy PCD proof $\sigma := \bot$.
3. Append $(0, \mathsf{in}, \mathsf{out}, \mathsf{acc}, \sigma)$ to τ.
4. For each $k = 1, \ldots, d$: if there are m nodes in the $(k-1)$-th layer of the frontier, i.e. τ contains $[k-1, \mathsf{in}_i, \mathsf{out}_i, \mathsf{acc}_i, \sigma_i]_{i=1}^m$:
   - Remove the nodes from τ.
   - Accumulate $(\mathsf{acc}, \mathsf{pf}) \leftarrow \mathrm{P}(\mathsf{apk}, [\mathsf{acc}_i]_{i=1}^m)$.
   - Generate a PCD proof $\sigma \leftarrow \mathbb{P}'(\mathsf{ipk}, (k, \mathsf{in}_1, \mathsf{out}_m, \mathsf{acc}.\mathbb{x}), \mathsf{pf}, [(k-1, \mathsf{in}_i, \mathsf{out}_i, \mathsf{acc}_i.\mathbb{x}), \sigma_i]_{i=1}^m)$.
   - Append $(k, \mathsf{in}_1, \mathsf{out}_m, \mathsf{acc}, \sigma)$ to τ.
5. Output τ.

$\mathsf{IVC.Verify}((\mathsf{pp}, F, \mathsf{dk}, \mathsf{ivk}), \tau, x, T)$:

1. If τ is empty, x is the initial value, and $T = 0$, accept.
2. Parse τ as a list of nodes $[k_j, \mathsf{in}_j, \mathsf{out}_j, \mathsf{acc}_j, \sigma_j]_{j=1}^\ell$.
3. For each $j \in [\ell]$, check that $\mathbb{V}'(\mathsf{ivk}, (k_j, \mathsf{in}_j, \mathsf{out}_j, \mathsf{acc}_j.\mathbb{x}), \sigma_j)$ accepts. If $k_j = 0$, recover π_j from acc_j and check that $\mathcal{V}(\mathsf{pp}, F, (\mathsf{in}_j, \mathsf{out}_j), \pi_j)$ accepts. Otherwise, check that $\mathrm{D}(\mathsf{dk}, \mathsf{acc}_j)$ accepts.
4. Check that in_1 is the initial value and $\mathsf{out}_\ell = x$.
5. Check that $\mathsf{out}_1 = \mathsf{in}_2, \mathsf{out}_2 = \mathsf{in}_3, \ldots, \mathsf{out}_{\ell-1} = \mathsf{in}_\ell$.
6. Accept if $T = \sum_{j=1}^\ell m^{k_j}$.

We sketch the knowledge soundness extraction strategy for the IVC scheme. First, run the PCD extractor on each node in the frontier to obtain all of the accumulator instances and proofs. Then, run the accumulation extractor at each layer of the tree to obtain accumulator witnesses. The PCD compliance predicate guarantees that the accumulators in the leaf layer are cast from NARK proofs. Run the NARK extractor on these proofs to obtain each step of the computation.

Efficiency. We say that an IVC scheme's proving cost is the amount of extra work the prover does, outside of the original computation. Let $S_d := \sum_{i=0}^{d} m^i$ be the number of nodes in a tree with height d and arity m. In the old construction, the overhead is generating a tree of S_d PCD proofs. In the new construction, the overhead is generating a tree of S_d accumulators and a tree of S_{d-1} PCD proofs.

Recall that in accumulation-based PCD, a PCD proof consists of a NARK proof and accumulator. The cost of generating a tree of S_d PCD proofs is roughly the cost of generating S_d NARK proofs, plus the cost of generating S_{d-1} accumulators (since the PCD prover outputs a dummy accumulator when there are no incoming edges). Let R be the PCD circuit which consists of the compliance predicate and accumulation verifier. We assume that the cost of generating a NARK proof or accumulator is roughly $|R|$. Hence, the cost of generating a tree of m^d PCD proofs is roughly $(S_d + S_{d-1}) \cdot |R|$.

In the old construction, the compliance predicate φ is dominated by the function F. The size of the circuit R is roughly $|F| + |\mathrm{V}_R|$, where V_R denotes the accumulation verifier for R. Hence, the IVC scheme's total cost is $(S_d + S_{d-1}) \cdot (|F| + |\mathrm{V}_R|)$.

