Proof-Carrying Data without Succinct Arguments

Benedikt Bünz
benedikt@cs.stanford.edu
Stanford University

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

William Lin
will.lin@berkeley.edu
UC Berkeley

Pratyush Mishra
pratyush@berkeley.edu
UC Berkeley

Nicholas Spooner
nspooner@bu.edu
Boston University

December 31, 2020

Abstract

Proof-carrying data (PCD) is a powerful cryptographic primitive that enables mutually distrustful parties to perform distributed computations that run indefinitely. Known approaches to construct PCD are based on succinct non-interactive arguments of knowledge (SNARKs) that have a succinct verifier or a succinct accumulation scheme for their proofs.

In this paper we show how to obtain PCD without relying on SNARKs. We construct a PCD scheme given any non-interactive argument of knowledge (e.g., with linear-size proofs) that has a *split accumulation scheme*, which is a weak form of accumulation that we introduce.

We additionally construct a transparent non-interactive argument of knowledge for R1CS whose accumulation is verifiable via a *constant number of group and field operations*. This leads, via the random oracle heuristic and our result above, to efficiency improvements for PCD. Along the way, we construct a split accumulation scheme for a simple polynomial commitment scheme based on Pedersen commitments.

Our results are supported by a modular and efficient implementation.

Keywords: proof-carrying data; accumulation schemes; recursive proof composition

Contents

1 Introduction

Proof-carrying data (PCD) [CT10] is a powerful cryptographic primitive that enables mutually distrustful parties to perform distributed computations that run indefinitely, while ensuring that every intermediate state of the computation can be efficiently verified. A special case of PCD is *incrementally-verifiable computation* (IVC) [Val08]. PCD has found applications in enforcing language semantics [CTV13], verifiable MapReduce computations [CTV15], image authentication [NT16], blockchains [Mina; KB20; BMRS20; CCDW20], and others. Given the theoretical and practical relevance of PCD, it is an important research question to build efficient PCD schemes from minimal cryptographic assumptions.

PCD from succinct verification. The canonical construction of PCD is via *recursive composition* of succinct non-interactive arguments (SNARGs) [BCCT13; BCTV14; COS20]. Informally, a proof that the computation was executed correctly for t steps consists of a proof of the claim "the t-th step of the computation was executed correctly, and there exists a proof that the computation was executed correctly for $t-1$ steps". The latter part of the claim is expressed using the SNARG verifier itself. This construction yields secure PCD (with IVC as a special case) provided the SNARG satisfies an adaptive knowledge soundness property (i.e., is a SNARK). Efficiency requires the SNARK to have sublinear-time verification, achievable via SNARKs for machine computations [BCCT13] or preprocessing SNARKs for circuit computations [BCTV14; COS20].

Requiring sublinear-time verification, however, significantly restricts the choice of SNARK, which limits what we can achieve for PCD. These restrictions have practical implications: the concrete efficiency of recursion is limited by the use of expensive curves for pairing-based SNARKs [BCTV14] or heavy use of cryptographic hash functions for hash-based SNARKs [COS20].

PCD from accumulation. Recently, [BCMS20] gave an alternative construction of PCD using SNARKs that have succinct *accumulation schemes*; this developed and formalized a novel approach for recursion sketched in [BGH19]. Informally, rather than being required to have sublinear-time verification, the SNARK is required to be accompanied by a cryptographic primitive that enables "postponing" the verification of SNARK proofs by way of an accumulator that is updated at each recursion step. The main efficiency requirement on the accumulation scheme is that the accumulation procedure must be succinctly verifiable, and in particular the accumulator itself must be succinct.

Requiring a SNARK to have a succinct accumulation scheme is a weaker condition than requiring it to have sublinear-time verification. This has enabled constructing PCD from SNARKs that do *not* have sublinear-time verification [BCMS20]. This yields PCD constructions from assumptions and with efficiency properties that were not previously achieved. Practitioners have exploited this freedom to design implementations of recursive composition with improved practical efficiency [Halo20; Pickles20].

Our motivation. The motivation of this paper is twofold. First, can PCD be built from a weaker primitive than SNARKs with succinct accumulation schemes? If so, can we leverage this to obtain PCD constructions with improved *concrete* efficiency?

1.1 Contributions

We make theory and systems contributions that advance the state of the art for PCD: (1) We introduce *split accumulation schemes for relations*, a cryptographic primitive that relaxes prior notions of accumulation. (2) We obtain PCD from any non-interactive argument of knowledge that satisfies this weaker notion of accumulation; surprisingly, this allows for arguments with no succinctness whatsoever. (3) We construct a non-interactive argument of knowledge based on discrete logarithms (and random oracles) whose accumulation verifier uses a constant number of group operations (improving over logarithmically many of prior

accumulation schemes in this setting). (4) We contribute libraries with generic implementation of PCD via accumulation from [BCMS20] and this paper and corresponding accumulation schemes.

We elaborate on each of these contributions next.

(1) Split accumulation for relations. Recall from [BCMS20] that an accumulation scheme for a predicate $\Phi\colon X \to \{0,1\}$ enables proving/verifying that each input in an infinite stream $\mathsf{q}_1, \mathsf{q}_2, \ldots$ satisfies the predicate Φ, by augmenting the stream with *accumulators*. Informally, for each i, the prover produces a new accumulator acc_i from the input q_i and the old accumulator acc_{i-1}; the verifier can check that the triple $(\mathsf{acc}_{i-1}, \mathsf{q}_i, \mathsf{acc}_i)$ is a valid accumulation step, much more efficiently than running Φ on q_i. At any time, the decider can validate acc_i, which establishes that for all $j \leq i$ it was the case that $\Phi(\mathsf{q}_j) = 1$. The accumulator size (and hence the running time of the three algorithms) cannot grow in the number of accumulation steps.

We extend this notion in two orthogonal ways. First we consider relations $\Phi\colon X \times W \to \{0,1\}$ and now for a stream of instances $\mathsf{q}_1.\mathbb{x}, \mathsf{q}_2.\mathbb{x}, \ldots$ the goal is to establish that there exist witnesses $\mathsf{q}_1.\mathbb{w}, \mathsf{q}_2.\mathbb{w}, \ldots$ such that $\Phi(\mathsf{q}_i.\mathbb{x}, \mathsf{q}_i.\mathbb{w}) = 1$ for each i. Second we consider accumulators acc_i that are split into an instance part $\mathsf{acc}_i.\mathbb{x}$ and a witness part $\mathsf{acc}_i.\mathbb{w}$ with the restriction that the accumulation verifier only gets to see the instance part (and possibly an auxiliary accumulation proof). We refer to this notion as *split accumulation for relations*, and refer (for contrast) the notion from [BCMS20] as *atomic accumulation for languages*.

The purpose of these extensions is to enable us to consider accumulation schemes in which predicate witnesses and accumulator witnesses are large while still requiring the accumulation verifier to be succinct (it receives short predicate instances and accumulator instances but not large witnesses). We will see that such accumulation schemes are both simpler and cheaper, while still being useful for primitives such as PCD.

(2) PCD via split accumulation. A non-interactive argument has a split accumulation scheme if the relation corresponding to its verifier has a split accumulation scheme (we make this precise later). We show that any non-interactive argument of knowledge (NARK) having a split accumulation scheme whose *accumulation verifier* is sublinear can be used to build a proof-carrying data (PCD) scheme, *even if the NARK does not have sublinear argument size*. This significantly broadens the class of non-interactive arguments from which PCD can be built, and is the first result to obtain PCD from non-interactive arguments that need not be succinct.

Theorem 1 (informal)**.** *There is an efficient transformation that compiles any NARK with a split accumulation scheme into a PCD scheme. If the NARK and its split accumulation scheme are zero knowledge, then the PCD scheme is also zero knowledge.*

Similarly to all PCD results known to date, the above theorem holds in a model where all parties have access to a common reference string, *but no oracles*. (The construction makes non-black-box use of the accumulation scheme verifier, and the theorem does not carry over to the random oracle model.)

A corollary of Theorem 1 is that any NARK with a split accumulation scheme can be "bootstrapped" into a SNARK for machine computations. (PCD implies IVC and, further assuming collision-resistant hashing, also efficient SNARKs for machine computations [BCCT13].) This is surprising: an argument with decidedly weak efficiency properties implies an argument with succinct proofs and succinct verification!

See Section 2.2 for a summary of the ideas behind Theorem 1, and Section 5 for technical details.

