Accumulation Without Homomorphism

Benedikt Bünz
New York University, NY, USA

Pratyush Mishra
University of Pennsylvania, Philadelphia, PA, USA

Wilson Nguyen
Stanford University, CA, USA

William Wang
New York University, NY, USA

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.

2012 ACM Subject Classification Theory of computation → Cryptographic protocols

Keywords and phrases Proof-carrying data, incrementally verifiable computation, accumulation schemes

Digital Object Identifier 10.4230/LIPIcs.ITCS.2025.23

Related Version *Full Version*: `https://eprint.iacr.org/2024/474`

Funding *Benedikt Bünz*: Supported by Alpen Labs, Chaincode Labs, and the Sui Foundation. *Wilson Nguyen*: 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.

1 Introduction

Proof-carrying data (PCD) [27] 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) [52].[1]

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


© Benedikt Bünz, Pratyush Mishra, Wilson Nguyen, and William Wang;
licensed under Creative Commons License CC-BY 4.0

PCD has found numerous applications in both theory and practice, including enforcing language semantics [29], complexity-preserving succinct arguments [10, 9], verifiable MapReduce computations [28], image provenance [42], and consensus protocols and blockchains [44, 38, 14, 24, 19]. It is thus a key question to understand security assumptions that PCD constructions require, as well as the efficiency that they can attain under these assumptions.

Let us review how existing PCD constructions fare along both dimensions. For simplicity, we focus on the special case of IVC in the following discussion.

PCD from succinctly verifiable arguments

The standard construction of PCD is via recursive composition of succinct non-interactive arguments of knowledge (SNARKs) [10, 8, 9, 26]. 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 state-of-the-art works in this line follow the template of Fractal [26] by relying on SNARKs in the random oracle model.[2] This means that we can achieve PCD (heuristically) from only symmetric-key cryptography, but at the cost of relying on the existence of SNARKs, which are complex to construct and incur (asymptotically and concretely) high costs for the PCD prover.

PCD from accumulation

A recent popular approach to avoid this reliance on SNARKs is to construct PCD via *accumulation schemes* [22, 21, 41]. 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 that incur low costs for the PCD prover, and so has seen much interest in new constructions and deployments [21, 41, 20, 34, 39, 40]. Unfortunately, all known accumulation schemes rely on homomorphic vector commitments that are only known to exist under public-key assumptions. Additionally, because most 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 an unsatisfactory state of affairs: on the one hand we have PCD from ROM-based SNARKs that (heuristically) relies on the minimal symmetric-key assumptions, but incurs high PCD prover costs due to this reliance on SNARKs, while on the other hand we have PCD from accumulation that relies on public-key assumptions, but achieves comparatively lower PCD prover costs by avoiding SNARKs. This motivates the questions we tackle in this paper: can we design PCD that achieves low prover costs by avoiding SNARKs, while simultaneously minimizing assumptions?

1.1 Our contributions

We answer this question positively by constructing accumulation schemes *solely* in the random-oracle model. Our constructions satisfy a weaker notion of accumulation than that considered in prior work, but we show that this weaker notion still suffices to construct PCD (again, heuristically after instantiating the random oracle). We provide details below.

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

(1) Bounded-depth accumulation

We introduce a new notion of *bounded-depth* accumulation schemes that can only provide (knowledge) soundness guarantees for a limited number of consecutive accumulation steps. In contrast, prior accumulation schemes support an unbounded number of accumulation steps.

(2) PCD from bounded-depth accumulation

We show that this weaker notion of accumulation still suffices to construct (a variant of) proof-carrying data. In particular, we show that bounded-depth accumulation suffices to construct PCD for computation graphs of a-priori bounded depth $O(1)$, which is already sufficient for many applications, including the primary application of constructing polynomial-length IVC [10].

We note that, like our construction, all prior PCD constructions only provably support computation graphs of bounded depth [10]. However, unlike these prior works, our construction is vulnerable to attacks when the depth of the computation graph exceeds an *a priori* fixed constant. See Remarks 2 and 3 for details.

(3) Efficient constructions of bounded-depth accumulation

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.

Compared to the prior state-of-the-art accumulation schemes, our construction has several advantages beyond just avoiding public-key assumptions, such as true linear time for the accumulation prover,[3] and plausible post-quantum security.[4] (We note that both our construction and prior accumulation schemes rely on the random oracle model; it is an open question to construct an accumulation scheme in the standard model.)