In the new construction, the compliance predicate φ' is dominated by V_F, where V_F denotes the accumulation verifier for F. The size of the corresponding circuit R' is roughly $|\mathrm{V}_F| + |\mathrm{V}_{R'}|$, where $\mathrm{V}_{R'}$ denotes the accumulation verifier for R'. Hence, the IVC scheme's total cost is $S_d \cdot |F| + (S_{d-1} + S_{d-2}) \cdot (|\mathrm{V}_F| + |\mathrm{V}_{R'}|)$.

To compare these costs, notice that V_F is cheaper than V_R, since the circuit R contains F. Assuming F is sufficiently large, we can also reasonably expect that $\mathrm{V}_{R'}$ is cheaper than V_R. This means that the new construction's overhead is bounded by $S_d \cdot |F| + (S_{d-1} + S_{d-2}) \cdot 2|\mathrm{V}_R|$. Hence, the new construction reduces cost by at least $S_{d-1} \cdot |F| + (m-2)(S_{d-1} + S_{d-2}) \cdot |\mathrm{V}_R|$.

7.3 PCD composition

We show that combining our accumulation-based PCD scheme with a SNARK-based PCD scheme can yield a new PCD scheme that is more performant than either scheme on their own.

Recursion overhead is linear in tree depth. In Section 5 we show that for every constant depth d predicate there exists a PCD scheme. Note that this is a different order of quantifiers than prior unbounded accumulation or SNARK-based schemes which showed that there exists a PCD scheme for any constant depth d predicate. This change of quantifiers is not only interesting from a security point of view but also from an efficiency perspective. The reason is that in our scheme the PCDs recursive overhead is linearly dependent on d. The reason is that within each accumulation step, we prove that the output accumulator code is δ close to the input accumulator code. After d steps the distance is $d \cdot \delta$. Even if the final accumulator is a codeword we need to ensure that the accumulators at the leaves of the PCD tree are still within the unique decoding radius of the code. If d increases, we, therefore, decrease δ by increasing the spot-checking parameter in our accumulation scheme. The relationship is roughly linear, as we need to do the linearity check at $O(\lambda d)$ positions.

Reducing PCD depth. This relationship motivates the need for lower-depth PCD trees. One avenue is using higher arity accumulation/PCD trees. Another optimization combines two PCD schemes. Let $\mathsf{PCD}^{\mathsf{acc}}$ be a depth-bounded accumulation-based PCD scheme. Let $\mathsf{PCD}^{\mathsf{ARG}}$ be a SNARK-based PCD scheme (that

is not depth bounded). This second PCD scheme checks the original predicate φ as well as checking the PCD proof output by $\mathsf{PCD}^{\mathsf{acc}}$.

Let d^* be the maximum depth supported by $\mathsf{PCD}^{\mathsf{acc}}$ and m it's arity. We combine the schemes by first building PCD trees up to depth d^*. Then we use $\mathsf{PCD}^{\mathsf{ARG}}$ to combine the roots of these PCD schemes. Note that we only need to use $\mathsf{PCD}^{\mathsf{ARG}}$ every m^{d^*} PCD step. Even for relatively small values of d^* this is likely a marginal cost compared to the cost of running $\mathsf{PCD}^{\mathsf{acc}}$. In fact, we can even pick our parameters optimally. Assume that $\mathsf{PCD}^{\mathsf{ARG}}$ is c times as expensive as running $\mathsf{PCD}^{\mathsf{acc}}$ for a depth 1 predicate. Let n be the cost of running $\mathsf{PCD}^{\mathsf{acc}}$ for a depth 1 predicate. Since $\mathsf{PCD}^{\mathsf{acc}}$ is linear in the maximum supported depth, we have that the total PCD cost is proportional to $d^* \cdot n + \frac{c \cdot n}{2^{d^*}}$. Setting $d^* = \log c$, we get a cost of $\log c \cdot n$, significantly lower than either scheme on its own.

This optimization can be further used for code switching techniques, as the first PCD scheme could be used using a linear-time code and the SNARK-based PCD scheme could be using Fractal [COS20] and FRI [BBHR18] which are based on Reed–Solomon codes.