(3) NARK with split accumulation based on DL. Our Theorem 1 motivates the question of whether we can leverage the weaker condition on the argument system to improve the efficiency of PCD. Our focus is on minimizing the cost of the accumulation verifier for the argument system, because it is the one component that is used in a black-box way, and thus typically determines concrete efficiency. Towards this end, we present a NARK with split accumulation based on discrete logarithms, with a *constant-size* accumulation verifier; the NARK has a transparent (public-coin) setup.

Theorem 2 (informal)**.** *In the random oracle model and assuming the hardness of the discrete logarithm problem, there exists a (transparent) NARK for R1CS and a corresponding split accumulation scheme with the following efficiency:*

NARK			***split accumulation scheme***			
prover time	*verifier time*	*argument size*	*prover time*	*verifier time*	*decider time*	*accumulator size*
$O(m)\ \mathbb{G}$	$O(m)\ \mathbb{G}$	$O(1)\ \mathbb{G}$	$O(m)\ \mathbb{G}$	$O(1)\ \mathbb{G}$	$O(m)\ \mathbb{G}$	$\lvert\mathsf{acc.x}\rvert = O(1)\ \mathbb{G} + O(1)\ \mathbb{F}$
$O(m \log m)\ \mathbb{F}$	$O(m)\ \mathbb{F}$	$O(m)\ \mathbb{F}$	$O(m)\ \mathbb{F}$	$O(1)\ \mathbb{F}$	$O(m)\ \mathbb{F}$	$\lvert\mathsf{acc.w}\rvert = O(m)\ \mathbb{F}$

Above, m the number of constraints in the R1CS instance, $\mathbb{G}$ denotes group scalar multiplications or group elements, and $\mathbb{F}$ denotes field operations or field elements.

The NARK construction from Theorem 2 is particularly simple: it is obtained by applying the Fiat–Shamir transformation to a sigma protocol for R1CS based on Pedersen commitments (and linear argument size). The only "special" feature about the construction is that, as we prove, it has a very efficient split accumulation scheme for the relation corresponding to its verifier. By heuristically instantiating the random oracle, we can apply Theorem 1 (and [BCCT13]) to obtain a SNARK for machines from this modest starting point.

We find it informative to compare Theorem 2 and SNARKs with atomic accumulation based on discrete logarithms [BCMS20]:

- the SNARK's argument size is $O(\log m)$ group elements, *much less* than the NARK's $O(m)$ field elements;
- the SNARK's accumulator verifier uses $O(\log m)$ group scalar multiplications and field operations, *much more* than the NARK's $O(1)$ group scalar multiplications and field operations.

Therefore Theorem 2 offers a tradeoff that minimizes the cost of the accumulator at the expense of argument size. (As we shall see later, this tradeoff has concrete efficiency advantages.)

Our focus on argument systems based on discrete logarithms is motivated by the fact that they can be instantiated based on efficient curves suitable for recursion: the Tweedle [BGH19] or Pasta [Hop20] curve cycles, which follow the curve cycle technique for efficient recursion [BCTV14]. This focus on discrete logarithms is a choice made for this paper, and we believe that our ideas can lead to efficiency improvements to recursion in other settings (e.g., pairing-based and hash-based arguments) and leave these to future work.

(4) Implementation and evaluation. We contribute a set of Rust libraries that realize PCD via accumulation via modular combinations of interchangeable components: (a) generic interfaces for atomic and split accumulation; (b) generic construction of PCD from arguments with atomic and split accumulation; (c) accumulation schemes for polynomial commitments based on the inner product argument or a trivial use of the Pedersen commitment (this latter underlies our NARK), including constraints for the accumulation verifiers. Practitioners interested in PCD will find these libraries useful for prototyping and comparing different types of recursion (and, e.g., may help decide if current systems based on atomic recursion [Halo20; Pickles20] are better off via split recursion or not).

We additionally conduct experiments to evaluate our implementation, and establish that for constraint systems of interest, recursion based on our split accumulation scheme is much cheaper in practice than recursion based on prior atomic accumulation schemes.

Remark 1.1 (concurrent work)**.** A concurrent work [BDFG20] studies similar questions as this paper. Below we summarize the similarities and the differences between the two papers.

Similarities. Both papers are motivated by the goal of reducing the cost of recursive arguments. The main object of study in [BDFG20] is additive polynomial commitment schemes (PC schemes), for which [BDFG20] considers different types of *aggregation schemes*: (1) *public* aggregation in [BDFG20] is closely related to atomic accumulation specialized to PC schemes from a prior work [BCMS20]; and (2) *private*

aggregation in [BDFG20] is closely related to split accumulation specialized to PC schemes from this paper. Moreover, the private aggregation scheme for additive PC schemes in [BDFG20] is similar to our split accumulation scheme for Pedersen PC schemes (overviewed in Section 2.4 and detailed in Section 6). The protocols differ in how efficiency depends on the n claims to aggregate/accumulate: the verifier in [BDFG20] uses $n + 1$ group scalar multiplications while ours uses $2n$. (Informally, [BDFG20] first randomly combines claims and then evaluates at a random point, while we first evaluate at a random point and then randomly combine claims.)

Differences. The two papers develop distinct, and complementary, directions.

The focus of [BDFG20] is to design protocols for any additive PC scheme (and, even more generally, any PC scheme with a linear combination scheme), including the aforementioned private aggregation protocol and a compiler that endows a given PC scheme with zero knowledge.

In contrast, our focus is to formulate a definition of split accumulation for general relation predicates that (a) we demonstrate suffices to construct PCD, and (b) in the random oracle model, we can also demonstrably achieve via a split accumulation scheme for Pedersen commitments. We emphasize that our definitions are materially different from the case of atomic accumulation in [BCMS20], and necessitate careful consideration of technicalities such as the flavor of adaptive knowledge soundness, which algorithms can be allowed to query oracles, and so on. Hence, we cannot simply rely on the existing foundations for atomic accumulation of [BCMS20] in order to infer the correct definitions and security reductions for split accumulation. Overall, our theoretical work enables us to achieve the first construction of PCD without succinct arguments, and also to obtain a novel NARK for R1CS with constant-size accumulation verifier.

We stress that the treatment of accumulation at a higher level of abstraction than for PC schemes is essential to prove theorems about PCD. There are no known techniques that enable provably constructing PCD from an aggregation/accumulation scheme for a PC scheme (unlike what is stated in [BDFG20]), because using the PC scheme would involve using the random oracle in a way that cannot be recursively proved. Instead, one can prove a theorem in the standard model starting from an aggregation/accumulation scheme for a NARK, as we do. Separately, similar to the disconnect in [BCMS20], we do not have accumulation schemes for NARKs in the standard model so instead we prove an analogous result in the ROM.

Another major difference is that we additionally contribute a comprehensive and modular implementation of protocols from [BCMS20] and this paper, and conduct an evaluation for the discrete logarithm setting. This supports the asymptotic improvements with measured improvements in concrete efficiency.

2 Techniques

We summarize the main ideas behind our results. In Section 2.1 we discuss the new notion of split accumulation for relation predicates, and compared it with the notion of atomic accumulation for language predicates from [BCMS20]. In Section 2.2 we discuss the proof of Theorem 1. In Section 2.3 we discuss the proof of Theorem 2; for this we rely, among other things, on a new result about split accumulation for a Pedersen-based polynomial commitment (Theorem 3), which we discuss in Section 2.4. Finally in Section 2.5 we elaborate on our implementation and evaluation. Figure 1 illustrates the relation between our results. The rest of the paper contains technical details, and we provide pointers to relevant sections along the way.

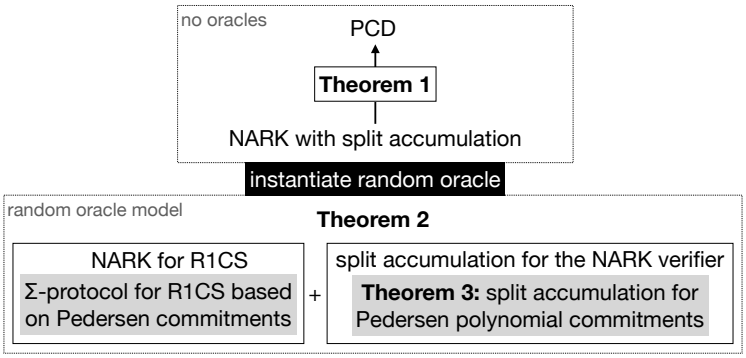

Figure 1: Diagram showing the relation between our results. Gray boxes within a result are notable subroutines.