(4) Efficiency of and optimizations for instantiated PCD

Instantiating our generic PCD construction with our accumulation scheme leads to PCD schemes with numerous prover efficiency benefits compared to prior work. We also provide several optimizations for this 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. We detail the impact of these optimizations in Table 1.

Our scheme also has prover efficiency benefits beyond those captured by Table 1. To elaborate on these, we briefly detail the key factors that contribute to PCD prover cost. In all practical PCD constructions, to produce a proof for the next step of the distributed computation, the PCD prover invokes an underlying cryptographic proof system to prove satisfaction of a circuit C_{PCD} that contains a circuit representation C_Φ of the computation step (or *PCD predicate* Φ), and also performs other checks required by the PCD construction. Thus, PCD prover efficiency is determined by the prover efficiency of this proof system, and by the cost $|C_{\mathsf{PCD}}| - |C_\Phi|$ of the additional checks. We call the former the *proof-system overhead*, and the latter the PCD *circuit overhead*.

[3] To accumulate proofs for circuits of size n, prior accumulation schemes require performing an n-sized multi-scalar multiplication over elliptic-curve groups. As explained in prior work [37], this operation requires $\omega(n)$ group operations, and is hence not truly linear time.

[4] We only claim plausible post-quantum security, as we prove our construction in the random oracle model, instead of the quantum random oracle model [12]. We believe that our results can be extended to the latter model by applying techniques from prior work [25], but leave this to future work.

Table 1 Comparison of IVC schemes constructed from PCD over a tree of depth d and arity m. All costs omit constant factors. The circuit overhead measures $|C_{\mathsf{PCD}}| - |C_{\Phi}|$. The table displays the cost per invocation of Φ. Above n is the size of the recursive circuit, $n^{\star} = O(d^{\star}mn)$ is the circuit size of the accumulation decider for the recursive circuit, $|C_{\mathsf{MT}}|$ is the circuit size of checking a membership proof in a Merkle Tree with n leaves. T_{MT} is the time it takes to verify a Merkle Tree opening. Note that Fractal supports IVC computations of unbounded length, whereas all but the hybrid construction presented in this paper have a length bound of m^d.

scheme	circuit overhead per step	IVC verifier
Fractal [26]	$\lambda \log n \cdot \lvert C_{\mathsf{MT}}\rvert$	$\lambda \log n \cdot T_{\mathsf{MT}}$
+ STIR [2] (concurrent)	$(\log n + \lambda \log\log n) \cdot \lvert C_{\mathsf{MT}}\rvert$	$(\log n + \lambda \log\log n) \cdot T_{\mathsf{MT}}$
this paper	$d \cdot \lambda \cdot \lvert C_{\mathsf{MT}}\rvert$	$d \cdot n$
+ batch comm. (Sec 2.5)	$\frac{d\cdot\lambda}{m} \cdot \lvert C_{\mathsf{MT}}\rvert$	$d \cdot m \cdot n$
+ low-overhead IVC (Sec 2.6)	$\frac{d\cdot\lambda}{m^2} \cdot \lvert C_{\mathsf{MT}}\rvert$	$d \cdot m \cdot n$
+ hybrid (Sec 2.7)	$(\frac{d^{\star}\cdot\lambda}{m^2} + \frac{\lambda \log n^{\star}}{m^{d^{\star}}}) \cdot \lvert C_{\mathsf{MT}}\rvert$	$\lambda \log n^{\star} \cdot T_{\mathsf{MT}}$

PCD constructions based on ROM-based SNARKs have been able to achieve best-in-class proof-system overhead via extensive asymptotic and concrete optimizations (e.g., by achieving truly linear-time provers [15, 16, 17, 37], or relying on small fields [31]), but suffer from high PCD circuit overhead as the SNARK verifier subcircuit must perform numerous expensive random oracle calls.

Accumulation-based PCD constructions, on the other hand, have the smallest PCD circuit overhead (e.g., just ten thousand gates for Nova [41] vs. over one million gates for Fractal [26]). They also try to lower proof-system overhead by avoiding some cryptographic work (e.g., by requiring commitments only to the witness [20]), and via efficient arithmetizations of predicates (e.g., via cheap custom gates[5]). However, the remaining cryptographic work is quite expensive as it requires public-key operations, and hence the proof-system overhead of these constructions is relatively high.