Compatibility. The optimization works for any depth-bounded PCD scheme with an arbitrary other PCD scheme. It is even possible to stack two accumulation-based schemes. For instance, one could use a fast linear-time encodable code for the first PCD and then use a Reed-Solomon code (which has longer encoding time but shorter codes) for the second PCD. The optimization is also fully compatible with both batch accumulators and the low-overhead IVC.

Acknowledgments

The third author was supported by NSF, DARPA, the Simons Foundation, UBRI, NTT Research, and the Stanford Future of Digital Currency Initiative (FDCI). Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

[AFK23] T. Attema, S. Fehr, and M. Klooß. "Fiat-Shamir Transformation of Multi-Round Interactive Proofs (Extended Version)". In: *Journal of Cryptology* 36.4 (2023), p. 36.

[AHIV17] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup". In: *Proceedings of the 24th ACM Conference on Computer and Communications Security*. CCS '17. 2017, pp. 2087–2104.

[BBHR18] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. "Fast Reed-Solomon Interactive Oracle Proofs of Proximity". In: *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming*. Vol. 107. ICALP '18. 2018, 14:1–14:17.

[BC23] B. Bünz and B. Chen. "Protostar: Generic Efficient Accumulation/Folding for Special-Sound Protocols". In: *Proceedings of the 29th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '23. 2023, pp. 77–110.

[BC24] D. Boneh and B. Chen. "LatticeFold: A Lattice-based Folding Scheme and its Applications to Succinct Proof Systems". Cryptology ePrint Archive, Report 2024/257. 2024.

[BCCT13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. "Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data". In: *Proceedings of the 45th ACM Symposium on the Theory of Computing*. STOC '13. 2013, pp. 111–120.

[BCG24] E. Boyle, R. Cohen, and A. Goel. "Breaking the $O(\sqrt{n})$-Bit Barrier: Byzantine Agreement with Polylog Bits Per Party". In: *Journal of Cryptology* 37.1 (2024), p. 2.

[BCIKS23] E. Ben-Sasson, D. Carmon, Y. Ishai, S. Kopparty, and S. Saraf. "Proximity Gaps for Reed-Solomon Codes". In: *Journal of the ACM* 70.5 (2023), 31:1–31:57.

[BCLMS21] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner. "Proof-Carrying Data Without Succinct Arguments". In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO '21. 2021, pp. 681–710.

[BCMS20] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. "Proof-Carrying Data from Accumulation Schemes". In: *Proceedings of the 18th Theory of Cryptography Conference*. TCC '20. 2020.

[BCS16] E. Ben-Sasson, A. Chiesa, and N. Spooner. "Interactive Oracle Proofs". In: *Proceedings of the 14th Theory of Cryptography Conference*. TCC '16-B. 2016, pp. 31–60.

[BCTV14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. "Scalable Zero Knowledge via Cycles of Elliptic Curves". In: *Proceedings of the 34th Annual International Cryptology Conference*. CRYPTO '14. 2014, pp. 276–294.

[BCTV17] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. "Scalable Zero Knowledge Via Cycles of Elliptic Curves". In: *Algorithmica* 79.4 (2017), pp. 1102–1160.

[BDFG21] D. Boneh, J. Drake, B. Fisch, and A. Gabizon. "Halo Infinite: Proof-Carrying Data from Additive Polynomial Commitments". In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO '21. 2021.

[BDFLSZ11] D. Boneh, Ö. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner, and M. Zhandry. "Random Oracles in a Quantum World". In: *Proceedings of the 17th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '11. 2011, pp. 41–69.

[BGH19] S. Bowe, J. Grigg, and D. Hopwood. "Halo: Recursive Proof Composition without a Trusted Setup". Cryptology ePrint Archive, Report 2019/1021. 2019.

[BMRS20] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro. "Coda: Decentralized Cryptocurrency at Scale". Cryptology ePrint Archive, Report 2020/352. 2020.

[Can+19] R. Canetti, Y. Chen, J. Holmgren, A. Lombardi, G. N. Rothblum, R. D. Rothblum, and D. Wichs. "Fiat–Shamir: from practice to theory". In: *Proceedings of the 51st Annual ACM Symposium on Theory of Computing*. STOC '19. 2019, pp. 1082–1090.

[CCDW20] W. Chen, A. Chiesa, E. Dauterman, and N. P. Ward. "Reducing Participation Costs via Incremental Verification for Ledger Systems". Cryptology ePrint Archive, Report 2020/1522. 2020.