2.1 Accumulation: atomic vs split

We review the notion of accumulation from [BCMS20], which we refer to as *atomic accumulation*, and then describe the weaker notion that we consider, which we call *split accumulation*.

Atomic accumulation for languages. An *accumulation scheme for a language predicate* $\Phi\colon X \to \{0,1\}$ consists of a tuple of algorithms $(\mathrm{P}, \mathrm{V}, \mathrm{D})$, known as the prover, verifier, and decider, that enable proving/verifying statements of the form $\Phi(\mathsf{q}_1) \wedge \Phi(\mathsf{q}_2) \wedge \cdots$ more efficiently than simply invoking the predicate Φ on each input.

This is done by starting from an initial ("empty") accumulator acc_0, the prover is used to accumulate the first input q_1 to produce a new accumulator $\mathsf{acc}_1 \leftarrow \mathrm{P}(\mathsf{acc}_0, \mathsf{q}_1)$; then the prover is used again to accumulate the second input q_2 to produce a new accumulator $\mathsf{acc}_2 \leftarrow \mathrm{P}(\mathsf{acc}_1, \mathsf{q}_2)$; and so on.

Each accumulator produced so far enables efficient verification of the predicate on all inputs that went into the accumulator. For example, to establish that $\Phi(\mathsf{q}_1) \wedge \cdots \wedge \Phi(\mathsf{q}_t) = 1$ it suffices to check that:

- the verifier accepts each accumulation step: $\mathrm{V}(\mathsf{acc}_0, \mathsf{q}_1, \mathsf{acc}_1) = 1$, $\mathrm{V}(\mathsf{acc}_1, \mathsf{q}_2, \mathsf{acc}_2) = 1$, and so on; and
- the decider accepts the final accumulator: $\mathrm{D}(\mathsf{acc}_t) = 1$.

Qualitatively, this replaces the naive cost $t \cdot |\Phi|$ with the new cost $t \cdot |\mathrm{V}| + |\mathrm{D}|$. This is beneficial when the verifier is much cheaper than checking the predicate directly and the decider is not much costlier than checking the predicate directly. Crucially, the verifier and decider costs (and, in particular, the accumulator size) should not grow with the number t of accumulation steps (which need not be known in advance).

The properties of an accumulation scheme are summarized in the following informal definition.

Definition 2.1 (informal)**.** *An* **accumulation scheme** *for a predicate* $\Phi\colon X \to \{0,1\}$ *consists of a triple of algorithms* $(\mathrm{P}, \mathrm{V}, \mathrm{D})$*, known as the prover, verifier, and decider, that satisfies the following properties.*

- Completeness: *For all accumulators* acc *and predicate inputs* $\mathsf{q} \in X$*, if* $\mathrm{D}(\mathsf{acc}) = 1$ *and* $\Phi(\mathsf{q}) = 1$*, then for* $\mathsf{acc}^\star \leftarrow \mathrm{P}(\mathsf{acc}, \mathsf{q})$ *it holds that* $\mathrm{V}(\mathsf{acc}, \mathsf{q}, \mathsf{acc}^\star) = 1$ *and* $\mathrm{D}(\mathsf{acc}^\star) = 1$.
- Soundness: *For every efficiently-generated accumulators* $\mathsf{acc}, \mathsf{acc}^\star$ *and predicate input* $\mathsf{q} \in X$*, if* $\mathrm{D}(\mathsf{acc}^\star) = 1$ *and* $\mathrm{V}(\mathsf{acc}, \mathsf{q}, \mathsf{acc}^\star) = 1$ *then, with all but negligible probability,* $\Phi(\mathsf{q}) = 1$ *and* $\mathrm{D}(\mathsf{acc}) = 1$.

The above definition omits many details, such as the ability to accumulate multiple inputs and multiple accumulators in one step, or features required for zero knowledge such as a validity proof for the accumulator (produced by P and checked by V but not passed on like the accumulator). We refer the reader to [BCMS20] for more details.

The aspect that we wish to highlight here is the following: in order for the verifier to be much cheaper than the predicate ($|\mathrm{V}| \ll |\Phi|$) it must be that the accumulator itself is much smaller than the predicate ($|\mathsf{acc}| \ll |\Phi|$) because the verifier receives the accumulator as input. (And if the accumulator is accompanied by a validity proof π_{V} then this proof must also be small.)

We refer to this setting as *atomic accumulation* because the entirety of the accumulator is treated as one short monolithic string. In contrast, next we consider a relaxed variant where this is not the case, and will enable us to consider new instantiations that ultimately lead to interesting theoretical and practical results.

Split accumulation for relations. We propose a relaxed notion of accumulation scheme. A *split accumulation scheme for a relation predicate* $\Phi\colon X \times W \to \{0,1\}$ is again a tuple of algorithms $(\mathrm{P}, \mathrm{V}, \mathrm{D})$ as before. The difference is that: (a) an input q to Φ is split into a short instance part $\mathsf{q}.\mathbb{x}$ and a (possibly) long witness part $\mathsf{q}.\mathbb{w}$; (b) an accumulator acc is split into a short instance part $\mathsf{acc}.\mathbb{x}$ and a (possibly) long witness part $\mathsf{acc}.\mathbb{w}$; (c) the verifier only needs the short parts of inputs and accumulators to verify an accumulation step, along with a short validity proof instead of the long witness parts.

As before, the prover is used to accumulate a new input q_i into a prior accumulator acc_{i-1} to obtain a new accumulator and validity proof $(\mathsf{acc}_i, \pi_{\mathrm{V},i}) \leftarrow \mathrm{P}(\mathsf{acc}_{i-1}, \mathsf{q}_i)$. Different from before, however, we wish to establish that there exists (or, more precisely, a party knows) witnesses $\mathsf{q}_1.\mathbb{w}, \ldots, \mathsf{q}_t.\mathbb{w}$ such that $\Phi(\mathsf{q}_1.\mathbb{x}, \mathsf{q}_1.\mathbb{w}) \wedge \cdots \wedge \Phi(\mathsf{q}_t.\mathbb{x}, \mathsf{q}_t.\mathbb{w}) = 1$, and for this it suffices to check that:

- the verifier accepts each accumulation step given only the short instance parts: $\mathrm{V}(\mathsf{acc}_0.\mathbb{x}, \mathsf{q}_1.\mathbb{x}, \mathsf{acc}_1.\mathbb{x}, \pi_{\mathrm{V},1}) = 1$, $\mathrm{V}(\mathsf{acc}_1.\mathbb{x}, \mathsf{q}_2.\mathbb{x}, \mathsf{acc}_2.\mathbb{x}, \pi_{\mathrm{V},2}) = 1$, and so on; and
- the decider accepts the final accumulator (made of both the instance and witness part): $\mathrm{D}(\mathsf{acc}_t) = 1$.

Again the naive cost $t \cdot |\Phi|$ is replaced with the new cost $t \cdot |\mathrm{V}| + |\mathrm{D}|$, but now it could be that an accumulator is, e.g., as large as $|\Phi|$; we only need the *instance part* of the accumulator (and predicate inputs) to be short.

The security property of a split accumulation scheme involves an extractor that outputs a long witness part from a short instance part and proof, and is reminiscent to the knowledge soundness of a succinct non-interactive argument. Turning this high level description into a working definition requires some care, however, and we view this as a contribution of this paper. (By "working definition" we mean a definition that we can provably fulfill under concrete hardness assumptions, and provably suffices for recursive composition.) Informally the security definition could be summarized as follows.