Our construction of bounded-depth accumulation achieves the best of both worlds: it enjoys the low PCD circuit overhead of accumulation-based PCD constructions while taking advantage of the proof-system optimizations enjoyed by both approaches. In fact, the combination opens up new efficiency improvements that are not possible in either of the two paradigms alone. For example, all known ROM-SNARK-based PCD constructions thus far have relied on Reed–Solomon codes, and incur quasilinear prover costs from the quasilinear encoding time of these codes. Our construction, on the other hand, allows the use of any linear code, including linear-time-encodable codes [50, 33, 37], thus reducing prover costs.

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 [26]. 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.[6]

[5] Custom gates enforce checks beyond simple addition and multiplication. Examples include high-degree polynomial gates [35] and lookup gates [15, 36]. See [23, 30, 3] for empirical evidence of the benefits of such gates.

[6] Like us, their PCD construction must instantiate the random oracle with a concrete hash function, and can hence only achieve heuristic security.

Boneh, Drake, Fisch, and Gabizon [13] 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) [7, 48]. For example, like recent work [51, 46, 31], 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 [9, 18, 22, 21, 13, 41, 39, 20, 34, 40] 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 [43]. 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 [11] 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, 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.

Another concurrent work, STIR [2], provides a new proximity test that can be used as a drop-in replacement for the proximity test [5] used in Fractal [26]. As noted in Table 1, our schemes still have lower recursion overhead than this optimized baseline. Furthermore, this new STIR + Fractal-based PCD should also be compatible with our hybrid construction from Section 2.7.

▶ Remark 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 [10]. See Remarks 2 and 3 for a more detailed discussion.

2 Techniques

We begin by reviewing the definition of an accumulation scheme [22, 21].[7] 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 3 for a comprehensive definition.

Prior constructions

All prior accumulation schemes [22, 21, 41, 20, 34, 39, 40] 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,

[7] We restrict our presentation to *split* accumulation schemes, as defined in [21].

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 [45], 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 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 $\mathsf{acc}_3.\mathbb{x}$ [41, 20]. 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" [13].

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 [47].

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 2.1.1, then our previous analysis of the spot check implies 2.1.1. 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 [52].[8]

Returning to accumulation, suppose that a (possibly malicious) prover outputs $\mathsf{acc}_1.\mathbb{x}$, $\mathsf{acc}_2.\mathbb{x}$, and 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.

[8] To be precise, we must also consider a malicious prover that does not commit to a full vector; see the full version of the paper.

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 [6], 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 [21].

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 [21, 13, 41] do not immediately translate to the bounded-depth setting. To see why, let us recall a simplified version of the construction from [21]. 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 $(\pi_{i+1}, \mathsf{acc}_{i+1})$.

$R(\mathbb{x} = (z_{i+1}, \mathsf{acc}_{i+1}.\mathbb{x}), \mathbb{w} = (z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}, \mathsf{pf}_{i+1})) :$
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.

- 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.

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. [21] 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 the full version of the paper.

▶ Remark 2 (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 [10]. 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 3 (bounded-depth vs. constant-depth PCD). Perhaps surprisingly, even with standard (unbounded) accumulation, [21] is only able to construct *constant-depth PCD*.[9] 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 is *necessary* for real-world implementations, unlike in prior work.

[9] 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 [20, 34]. They support a general class of non-interactive arguments where consists of the NP witness along with a short vector commitment to the witness, and whose verifier performs three steps: (a) computing Fiat–Shamir challenges; (b) checking vector commitment openings; and (c) evaluating a polynomial over the instance, challenges, and openings. The key insight of ProtoStar and ProtoGalaxy 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 [34], focusing on the case where we are trying to accumulate claims about satisfaction of R1CS, a popular NP language in the succinct argument literature. Recall that an R1CS instance consists of constraint matrices $A, B, C \in \mathbb{F}^{N \times (\ell+n)}$, and is said to be satisfied by a public input $x \in \mathbb{F}^\ell$ and witness $w \in \mathbb{F}^n$ if $A \cdot (x||w) \circ B \cdot (x||w) - C \cdot (x||w) = 0^N$. In ProtoGalaxy, a NARK proof for an R1CS instance with public input x would consist of the proof witness, which is just the R1CS witness w, and the proof instance, which is just a vector commitment to w. The polynomial p evaluated by the verifier is of the form $p(x, w) = A \cdot (x||w) \circ B \cdot (x||w) - C \cdot (x||w)$. Then, to accumulate R1CS claims, ProtoGalaxy's accumulator witness $\mathsf{acc.w}$ is *also* a vector $w \in \mathbb{F}^n$, and its accumulator instance $\mathsf{acc.x}$ consists of a vector $\vec{x} \in \mathbb{F}^\ell$, a (homomorphic) vector commitment to w, and an error term $e \in \mathbb{F}$. Thus, a NARK proof verification claim can be seen as a special case of the accumulator verification case where the error term is zero.