[COS20] A. Chiesa, D. Ojha, and N. Spooner. "Fractal: Post-Quantum and Transparent Recursive Proofs from Holography". In: *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '20. 2020.

[CT10] A. Chiesa and E. Tromer. "Proof-Carrying Data and Hearsay Arguments from Signature Cards". In: *Proceedings of the 1st Symposium on Innovations in Computer Science*. ICS '10. 2010, pp. 310–331.

[CTV13] S. Chong, E. Tromer, and J. A. Vaughan. "Enforcing Language Semantics Using Proof-Carrying Data". Cryptology ePrint Archive, Report 2013/513. 2013.

[CTV15] A. Chiesa, E. Tromer, and M. Virza. "Cluster Computing in Zero Knowledge". In: *Proceedings of the 34th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '15. 2015, pp. 371–403.

[DP23a] B. E. Diamond and J. Posen. "Proximity Testing with Logarithmic Randomness". Cryptology ePrint Archive, Report 2023/630. 2023.

[DP23b] B. E. Diamond and J. Posen. "Succinct Arguments over Towers of Binary Fields". Cryptology ePrint Archive, Report 2023/1784. 2023.

[EG23] L. Eagen and A. Gabizon. "ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances". Cryptology ePrint Archive, Report 2023/1106. 2023.

[GW11] C. Gentry and D. Wichs. "Separating Succinct Non-Interactive Arguments From All Falsifiable Assumptions". In: *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*. STOC '11. 2011, pp. 99–108.

[Hol19] J. Holmgren. "On Round-By-Round Soundness and State Restoration Attacks". Cryptology ePrint Archive, Report 2019/1261. 2019.

[KB20] A. Kattis and J. Bonneau. "Proof of Necessary Work: Succinct State Verification with Fairness Guarantees". Cryptology ePrint Archive, Report 2020/190. 2020.

[KS23a] A. Kothapalli and S. Setty. "CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves". Cryptology ePrint Archive, Report 2023/1192. Aug. 2023.

[KS23b] A. Kothapalli and S. Setty. "HyperNova: Recursive arguments for customizable constraint systems". Cryptology ePrint Archive, Report 2023/573. Apr. 2023.

[KST22] A. Kothapalli, S. T. V. Setty, and I. Tzialla. "Nova: Recursive Zero-Knowledge Arguments from Folding Schemes". In: *Proceedings of the 42nd Annual International Cryptology Conference*. CRYPTO '22. 2022, pp. 359–388.

[Mina] O(1) Labs. "Mina Cryptocurrency". `minaprotocol.org`. 2020.

[NBS23] W. D. Nguyen, D. Boneh, and S. T. V. Setty. "Revisiting the Nova Proof System on a Cycle of Curves". In: *Proceedings of the 5th Conference on Advances in Financial Technologies*. AFT '23. 2023, 18:1–18:22.

[NT16] A. Naveh and E. Tromer. "PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations". In: *Proceedings of the 37th IEEE Symposium on Security and Privacy*. S&P '16. 2016, pp. 255–271.

[Ped92] T. P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Proceedings of the 11th Annual International Cryptology Conference*. CRYPTO '91. 1992, pp. 129–140.

[Pol] Polygon Zero Team. "Plonky2: Fast Recursive Arguments with PLONK and FRI".

[RRR21] O. Reingold, G. N. Rothblum, and R. D. Rothblum. "Constant-Round Interactive Proofs for Delegating Computation". In: *SIAM Journal on Computing* 50.3 (2021).

[RSM60] I. S. Reed, G. Solomon, and K. H. March. "Polynomial Codes Over Certain Finite Fields". In: *Journal of The Society for Industrial and Applied Mathematics* 8 (1960), pp. 300–304.

[RVW13] G. N. Rothblum, S. P. Vadhan, and A. Wigderson. "Interactive proofs of proximity: delegating computation in sublinear time". In: *Proceedings of the 45th ACM Symposium on the Theory of Computing*. STOC '13. 2013, pp. 793–802.

[Sta21] StarkWare. "ethSTARK Documentation". Cryptology ePrint Archive, Report 2021/582. 2021.

[Val08] P. Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *Proceedings of the 5th Theory of Cryptography Conference*. TCC '08. 2008, pp. 1–18.