Definition 2.2 (informal)**.** *A* **split accumulation scheme** *for a predicate* $\Phi\colon X \times W \to \{0,1\}$ *consists of a triple of algorithms* $(\mathrm{P}, \mathrm{V}, \mathrm{D})$ *that satisfies the following properties.*

- Completeness: *For all accumulators* acc *and predicate inputs* $\mathsf{q} \in X \times W$*, if* $\mathrm{D}(\mathsf{acc}) = 1$ *and* $\Phi(\mathsf{q}) = 1$*, then for* $(\mathsf{acc}^\star, \pi_{\mathrm{V}}^\star) \leftarrow \mathrm{P}(\mathsf{acc}, \mathsf{q})$ *it holds that* $\mathrm{V}(\mathsf{acc}.\mathbb{x}, \mathsf{q}.\mathbb{x}, \mathsf{acc}^\star.\mathbb{x}, \pi_{\mathrm{V}}^\star) = 1$ *and* $\mathrm{D}(\mathsf{acc}^\star) = 1$.
- Knowledge: *For every efficiently-generated old accumulator instance* $\mathsf{acc}.\mathbb{x}$*, old input instance* $\mathsf{q}.\mathbb{x}$*, accumulation proof* $\pi_{\mathrm{V}}^\star$*, and new accumulator* $\mathsf{acc}^\star$*, if* $\mathrm{D}(\mathsf{acc}^\star) = 1$ *and* $\mathrm{V}(\mathsf{acc}.\mathbb{x}, \mathsf{q}.\mathbb{x}, \mathsf{acc}^\star.\mathbb{x}, \pi_{\mathrm{V}}^\star) = 1$ *then, with all but negligible probability, an efficient extractor can find old witness parts* $\mathsf{acc}.\mathbb{x}$ *and* $\mathsf{q}.\mathbb{x}$ *such that* $\Phi((\mathsf{q}.\mathbb{x}, \mathsf{q}.\mathbb{w})) = 1$ *and* $\mathrm{D}((\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})) = 1$.

One can verify that split accumulation is indeed a relaxation of atomic accumulation: any atomic accumulation scheme is (trivially) a split accumulation scheme with empty witnesses. Crucially, however, a split accumulation scheme alleviates a major restriction of atomic accumulation, namely, that the accumulator itself (and a predicate input) has to be short.

See Section 4 for details about the definition of split accumulation.

Next, in Section 2.2 we show that split accumulation suffices for recursive composition (which has surprising theoretical consequences) and then in Section 2.3 we present a NARK with split accumulation scheme based on discrete logarithms.

2.2 PCD from split accumulation

We summarize the main ideas behind Theorem 1, which obtains proof-carrying data (PCD) from any NARK that has a split accumulation scheme. To ease exposition, in this summary we focus on IVC, which can be viewed as the special case where a circuit F is repeatedly applied. That is, we wish to incrementally prove a claim of the form "$F^T(z_0) = z_T$" where F^T denotes F composed with itself T times.

Prior work: recursion via atomic accumulation. Our starting point is a theorem from [BCMS20] that obtains PCD from any SNARK that has an atomic accumulation scheme. The IVC construction implied by that theorem is roughly follows.

- The *IVC prover* receives as input a previous instance z_i, proof π_i, and accumulator acc_i; then accumulates (z_i, π_i) with acc_i to obtain a new accumulator acc_{i+1}; and finally generates a SNARK proof π_{i+1} of the claim: "$z_{i+1} = F(z_i)$, and there exist a proof π_i and an accumulator acc_i such that the accumulation verifier accepts $((z_i, \pi_i), \mathsf{acc}_i, \mathsf{acc}_{i+1})$", expressed as a circuit R (see Fig. 2, middle box). The IVC proof for z_{i+1} is $(\pi_{i+1}, \mathsf{acc}_{i+1})$.
- The *IVC verifier* validates an IVC proof (π_i, acc_i) for z_i by running the SNARK verifier on $((z_i, \mathsf{acc}_i), \pi_i)$ and running the accumulation scheme decider on acc_i.

In each iteration we maintain the invariant that if acc_i is a valid accumulator (according to the decider) and π_i is a valid proof, then the computation is correct up to the i-th step.

Note that while it would suffice to prove that "$z_{i+1} = F(z_i)$, π_i is a valid proof, and acc_i is a valid accumulator", we cannot afford to do so. Indeed: (i) proving that π_i is a valid proof requires proving a statement about the argument verifier, which may not be sublinear; and (ii) proving that acc_i is a valid accumulator requires proving a statement about the decider, which may not be sublinear. Instead of proving this claim directly, we "defer" it by having the prover accumulate (z_i, π_i) into acc_i to obtain a new accumulator acc_{i+1}. The soundness property of the accumulation scheme ensures that if acc_{i+1} is valid and the accumulation verifier accepts $((z_i, \pi_i), \mathsf{acc}_i, \mathsf{acc}_{i+1})$, then π_i is a valid proof and acc_i is a valid accumulator. Thus all that remains to maintain the invariant is for the prover to prove that the accumulation verifier accepts; this is possible provided that the *accumulation verifier* is sublinear.

Our construction: recursion via split accumulation. Our construction naturally extends the above idea in the setting of NARKs with split accumulation schemes. Indeed, the only difference to the above construction is that the proof π_{i+1} generated by the IVC prover is for the statement "$z_{i+1} = F(z_i)$, and there exist a proof *instance* $\pi_i.\mathbb{x}$ and an accumulator *instance* $\mathsf{acc}_i.\mathbb{x}$ such that the accumulation verifier accepts $((z_i, \pi_i.\mathbb{x}), \mathsf{acc}_i.\mathbb{x}, \mathsf{acc}_{i+1}.\mathbb{x})$", and accordingly the IVC verifier runs the NARK verifier on $((z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i)$. This is illustrated in Fig. 2 (lower box). Note that the circuit R itself is unchanged from the atomic case; the difference is in whether we pass the *entire* proof and accumulators or just the $\mathbb{x}$ part.

Proving that this relaxation yields a secure construction is more complex. Similar to prior work, the proof of security proceeds via a recursive extraction argument, as we explain next.

For an atomic accumulation scheme ([BCMS20]), one maintains the following extraction invariant: the i-th extractor outputs $(z_i, \pi_i, \mathsf{acc}_i)$ such that π_i is valid according to the SNARK, acc_i is valid according to the decider, and $F^{T-i}(z_i) = z_T$. The T-th "extractor" is simply the malicious prover, and we can obtain the i-th extractor by applying the knowledge guarantee of the SNARK to the $(i+1)$-th extractor. That the invariant is maintained is implied by the soundness guarantee of the atomic accumulation scheme.

For a split accumulation scheme, we want to maintain the same extraction invariant; however, the extractor for the NARK will only yield $(z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x})$, and not the corresponding witnesses. This is where we make use of the extraction property of the split accumulation scheme itself. Specifically, we interleave the knowledge guarantees of the NARK and accumulation scheme as follows: the i-th NARK extractor is obtained from the $(i+1)$-th accumulation extractor using the knowledge guarantee of the NARK, and the i-th accumulation extractor is obtained from the i-th NARK extractor using the knowledge guarantee of the accumulation scheme. We take the malicious prover to be the T-th accumulation extractor.

From sketch to proof. In Section 5, we give the formal details of our construction and a proof of correctness. In particular, we show how to construct PCD, a more general primitive than IVC. In the PCD setting, rather than each computation step having a single input z_i, it receives m inputs from different nodes. Proving correctness hence requires proving that *all* of these inputs were computed correctly. For our construction, this entails checking m proofs and m accumulators. To do this, we extend the definition of an accumulation scheme to allow accumulating multiple instance-proof pairs and multiple "old" accumulators.

We also note that the application to PCD leads to other definitional considerations, which are similar to those that have appeared in previous works [COS20; BCMS20]. In particular, the knowledge soundness guarantee for both the NARK *and* the accumulation scheme should be of the stronger "witness-extended emulation with auxiliary input and output" type used in previous work. Additionally, the underlying construction of split accumulation achieves only expected polynomial-time extraction (in the ROM), and so the recursive extraction technique requires that we are able to extract from expected-time adversaries.

Remark 2.3 (flavors of PCD)**.** The recent advances in PCD from accumulation achieve weaker forms than PCD from succinct verification, and formally these results are incomparable. (Starting from weaker assumptions they obtain weaker conclusions.) The essential feature that all these works achieve is that the efficiency of PCD algorithms is independent of the number of nodes in the PCD computation, which is how PCD is defined (see Section 3.2). That said, prior work on PCD from succinct verification [BCCT13; BCTV14; COS20] additionally guarantees that verifying a PCD proof is sublinear in a node's computation; and prior work on PCD from atomic accumulation [BCMS20] merely ensures that a PCD proof has size (but not necessarily verification time) that is sublinear in a node's computation. The PCD scheme obtained in this paper does not have these additional features: a PCD proof has size that is linear in a node's computation.