To accumulate m such (accumulation or NARK) verification claims for accumulators $\mathsf{acc}_1, \ldots, \mathsf{acc}_m$, ProtoGalaxy relies on the following univariate polynomial identity:

$$p\left(\sum_{i=1}^{m} L_i(X) \cdot x_i, \sum_{i=1}^{m} L_i(X) \cdot w_i\right) = \sum_{i=1}^{m} L_i(X) \cdot e_i \mod v_H(X),$$

where L_i is the i-th Lagrange polynomial of some m-sized set $H \subset \mathbb{F}$, and $v_H(X)$ is the vanishing polynomial on H. For some quotient polynomial q, this claim can be rewritten as

$$p\left(\sum_{i=1}^{m} L_i(X) \cdot x_i, \sum_{i=1}^{m} L_i(X) \cdot w_i\right) = \sum_{i=1}^{m} L_i(X) \cdot e_i + q(X) \cdot v_H(X),$$

and this formulation allows the verifier to check the identity at a random point $\alpha \in \mathbb{F}$ if the prover sends q.

This allows us to accumulate the m input claims as follows. The prover constructs a new accumulator acc whose new instance is $x := \sum_i L_i(\alpha) \cdot x_i$, new witness 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 instance and new error term. Finally, the accumulation decider completes the checks by asserting that that $p(x, w) = e$.[10]

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

[10] Technically speaking, this check, as stated, is ill-formed, as the output of p is an N-sized vector, while e is a single field element. However, this can be fixed by taking a random linear combinations of the output vector. ProtoStar and ProtoGalaxy show how to represent the latter task *also* as a low-degree polynomial evaluation.

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 $p(x, w) = e$. 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 [20] and ProtoGalaxy [34], including support for arbitrary arity $m = \text{poly}(\lambda)$, which is crucial for constructing PCD (see Remark 2), and efficient support for custom gates.

Extension to arbitrary linear codes

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, a modified version of our construction extends to arbitrary linear codes. This is advantageous as these codes can have asymptotically faster encoding time compared to Reed–Solomon codes [37]. The construction can work with any proximity gap (e.g., uniformly random coefficients). The key idea is to commit to two codes within the accumulator, one for proximity checking and the other for the algebraic check using $p(X)$. See Section 4.3 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 [26].

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 and 3 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 [10] works by constructing a *PCD tree* whose leaves comprise the actual computation being performed. To achieve constant computation depth, Bitansky et al. [10] 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 how to reduce 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. We provide more details in the full version of the paper.

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. [52, 10]. 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). We provide more details in the full version of this paper.

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 [26], 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 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 scheme, which only supports an *a priori* fixed depth. We provide more details in the full version of this paper.

3 Bounded-depth accumulation

Let $\mathsf{ARG} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ be a non-interactive argument where knowledge soundness (but not necessarily completeness) additionally holds for a "relaxed" verifier $\tilde{\mathcal{V}}$. 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 $\mathsf{pp}_{\mathsf{AS}}$.
- $\mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathbb{i}) \to (\mathsf{apk}, \mathsf{avk}, \mathsf{dk})$: 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 $\mathsf{acc} = \mathrm{P}(\mathsf{apk}, \bot)$, 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:

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

The extractor implicitly receives s as input.

▶ Remark 4. 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.

4 Constructing bounded-depth accumulation

We construct a bounded-depth accumulation scheme, which supports (possibly distinct) arities $m_1 = \text{poly}(\lambda)$ and $m_2 = \text{poly}(\lambda)$, 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. All security proofs are deferred to the full version of the paper.

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 $(\vec{\mathsf{cm}}^{(j)}, \vec{\mathsf{aux}}^{(j)}) \leftarrow \mathsf{VC.Commit}(\mathsf{vp}, f^{(j)})$."

4.1 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 $\mathcal{R}(\mathbb{F})$. 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 [34, 20].

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'.

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 [20] 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 [4]. 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 2, for the indexed relation family $\{\mathcal{R}_{\mathsf{pp}} = \mathcal{R}(\mathbb{F})\}$. We also define an indexer $\mathcal{I}$ as a helper algorithm.

Attema et al. [4] 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.