2.3 NARK with split accumulation based on DL

We summarize the main ideas behind Theorem 2, which provides, in the discrete logarithm setting with random oracles, a NARK for R1CS that has a split accumulation scheme whose accumulation verifier uses a constant number of group scalar multiplications. We describe the NARK and then explain at high level how we accumulate the verifier's computation. This in turn motivates accumulation for polynomial commitments based on Pedersen commitments, which we address in the next subsection (Section 2.4).

recursion circuit via succinct verification

$R\big((\text{ivk}, z_i), (z_{i-1}, \pi_{i-1})\big)$:
- $z_i = F(z_{i-1})$
- $\mathsf{SNARK}.\mathcal{V}(\text{ivk}, z_{i-1}, \pi_{i-1}) = 1$

recursion circuit via atomic accumulation

$R\big((\mathsf{avk}, z_i, \mathsf{acc}_i), (z_{i-1}, \pi_{i-1}, \mathsf{acc}_{i-1}, \pi_{\mathrm{V},i})\big)$:
- $z_i = F(z_{i-1})$
- $\mathsf{ACC}.\mathrm{V}\big(\mathsf{avk}, ((\mathsf{avk}, z_{i-1}, \mathsf{acc}_{i-1}), \pi_{i-1}), \mathsf{acc}_{i-1}, \mathsf{acc}_i, \pi_{\mathrm{V},i}\big) = 1$

recursion circuit via split accumulation

$R\big((\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), (z_{i-1}, \pi_{i-1}.\mathbb{x}, \mathsf{acc}_{i-1}.\mathbb{x}, \pi_{\mathrm{V},i})\big)$:
- $z_i = F(z_{i-1})$
- $\mathsf{ACC}.\mathrm{V}\big(\mathsf{avk}, ((\mathsf{avk}, z_{i-1}, \mathsf{acc}_{i-1}.\mathbb{x}), \pi_{i-1}.\mathbb{x}), \mathsf{acc}_{i-1}.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}, \pi_{\mathrm{V},i}\big) = 1$

Figure 2: Comparison of circuits used to realize recursion with different techniques.

We highlight here that both the NARK and the accumulation scheme are particularly simple compared to other protocols in the SNARK literature (especially with regard to recursion!), and view this as a significant advantage for potential practical deployments.

NARK for R1CS. Recall that R1CS is a standard generalization of arithmetic circuit satisfiability where the "circuit description" is given by coefficient matrices, as below; there "$\circ$" denotes the entry-wise product.

Definition 2.4 (R1CS problem)**.** *Given a finite field $\mathbb{F}$, coefficient matrices $A, B, C \in \mathbb{F}^{m \times n}$ and an instance vector $x \in \mathbb{F}^{\ell}$, is there a witness vector $w \in \mathbb{F}^{n-\ell}$ such that $z := (x, w) \in \mathbb{F}^n$ and $Az \circ Bz = Cz$?*

The NARK for R1CS in Theorem 2 is obtained by applying the Fiat–Shamir transformation to a sigma protocol for R1CS based on Pedersen commitments. We provide the sigma protocol in Figure 3. The public parameters pp are those for the Pedersen commitment to have message space $\mathbb{F}^{m+1}$ (i.e., pp contains a group description and $m+1$ random group elements) and Commit its commitment function; also, $V_H(X)$ denotes the vanishing polynomial of a subset $H \subseteq \mathbb{F}$ of size m.

The efficiency is evident from the construction: quasilinear prover time (due to polynomial arithmetic); linear argument size (the first message is succinct but the third message is not); and linear verifier time (more precisely, linear number of group scalar multiplications). In more detail, denoting by k the number of non-zero entries in the coefficient matrices $A, B, C \in \mathbb{F}^{m \times n}$, the asymptotic efficiency is as follows:

NARK prover time	NARK verifier time	NARK argument size
$O(m+n)\ \mathbb{G}$	$O(m+n)\ \mathbb{G}$	$O(1)\ \mathbb{G}$
$O(m \log m + n + k)\ \mathbb{F}$	$O(m+n+k)\ \mathbb{F}$	$O(m+n)\ \mathbb{F}$

The sigma protocol may superficially appear useless because it has linear argument size and is not zero knowledge as stated (though it could be modified to be); in fact, the prover commits to the witness (and more) in the first message, only to send "everything" to the verifier in the next round, so that the verifier is (almost) checking the R1CS relation itself!

How is this "better" than the prover simply sending the witness as a non-interactive proof?

The key point is that we do not know how to obtain a split accumulation scheme for the computation that directly checks the witness but we do know how to obtain one for the NARK verifier in Figure 3.

Split accumulation for the NARK. We describe a split accumulation scheme for the NARK for R1CS with the efficiency as reported in the table below (where we additionally report the number of oracle calls).

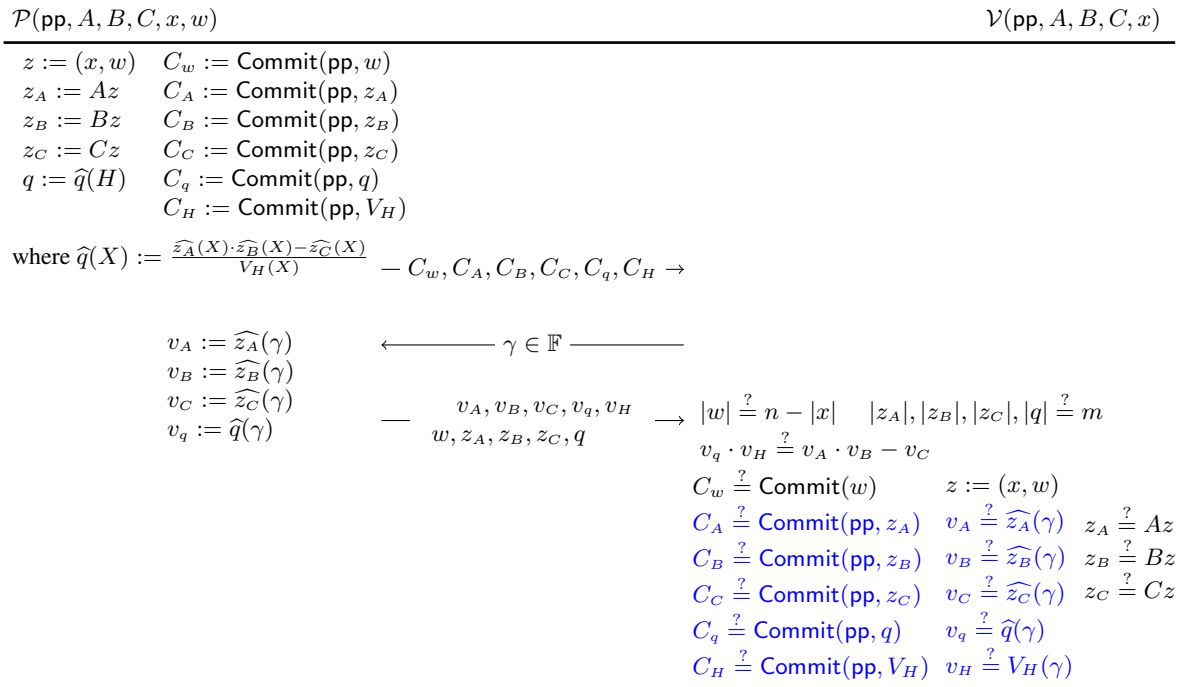

Figure 3: The sigma protocol for R1CS that underlies the NARK for R1CS.

accumulation prover time (per accumulated NARK)	accumulation verifier time (per accumulated NARK)	decider time	accumulator size
$O(m+n)$ $\mathbb{G}$	$O(1)$ $\mathbb{G}$	$O(m+n)$ $\mathbb{G}$	$\lvert\mathsf{acc.x}\rvert = O(1)$ $\mathbb{G} + O(1)$ $\mathbb{F}$
$O(m+n+k)$ $\mathbb{F}$	$O(1)$ $\mathbb{F}$	$O(m+n+k)$ $\mathbb{F}$	$\lvert\mathsf{acc.w}\rvert = O(m+n)$ $\mathbb{F}$
$O(1)$ RO	$O(1)$ RO	–	–

We view the first message of the protocol as a succinct instance and the third message of the protocol as a non-succinct "opening" witness aimed at making the NARK verifier accept. We now sketch how to design a split accumulation scheme for the relation implied by this instance-witness split of the NARK verifier; technical details are in Appendix A.

The NARK verifier performs four types of checks:

1. a test for the polynomial equation $\widehat{q}(X) \cdot V_H(X) = \widehat{z_A}(X) \cdot \widehat{z_B}(X) - \widehat{z_C}(X)$;
2. checks for the linear equations $z_A = Az, z_B = Bz, z_C = Cz$;
3. checking a commitment ($C_w \stackrel{?}{=} \mathsf{Commit}(w)$);
4. several checks of the form "$C = \mathsf{Commit}(f)$ and $\hat{f}(\gamma) = v$" (highlighted in blue in Figure 3).

The first check is cheap so does not have to be accumulated (the accumulation verifier directly performs this check). The second and third checks are linear and so can be easily (split) accumulated via random linear combinations. The last type of check is concerned with *evaluation claims about committed polynomials*.[1]

[1]Note that it may appear odd that these include even a claim about the vanishing polynomial V_H because, for suitable choices of H, can be evaluated in only $\log |H| = \log m$ field operations directly. This is because, similarly to an optimization in [CCDW20], by having the prover commit to the vanishing polynomial, we can ensure later on that the accumulation verifier only has to perform $O(1)$ field operations rather than $O(\log m)$.

This requires some work: designing a highly-efficient split accumulation scheme for the "trivial" polynomial commitment scheme based on Pedersen commitments — we discuss this in the next section.

Remark 2.5. If one replaced the (succinct) polynomial commitment scheme that underlies the preprocessing SNARK in [CHMMVW20] with a (non-succinct) Pedersen polynomial commitment scheme then one would obtain a NARK for R1CS with a split accumulation scheme whose accumulation verifier *is* of constant size but other asymptotics would be worse compared to Theorem 2. First, the cryptographic costs and the quasilinear costs of the NARK and accumulation scheme would also grow in the number k of non-zero entries in the coefficient matrices, which can be much larger than m and n (asymptotically and concretely). Second, the NARK verifier and accumulation decider would additionally use a quasilinear number of field operations due to FFTs (to run the the indexer from that paper). Finally, in addition to poorer asymptotics, this approach would lead to a concretely more expensive accumulation verifier and overall a more complex protocol.

2.4 Split accumulation for Pedersen polynomial commitments

Any commitment implies a "trivial" polynomial commitment scheme: commit to the polynomial (in coefficient or evaluation form), and then reveal the entire polynomial (the receiver can then check the evaluation claim directly). We fix the commitment to be a Pedersen commitment (what we used in Section 2.3), which implies that commitments are linear in the underlying polynomials. We prove that this simple polynomial commitment scheme has a very efficient split accumulation scheme in the random oracle model.

Theorem 3 (informal)**.** *The Pedersen polynomial commitment scheme has a split accumulation scheme* $\mathsf{AS}_{\mathsf{HC}}$ *that is secure in the random oracle model (and assuming the hardness of the discrete logarithm problem). Verifying accumulation requires* 2 *group scalar multiplications and* $O(1)$ *field additions/multiplications per claim, and results in an accumulator whose instance part is* 1 *group element and* 2 *field elements and whose witness part is* d *field elements. (See Table 1.)*

In Table 1 we compare our the efficiency of the split accumulation scheme $\mathsf{AS}_{\mathsf{HC}}$ for the (non-succinct) Pedersen PC scheme $\mathsf{PC}_{\mathsf{HC}}$ and the efficiency of the atomic accumulation scheme $\mathsf{AS}_{\mathsf{IPA}}$ in [BCMS20] for the (succinct) PC scheme $\mathsf{PC}_{\mathsf{IPA}}$ based on the inner-product argument on cyclic groups [BCCGP16; BBBPWM18; WTSTW18]. The takeaway is that accumulation verifier for $\mathsf{AS}_{\mathsf{HC}}$ is significantly cheaper than the accumulation verifier for $\mathsf{AS}_{\mathsf{IPA}}$.

Technical details are in Section 6; in the rest of this section we sketch the ideas behind Theorem 3.

accumulation scheme	type	assumption	accumulation prover (per claim)	accumulation verifier (per claim)	accumulation decider	accumulator size instance	accumulator size witness
$\mathsf{AS}_{\mathsf{IPA}}$ [BCMS20]	atomic	DLOG + RO †	$O(\log d)$ $\mathbb{G}$ $O(d)$ $\mathbb{F}$ $[+O(d)$ $\mathbb{G}$ per accumulation$]$	$O(\log d)$ $\mathbb{G}$ $O(\log d)$ $\mathbb{F}$ $O(\log d)$ RO	$O(d)$ $\mathbb{G}$ $O(d)$ $\mathbb{F}$	1 $\mathbb{G}$ $O(\log d)$ $\mathbb{F}$	0
$\mathsf{AS}_{\mathsf{HC}}$ [this work]	split	DLOG + RO	$O(d)$ $\mathbb{G}$ $O(d)$ $\mathbb{F}$	2 $\mathbb{G}$ $O(1)$ $\mathbb{F}$ 2 RO	$O(d)$ $\mathbb{G}$ $O(d)$ $\mathbb{F}$	1 $\mathbb{G}$ 2 $\mathbb{F}$	d $\mathbb{F}$

Table 1: Efficiency comparison between the atomic accumulation scheme $\mathsf{AS}_{\mathsf{IPA}}$ for $\mathsf{PC}_{\mathsf{IPA}}$ in [BCMS20] and the split accumulation scheme $\mathsf{AS}_{\mathsf{HC}}$ for $\mathsf{PC}_{\mathsf{HC}}$ in this work. Above $\mathbb{G}$ denotes group scalar multiplications or group elements, and $\mathbb{F}$ denotes field operations or field elements. (†: $\mathsf{AS}_{\mathsf{IPA}}$ relies on knowledge soundness of $\mathsf{PC}_{\mathsf{IPA}}$, which results from applying the Fiat–Shamir transformation to a logarithmic-round protocol. The security of this protocol has only been proven via a superpolynomial-time extractor [BMMTV19] or in the algebraic group model [GT20].)

2.4.1 A simple linear polynomial commitment scheme

A polynomial commitment scheme is a cryptographic primitive that enables one to produce a commitment C to a polynomial p, and then to prove that this committed polynomial evaluates to a claimed value v at a desired point z. (See Section 3.6 for a definition.)

Any commitment scheme Commit implies a "trivial" commitment scheme: in the commit phase, the sender commits to the coefficients of the polynomial p; in the reveal phase, the sender sends the entire polynomial p as opening, and the receiver can itself check that $p(z) = v$ as claimed.[2] (We are assuming that the message space of Commit is $\mathbb{F}^n$ where $\mathbb{F}$ is the field over which p is defined and that $n \geq \deg(p) + 1$.)

If additionally we know that Commit is linear (i.e., for every a and b in $\mathbb{F}^n$, $\mathsf{Commit}(a) + \mathsf{Commit}(b) = \mathsf{Commit}(a+b)$) then the trivial construction yields a (trivial) linear polynomial commitment scheme. Typically the commitment space is a cryptographic group $\mathbb{G}$ (so addition is over the group), and a natural example is the Pedersen commitment scheme, which is secure provided the discrete logarithm problem is hard in $\mathbb{G}$.

We refer to the foregoing trivial scheme as $\mathsf{PC}_{\mathsf{HC}}$, and next discuss accumulation for it.

2.4.2 Split accumulation for $\mathsf{PC}_{\mathsf{HC}}$

An (atomic or split) accumulation scheme for a PC scheme accumulates claims of the form "I know a polynomial p of degree at most d such that C is a commitment to p and $p(z) = v$". We summarize such an *evaluation claim* via the tuple (C, z, v, d). Below we summarize our split accumulation scheme for $\mathsf{PC}_{\mathsf{HC}}$.

First we describe a simple public-coin interactive reduction for combining two or more evaluation claims into a single evaluation claim, and then explain how this interactive reduction gives rise to the split accumulation scheme. We prove security in the random oracle model, using an expected-time extractor.

Batching evaluation claims. First consider two evaluation claims (C_1, z, v_1, d) and (C_2, z, v_2, d) for the *same* evaluation point z (and degree d). We can use a random challenge $\alpha \in \mathbb{F}$ to combine these claims into one claim (C', z, v', d) where $C' := C_1 + \alpha C_2$ and $v' := v_1 + \alpha v_2$. If either of the original claims does not hold then, with high probability over the choice of α, neither does the new claim. This idea extends to any number of claims for the same evaluation point, by taking $C' := \sum_i \alpha^i C_i$ and $v' := \sum_i \alpha^i v_i$.

Next consider two evaluation claims (C_1, z_1, v_1, d) and (C_2, z_2, v_2, d) at (possibly) different evaluation points z_1 and z_2. We explain how these can be combined into four claims all at the *same* point. Below we use the fact that $p(z) = v$ if and only if there exists a polynomial $w(X)$ such that $p(X) = w(X) \cdot (X - z) + v$.