$\mathcal{G}(1^\lambda)$:

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 $\mathsf{ipk} = (\mathbb{F}, \mathsf{vp}, \mathbb{i}, \tau, \mathbb{p}, \mathbf{C})$ and $\mathsf{ivk} = (\mathbb{F}, \mathsf{vp}, \tau, \mathbb{p}, p, \mathbf{C})$.

$\mathcal{P}^\rho(\mathsf{pp}, \mathbb{i}, \mathbb{x}, \mathbb{w})$:

1. Compute the proving key $\mathsf{ipk} = (\mathbb{F}, \mathsf{vp}, \mathbb{i}, \tau, \mathbb{p}, \mathbf{C})$ 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 $\mathsf{ivk} = (\mathbb{F}, \mathsf{vp}, \tau, \mathbb{p}, p, \mathbf{C})$ 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 $p(\mathbb{x}, \mathbf{r}, \hat{\mathbf{f}}) = 0$.

$\tilde{\mathcal{V}}^\rho(\mathsf{pp}, \mathbb{i}, \mathbb{x}, \pi = (\vec{\mathsf{cm}}, \vec{\mathsf{aux}}))$:

1. Compute the verification key $\mathsf{ivk} = (\mathbb{F}, \mathsf{vp}, \tau, \mathbb{p}, p, \mathbf{C})$ 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 2 Non-interactive argument algorithms.

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 $\tilde{\mathcal{V}}$ (also shown in Figure 2). First, we allow it to decode noisy codewords. Second, we allow it to accept partial openings to the vector commitment where the missing positions are interpreted as erasure errors. Because the verifier only accepts when the partial opening is decodable, the prover is still bound to a unique (decoded) vector; hence, knowledge soundness is not affected.

4.2 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 $\mathsf{qw} = \mathsf{a}\vec{\mathsf{u}}\mathsf{x}$. 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.

$\textsc{Prove}^{\rho}(\mathsf{apk}, [e_i, x_i, \vec{\mathsf{cm}}_i, \mathsf{a}\vec{\mathsf{u}}\mathsf{x}_i]_{i=1}^{m})$:
1. For each $i \in [m]$, let $\mathbf{f}_i \leftarrow \mathsf{VC.Answer}^{\rho_H}(\mathsf{vp}, \vec{\mathsf{cm}}_i, \mathsf{a}\vec{\mathsf{u}}\mathsf{x}_i)$.
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 $(\vec{\mathsf{cm}}, \mathsf{a}\vec{\mathsf{u}}\mathsf{x}) \leftarrow \mathsf{VC.Commit}^{\rho_H}(\mathsf{vp}, \mathbf{f})$.
7. Set $\mathsf{acc}.\mathbb{x} = (e, x, \vec{\mathsf{cm}})$ and $\mathsf{acc}.\mathbb{w} = \mathsf{a}\vec{\mathsf{u}}\mathsf{x}$.
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 $\vec{\mathsf{op}}_i \leftarrow \mathsf{VC.Open}^{\rho_H}(\mathsf{vp}, \mathsf{a}\vec{\mathsf{u}}\mathsf{x}_i, \mathbf{Q})$.
10. Let $\vec{\mathsf{op}} \leftarrow \mathsf{VC.Open}^{\rho_H}(\mathsf{vp}, \mathsf{a}\vec{\mathsf{u}}\mathsf{x}, \mathbf{Q})$.
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 $e = v_H(\alpha) \cdot q(\alpha) + \sum_{i=1}^m L_{i,H}(\alpha) \cdot e_i$.
6. Verify that $(x, \mathbf{v}) = \sum_{i=1}^m L_{i,H}(\alpha) \cdot (x_i, \mathbf{v}_i)$.

See Figure 3 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 3 Accumulation scheme algorithms.

4.3 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 [49, 1, 32] 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 5** (Proximity gaps for linear codes)**.** *Let C be a linear code with relative distance d and blocklength n. We will treat elements of C as functions $v : [n] \to \mathbb{F}$. Let* Gen *be a function that takes in randomness* rnd *and outputs coefficients $r_1, \ldots, r_m \in \mathbb{F}$. We say that C has a proximity gap with respect to distribution* Gen*, error* err*, and proximity bound* B *if the following holds.*

For any $\delta \leq \mathsf{B}$ and arbitrary functions $u_1, \ldots, u_m \in \mathbb{F}^n : [n] \to \mathbb{F}$, if

$$\Pr_{\mathsf{rnd}}\left[\Delta\left(\sum_{i=1}^m r_i \cdot u_i, C\right) \leq \delta \;:\; (r_1, \ldots, r_m) \leftarrow \mathsf{Gen}(\mathsf{rnd})\right] > \mathsf{err}(m),$$

then for all $(r_1, \ldots, r_m)$ in the support of the distribution $\mathsf{Gen}(\mathsf{rnd})$*, we have*

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

Furthermore, there exists $v_1, \ldots, v_m \in C$ such that for all $(r_1, \ldots, r_m)$ in the support,

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

and in fact $|\{x \in [n] : (u_1(x), \ldots, u_m(x)) \neq (v_1(x), \ldots, v_m(x))\}| \leq \delta n$.