Let $p_1(X)$ and $p_2(X)$ be the polynomials "inside" C_1 and C_2, respectively, that are known to the prover.

1. The prover computes the witness polynomials $w_1 := \frac{p_1(X) - v_1}{X - z_1}$ and $w_2 := \frac{p_2(X) - v_2}{X - z_2}$ and sends the commitments $W_1 := \mathsf{Commit}(w_1)$ and $W_2 := \mathsf{Commit}(w_2)$.

2. The verifier sends a random evaluation point $z^* \in \mathbb{F}$.

3. The prover computes and sends the evaluations $y_1 := p_1(z^*), y_2 := p_2(z^*), y_1' := w_1(z^*), y_2' := w_2(z^*)$.

4. The verifier checks the relation between each witness polynomial and the original polynomial at the random evaluation point z^*:

$$y_1 = y_1'(z^* - z_1) + y_1' \quad \text{and} \quad y_2 = y_2'(z^* - z_2) + y_2' \ .$$

[2] Alternatively, and this is what we actually use in Section 2.3 (though we ignore the difference in this high level overview), the sender commits to the evaluation of the polynomial, and subsequently the receiver checks that extending the revealed evaluation table at z yields the claimed value v. Over suitable domains this operation is also linear time.

Next the verifier outputs four evaluation claims for $p_1(z^*) = y_1, p_2(z^*) = y_2, w_1(z^*) = y'_1, w_2(z^*) = y'_2$:

$$(C_1, z^*, y_1, d)\ ,\ (C_2, z^*, y_2, d)\ ,\ (W_1, z^*, y'_1, d)\ ,\ (W_2, z^*, y'_2, d)\ .$$

More generally, we can reduce m evaluation claims at m points to $2m$ evaluation claims all at the same point.

By combining the two techniques, one obtains a public-coin interactive reduction from any number of evaluation claims (regardless of evaluation points) to a single evaluation claim.

Split accumulation. The batching protocol described above yields a split accumulation scheme for $\mathsf{PC}_{\mathsf{HC}}$ (and, more generally, any linear polynomial commitment scheme) in the random oracle model. The accumulated predicate Φ takes inputs q whose instance part is an evaluation claim $\mathsf{q}.\mathbb{x} = (C, z, v, d)$ and whose witness part is a polynomial $\mathsf{q}.\mathbb{w} = p(X)$, and accepts if and only if $C = \mathsf{Commit}(p)$ and $p(z) = v$. An accumulator acc has the same form as a predicate input: the instance part is an evaluation claim and the witness part is a polynomial. Next we describe the algorithms of the accumulation scheme.

- The accumulation prover P runs the interactive reduction by relying on the random oracle to generate the random verifier messages (i.e., it applies the Fiat–Shamir transformation to the reduction), in order to combine the instance parts of old accumulators and inputs to obtain the instance part of a new accumulator. Then P also combines the committed polynomials using the same linear combinations in order to derive the new committed polynomial, which is the witness part of the new accumulator.
- The accumulation verifier V checks that the challenges were correctly computed from the random oracle, and performs the checks of the reduction (the claims were correctly combined and that the proper relation between each y_i, y'_i, z_i, z^* holds).
- The accumulation decider D reads the accumulation in its entirety and checks that the polynomial (the witness part) satisfies the evaluation claim (the instance part). (Here the random oracle is not used.)

Efficiency. The efficiency claimed in Theorem 3 (and Table 1) is evident from the construction. The accumulation prover P computes $n + m$ commitments to polynomials when combining n old accumulators and m predicate inputs (all polynomials are for degree at most d). An accumulator consists of $O(d)$ group elements, with the (short) instance part consisting of 1 group element and 2 field elements. The accumulator decider D computes 1 commitment (and 1 polynomial evaluation at 1 point) in order to validate an accumulator. Finally, the cost of running the accumulator verifier V is dominated by $2(n + m)$ scalar multiplication of the linear commitments.

2.4.3 Security

While proving the completeness property of our split accumulation scheme is straightforward, proving its knowledge property is less so. Given an adversary that produces evaluation claims (C_i, z_i, v_i, d_i), a single claim (C, z, v, d) which passes the checks in the reduction procedure, and a polynomial s with $s(z) = v$ to which C is a commitment, we obtain an extractor which outputs polynomials p_i with $p_i(z_i) = v_i$ to which C_i is a commitment. Our security proof (detailed in Section 6.3) works in the random oracle model, assuming only the hardness of the discrete logarithm problem.

The extractor obtains $2n$ polynomials $s^{(j)}$ for the same evaluation point z^* but distinct challenges α_j, where n is the number of evaluation claims. The checks in the reduction procedure imply that $s^{(j)} = \sum_i \alpha_j^i p_i + \sum_i \alpha_j^{n+i} w_i$, where w_i is the witness corresponding to p_i; hence we can recover the p_i, w_i by solving a linear system. We then use a variant of the zero-finding game lemma from [BCMS20] (see

Section 6.3.3) to show that if a particular polynomial equation on p_i, w_i holds at the point z^* obtained from the random oracle, it must with overwhelming probability be an identity. Applying this to the equation induced by the reduction shows that the original claims hold with high probability.

The proof crucially relies on the ability to efficient generate the $2n$ related proofs. For this we formulate a new expected-time forking lemma in the random oracle model, which is informally stated below (and detailed in Section 6.3.2). For this application, the set L will be valid accumulators and proofs.

Lemma 1 (informal)**.** *Let L be an efficiently recognizable set. Suppose there exists an expected polynomial time algorithm with access to a random oracle ρ that outputs, with probability δ, a tuple $(q, \alpha, y) \in L$ such that $\rho(q) = \alpha$. Then there exists an expected polynomial time extractor E such that for $N = \mathrm{poly}(\lambda)$, E outputs N related tuples $(q, \alpha_i, y_i) \in L$ for distinct α_i with probability at least $\delta - \mathrm{negl}(\lambda)$.*

This forking lemma differs from prior forking lemmas in three significant ways. First, it is in the random oracle model rather than the interactive setting (unlike [BCCGP16]). Second, we can obtain any polynomial number of accepting transcripts in polynomial time with only negligible loss in success probability (unlike [BN06]). Finally, it holds even if the adversary itself runs in expected (as opposed to strict) polynomial time. This is important for our application to PCD where the extractor of one recursive step becomes the adversary of the next recursive step.

2.5 Implementation and evaluation

We elaborate on our implementation and evaluation of accumulation schemes and their application to PCD.

The case for a PCD framework. Different PCD constructions offer different trade-offs. The tradeoffs are both about asymptotics (see Remark 2.3) and about practical concerns, as we review below.

- *PCD from sublinear verification* [BCCT13; BCTV14; COS20] is typically instantiated via preprocessing SNARKs based on pairings.[3] This route offers excellent verifier time (a few milliseconds regardless of the computation at a PCD node) but requires a circuit-specific trusted setup (which complicates deployment) and cycles of pairing-friendly elliptic curves (which are costly in terms of group arithmetic and size).
- *PCD from atomic accumulation* [BCMS20] can, e.g., be instantiated via SNARKs based on cyclic groups [BGH19]. This route offers a transparent setup (easy to deploy) and logarithmic-size arguments (a few kilobytes even for large computations), using cycles of standard elliptic curves (more efficient than their pairing-friendly counterparts). On the other hand, this route yields linear verification times (expensive for large computations) and logarithmic costs for accumulation (increasing the cost of recursion).
- *PCD from split accumulation* (this work) can, e.g., be instantiated via NARKs based on cyclic groups. This route still offers a transparent setup and allows using cycles of standard elliptic curves. Moreover, it offers constant costs for accumulation, but at the expense of argument size, which is now linear.

It would be desirable to have a *single framework that supports different PCD constructions via a modular composition of simpler building blocks*. It would allow for replacing older building blocks with new ones. It would enable prototyping different PCD constructions for different applications (which may have different needs) and thereby enable practitioners to make informed choices about which PCD construction is best for them. It would facilitate auditing of what would otherwise be a complex cryptographic system with many intermixed layers. (Realizing even a single PCD construction is a substantial implementation task.) Finally, it would be useful for applications to be separated from the underlying recursion via a common PCD interface.

[3] Instantiations based hashes are also possible [COS20] but are (post-quantum and) less efficient.