▶ **Lemma 6.** *Let C be a linear code with relative distance d, blocklength n, and proximity gap with respect to distribution* Gen*, error* err*, and proximity bound* B*. For $\delta \leq \mathsf{B}$ and vectors $f_1, \ldots, f_m \in \mathbb{F}^n$, suppose the following holds:*

$$\Pr_{\mathsf{rnd}}\left[\Delta\left(\sum_{i=1}^m r_i \cdot f_i, C\right) \leq \delta \;:\; (r_1, \ldots, r_m) \leftarrow \mathsf{Gen}(\mathsf{rnd})\right] > \mathsf{err}(m)$$

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

Construction

We use linear codes with efficient unique decoding up to $\mathsf{B} = O(1)$, the error bound of the proximity gap. Fix $\delta \leq \mathsf{B}/d_s$; this guarantees that $d_s\delta \leq \mathsf{B}$ which is smaller than the unique decoding radius. The modified construction is shown below; all changes are highlighted in blue. We omit the description of unchanged algorithms from Figure 3.

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

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.

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

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 $\alpha, \mathsf{rnd} \leftarrow \rho_{\mathsf{AS}}(\tau, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, q)$, where $\alpha \in \mathbb{F}$ and rnd 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 $(r_1, \ldots, r_m, r'_1, \ldots, r'_m) \leftarrow \mathsf{Gen}(\mathsf{rnd})$. 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 $\mathsf{pf} = (q, [\vec{\mathsf{op}}_i, \vec{\mathsf{op}}'_i]_{i=1}^m, \vec{\mathsf{op}}, \vec{\mathsf{op}}')$.

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

1. Query $\alpha, \mathsf{rnd} \leftarrow \rho_{\mathsf{AS}}(\tau, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, q)$.
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 $(r_1, \ldots, r_m, r'_1, \ldots, r'_m) \leftarrow \mathsf{Gen}(\mathsf{rnd})$. Verify that $\mathbf{v}' = \sum_{i=1}^m r_i \cdot v_i + r'_i \cdot v'_i$.

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

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

References

1 Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 24th ACM Conference on Computer and Communications Security*, CCS '17, pages 2087–2104, 2017. URL: `https://eprint.iacr.org/2022/1608`, `doi:10.1145/3133956.3134104`.

2 Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. STIR: reed-solomon proximity testing with fewer queries. In *Proceedings of the 44th Annual International Cryptology Conference*, CRYPTO '24, pages 380–413, 2024. URL: `https://eprint.iacr.org/2024/390`, `doi:10.1007/978-3-031-68403-6_12`.

3 Arasu Arun, Srinath T. V. Setty, and Justin Thaler. Jolt: Snarks for virtual machines via lookups. In *Proceedings of the 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '24, pages 3–33, 2024. URL: https://eprint.iacr.org/2023/1217, doi:10.1007/978-3-031-58751-1_1.

4 Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs (extended version). *J. Cryptol.*, 36(4):36, 2023. URL: https://eprint.iacr.org/2021/1377, doi:10.1007/S00145-023-09478-Y.

5 Eli Ben-Sasson, Iddo Bentov, Ynon Horesh, and Michael Riabzev. Fast Reed–Solomon interactive oracle proofs of proximity. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming*, ICALP '18, pages 14:1–14:17, 2018. doi:10.4230/LIPICS.ICALP.2018.14.

6 Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for reed-solomon codes. *JACM*, 70(5):31:1–31:57, 2023. URL: https://eprint.iacr.org/2020/654, doi:10.1145/3614423.

7 Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Proceedings of the 14th Theory of Cryptography Conference*, TCC '16-B, pages 31–60, 2016. URL: https://eprint.iacr.org/2016/116, doi:10.1007/978-3-662-53644-5_2.

8 Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In *Proceedings of the 34th Annual International Cryptology Conference*, CRYPTO '14, pages 276–294, 2014. URL: https://eprint.iacr.org/2014/595, doi:10.1007/978-3-662-44381-1_16.

9 Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. *Algorithmica*, 79(4):1102–1160, 2017. URL: https://eprint.iacr.org/2014/595, doi:10.1007/S00453-016-0221-0.

10 Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran 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, pages 111–120, 2013. URL: http://eprint.iacr.org/2012/095, doi:10.1145/2488608.2488623.

11 Dan Boneh and Binyi Chen. Latticefold: A lattice-based folding scheme and its applications to succinct proof systems. ePrint Report 2024/257, 2024. URL: https://eprint.iacr.org/2024/257.

12 Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark 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, pages 41–69, 2011. URL: https://eprint.iacr.org/2010/428, doi:10.1007/978-3-642-25385-0_3.

13 Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *Proceedings of the 41st Annual International Cryptology Conference*, CRYPTO '21, 2021. URL: https://eprint.iacr.org/2020/1536.

14 Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. ePrint Report 2020/352, 2020. URL: https://eprint.iacr.org/2020/352.

15 Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In *Proceedings of the 24th International Conference on the Theory and Application of Cryptology and Information Security*, ASIACRYPT '18, pages 595–626, 2018. URL: https://eprint.iacr.org/2018/380, doi:10.1007/978-3-030-03326-2_20.

16 Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sublinear verification from tensor codes. In *Proceedings of the 18th International Conference on the Theory of Cryptography*, TCC '20, pages 19–46, 2020. URL: https://eprint.iacr.org/2020/1426, doi:10.1007/978-3-030-64378-2_2.

17 Jonathan Bootle, Alessandro Chiesa, and Siqi Liu. Zero-knowledge iops with linear-time prover and polylogarithmic-time verifier. In *Proceedings of the 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '22, pages 275–304, 2022. URL: https://eprint.iacr.org/2020/1527, doi:10.1007/978-3-031-07085-3_10.

18 Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. ePrint Report 2019/1021, 2019. URL: https://eprint.iacr.org/2019/1021.

19 Elette Boyle, Ran Cohen, and Aarushi Goel. Breaking the $o(\sqrt{n})$-bit barrier: Byzantine agreement with polylog bits per party. *J. Cryptol.*, 37(1):2, 2024. arXiv:2002.02516.

20 Benedikt Bünz and Binyi 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, pages 77–110, 2023. URL: https://eprint.iacr.org/2023/620, doi:10.1007/978-981-99-8724-5_3.

21 Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In *Proceedings of the 41st Annual International Cryptology Conference*, CRYPTO '21, pages 681–710, 2021. URL: https://eprint.iacr.org/2020/1618, doi:10.1007/978-3-030-84242-0_24.

22 Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. In *Proceedings of the 18th Theory of Cryptography Conference*, TCC '20, 2020. URL: https://eprint.iacr.org/2020/499.

23 Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *Proceedings of the 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '23, pages 499–530, 2023. URL: https://eprint.iacr.org/2022/1355, doi:10.1007/978-3-031-30617-4_17.

24 Weikeng Chen, Alessandro Chiesa, Emma Dauterman, and Nicholas P. Ward. Reducing participation costs via incremental verification for ledger systems. ePrint Report 2020/1522, 2020. URL: https://eprint.iacr.org/2020/1522.

25 Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In *Proceedings of the 17th International Conference on the Theory of Cryptography*, TCC '19, pages 1–29, 2019. URL: https://eprint.iacr.org/2019/834, doi:10.1007/978-3-030-36033-7_1.

26 Alessandro Chiesa, Dev Ojha, and Nicholas 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. URL: https://eprint.iacr.org/2019/1076.

27 Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Proceedings of the 1st Symposium on Innovations in Computer Science*, ICS '10, pages 310–331, 2010. URL: https://people.eecs.berkeley.edu/~alexch/docs/CT10.pdf.

28 Alessandro Chiesa, Eran Tromer, and Madars Virza. Cluster computing in zero knowledge. In *Proceedings of the 34th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '15, pages 371–403, 2015. URL: https://eprint.iacr.org/2015/377, doi:10.1007/978-3-662-46803-6_13.

29 Stephen Chong, Eran Tromer, and Jeffrey A. Vaughan. Enforcing language semantics using proof-carrying data. ePrint Report 2013/513, 2013. URL: http://eprint.iacr.org/2013/513.

30 Michel Dellepere, Pratyush Mishra, and Alireza Shirzad. Garuda and pari: Faster and smaller snarks via equifficient polynomial commitments. ePrint Report 2024/1245, 2024. URL: https://eprint.iacr.org/2024/1245.

31 Benjamin E. Diamond and Jim Posen. Succinct arguments over towers of binary fields. ePrint Report 2023/1784, 2023. URL: https://eprint.iacr.org/2023/1784.

32 Benjamin E. Diamond and Jim Posen. Proximity testing with logarithmic randomness. *IACR Commun. Cryptol.*, 1(1):2, 2024. URL: https://eprint.iacr.org/2023/630, doi:10.62056/AKSDKP10.

33 Erez Druk and Yuval Ishai. Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In *Proceedings of the 5th Innovations in Theoretical Computer Science Conference*, ITCS '14, pages 169–182, 2014. URL: https://dl.acm.org/doi/pdf/10.1145/2554797.2554815, doi:10.1145/2554797.2554815.

34 Liam Eagen and Ariel Gabizon. Protogalaxy: Efficient protostar-style folding of multiple instances. ePrint Report 2023/1106, 2023. URL: https://eprint.iacr.org/2023/1106.

35 Ariel Gabizon and Zachary J. Williamson. The turbo-plonk program syntax for specifying snark programs, 2019. URL: https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo_plonk.pdf.

36 Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. ePrint Report 2020/315, 2020. URL: https://eprint.iacr.org/2020/315.

37 Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic snarks for R1CS. In *Proceedings of the 43rd Annual International Cryptology Conference*, CRYPTO '23, pages 193–226, 2023. URL: https://eprint.iacr.org/2021/1043, doi:10.1007/978-3-031-38545-2_7.

38 Assimakis Kattis and Joseph Bonneau. Proof of necessary work: Succinct state verification with fairness guarantees. ePrint Report 2020/190, 2020. URL: https://eprint.iacr.org/2020/190.

39 Abhiram Kothapalli and Srinath Setty. Cyclefold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. ePrint Report 2023/1192, August 2023. URL: https://eprint.iacr.org/2023/1192.

40 Abhiram Kothapalli and Srinath T. V. Setty. Hypernova: Recursive arguments for customizable constraint systems. In *Proceedings of the 44th Annual International Cryptology Conference*, CRYPTO '24, pages 345–379, 2024. URL: https://eprint.iacr.org/2023/573, doi:10.1007/978-3-031-68403-6_11.

41 Abhiram Kothapalli, Srinath T. V. Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Proceedings of the 42nd Annual International Cryptology Conference*, CRYPTO '22, pages 359–388, 2022. URL: https://eprint.iacr.org/2021/370, doi:10.1007/978-3-031-15985-5_13.

42 Assa Naveh and Eran 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, pages 255–271, 2016. URL: https://ieeexplore.ieee.org/document/7546506, doi:10.1109/SP.2016.23.

43 Wilson D. Nguyen, Dan Boneh, and Srinath 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, pages 18:1–18:22, 2023. URL: https://eprint.iacr.org/2023/969, doi:10.4230/LIPICS.AFT.2023.18.

44 O(1) Labs. Mina Cryptocurrency, 2020. minaprotocol.org.

45 Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference*, CRYPTO '91, pages 129–140, 1992. URL: https://link.springer.com/content/pdf/10.1007/3-540-46766-1_9.pdf.

46 Polygon Zero Team. Plonky2: Fast recursive arguments with plonk and fri. URL: https://github.com/0xPolygonZero/plonky2/blob/main/plonky2/plonky2.pdf.

47 Irving S. Reed, Gustave Solomon, and Kim Hamilton March. Polynomial codes over certain finite fields. *Journal of The Society for Industrial and Applied Mathematics*, 8:300–304, 1960.

48 Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. *SIAM J. Comp.*, 50(3), 2021. doi:10.1137/16M1096773.

49 Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In *Proceedings of the 45th ACM Symposium on the Theory of Computing*, STOC '13, pages 793–802, 2013. URL: https://privacytools.seas.harvard.edu/files/privacytools/files/stoc283fp-rothblum.pdf. doi:10.1145/2488608.2488709.

50 Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. on Inf. Theory*, 42(6):1723–1731, 1996. URL: http://cs.yale.edu/homes/spielman/PAPERS/linearTimeIT.pdf, doi:10.1109/18.556668.

51 StarkWare. ethstark documentation. ePrint Report 2021/582, 2021. URL: https://eprint.iacr.org/2021/582.

52 Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Proceedings of the 5th Theory of Cryptography Conference*, TCC '08, pages 1–18, 2008. URL: https://iacr.org/archive/tcc2008/49480001/49480001.pdf, doi:10.1007/978-3-540-78524-8_1.