Implementation (Section 7). The foregoing considerations motivated our implementation efforts for PCD. Our code base has two main parts, one for realizing accumulation schemes and another for realizing PCD from accumulation (which we integrated with PCD from succinct verification).

- *Framework for accumulation.* We designed a modular framework for (atomic and split) accumulation schemes, and use it to provide a common interface for accumulation schemes for popular polynomial commitments (and thus for SNARKs that use these), including: (a) the atomic accumulation scheme $\mathsf{AS}_{\mathsf{AGM}}$ in [BCMS20] for the PC scheme $\mathsf{PC}_{\mathsf{AGM}}$; (b) the atomic accumulation scheme $\mathsf{AS}_{\mathsf{IPA}}$ in [BCMS20] for the PC scheme $\mathsf{PC}_{\mathsf{IPA}}$; (c) the split accumulation scheme $\mathsf{AS}_{\mathsf{HC}}$ in this paper for $\mathsf{PC}_{\mathsf{HC}}$. Our framework also provides a generic method for defining R1CS constraints for the verifiers of these accumulation schemes; we leverage this to implement R1CS constraints for all these accumulation schemes.

- *PCD from accumulation.* We use the foregoing framework to implement a generic construction of PCD from accumulation. We support the PCD construction of [BCMS20] (which uses atomic accumulation) and the PCD construction in this paper (which uses split accumulation). Our code builds on and extends an existing PCD library.[4] Our implementation is modular: it takes as ingredients an implementation of any NARK, an implementation of any accumulation scheme for that NARK, and constraints for the accumulation verifier, and produces a concrete PCD construction. We use our concrete instantiations of these ingredients to construct PCD that is based on atomic accumulation of $\mathsf{PC}_{\mathsf{AGM}}$ and $\mathsf{PC}_{\mathsf{IPA}}$, and split accumulation of $\mathsf{PC}_{\mathsf{HC}}$.

Evaluation for DL setting (Section 8). When realizing PCD in practice the main goal is to "minimize the cost of recursion", that is, to minimize the number of constraints that need to be recursively proved in each PCD step (excluding the constraints for the application) without hurting other parameters too much (prover time, argument size, and so on). We evaluate our implementation with respect to this goal, with a focus on understanding the trade-offs between atomic and split accumulation *in the discrete logarithm setting*.

The DL setting is of particular interest to practitioners, as it leads to systems with a transparent (public-coin) setup that can be based on efficient cycles of (standard) elliptic curves [BGH19; Hop20]; indeed, some projects are developing real-world systems that use PCD in the DL setting [Halo20; Pickles20]. The main drawback of the DL setting is that verification time (and sometimes argument size) is linear in a PCD node's computation. This inefficiency is, however, tolerable if a PCD node's computation is not too large, as is the case in the aforementioned projects. (Especially so when taking into account the disadvantages of PCD based on pairings, which involves relying on a circuit-specific trusted setup and more expensive curve cycles.)

We compare two routes for PCD in the DL setting:

- recursion based on the atomic accumulation scheme for the PC scheme $\mathsf{PC}_{\mathsf{IPA}}$ [BCMS20];
- recursion based on the split accumulation scheme for $\mathsf{PC}_{\mathsf{HC}}$ (Section 6).

Our evaluation focuses on these polynomial commitment schemes and their corresponding accumulation schemes, as they comprise the primary cost of recursion in popular accumulation-friendly (S)NARKs, and demonstrates that the constraint cost of the $\mathsf{AS}_{\mathsf{HC}}$ accumulation verifier is 8 to 20 times cheaper than that of the $\mathsf{AS}_{\mathsf{IPA}}$ accumulation verifier.

We emphasize that the key aspect to focus on is the relative costs of the two components. This is because the cost of both components is dominated by the cost of many common subcomponents, and so improvements in these subcomponents will preserve the relative cost. For example, applying existing techniques [Halo20; Pickles20] for optimizing scalar multiplications should benefit both schemes in a similar way.

[4] `https://github.com/arkworks-rs/pcd`

3 Preliminaries

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

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

Random oracles. We denote by $\mathcal{U}(\lambda)$ the set of all functions that map $\{0,1\}^*$ to $\{0,1\}^\lambda$. We denote by $\mathcal{U}(*)$ the set $\bigcup_{\lambda \in \mathbb{N}} \mathcal{U}(\lambda)$. A *random oracle* with security parameter λ is a function $\rho \colon \{0,1\}^* \to \{0,1\}^\lambda$ sampled uniformly at random from $\mathcal{U}(\lambda)$.

3.1 Non-interactive arguments in the ROM

A tuple of algorithms $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ is a (preprocessing) *non-interactive argument* in the random oracle model (ROM) for an indexed relation $\mathcal{R}$ if the following properties hold.

- **Completeness.** For every adversary $\mathcal{A}$,

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

- **Soundness.** For every polynomial-size adversary $\tilde{\mathcal{P}}$,

$$\Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}) & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge & \mathsf{pp} \leftarrow \mathcal{G}^\rho(1^\lambda) \\ \mathcal{V}^\rho(\mathrm{ivk}, \mathbb{x}, \pi) = 1 & (\mathbb{i}, \mathbb{x}, \pi) \leftarrow \tilde{\mathcal{P}}^\rho(\mathsf{pp}) \\ & (\mathrm{ipk}, \mathrm{ivk}) \leftarrow \mathcal{I}^\rho(\mathsf{pp}, \mathbb{i}) \end{array}\right] \leq \mathrm{negl}(\lambda) \ .$$

The above formulation of completeness allows $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ to depend on the random oracle ρ and public parameters pp, and the above formulation of soundness allows $(\mathbb{i}, \mathbb{x})$ to depend on the random oracle ρ and public parameters pp.

Our PCD construction makes use of the stronger property of *knowledge soundness*, and optionally also the property of (statistical) *zero knowledge*. We define both of these properties below. Note that this definition is stronger the standard definition of knowledge soundness; this is required to prove post-quantum security in Theorem 5.2. This stronger definition is similar to the notion of *witness-extended emulation* [Lin03]. We refer to an argument with knowledge soundness as a NARK (non-interactive argument of knowledge) whereas an argument that just satisfies soundness is a NARG.

Knowledge soundness. We say that $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ has *knowledge soundness* (with respect to auxiliary input distribution $\mathcal{D}$) if for every expected polynomial time adversary $\tilde{\mathcal{P}}$ there exists an expected

polynomial time extractor $\mathcal{E}$ such that for every set Z,

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

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

Zero knowledge. We say that $\mathsf{ARG}=(\mathcal{G},\mathcal{I},\mathcal{P},\mathcal{V})$ has (statistical) zero knowledge if there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that for every polynomial-size honest adversary $\mathcal{A}$ the distributions below are computationally indistinguishable:

$$\left\{(\rho,\mathsf{pp},\mathbb{i},\mathbb{x},\pi)\;\middle|\;\begin{array}{r}\rho\leftarrow\mathcal{U}(\lambda)\\ \mathsf{pp}\leftarrow\mathcal{G}^\rho(1^\lambda)\\ (\mathbb{i},\mathbb{x},\mathbb{w})\leftarrow\mathcal{A}^\rho(\mathsf{pp})\\ (\mathrm{ipk},\mathrm{ivk})\leftarrow\mathcal{I}^\rho(\mathsf{pp},\mathbb{i})\\ \pi\leftarrow\mathcal{P}^\rho(\mathrm{ipk},\mathbb{x},\mathbb{w})\end{array}\right\}\quad\text{and}\quad\left\{(\rho[\mu],\mathsf{pp},\mathbb{i},\mathbb{x},\pi)\;\middle|\;\begin{array}{r}\rho\leftarrow\mathcal{U}(\lambda)\\ (\mathsf{pp},\tau)\leftarrow\mathcal{S}^\rho(\mathbb{i},\mathbb{x})\\ (\mathbb{i},\mathbb{x},\mathbb{w})\leftarrow\mathcal{A}^\rho(\mathsf{pp})\\ (\pi,\mu)\leftarrow\mathcal{S}^\rho(\mathsf{pp},\mathbb{i},\mathbb{x},\tau)\end{array}\right\}\ .$$

Above, $\rho[\mu]$ is the function that, on input x, equals $\mu(x)$ if μ is defined on x, or $\rho(x)$ otherwise. This definition uses explicitly-programmable random oracles [BR93]. (Non-interactive zero knowledge with non-programmable random oracles is impossible for non-trivial languages [Pas03; BCS16].)

3.2 Proof-carrying data

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

Definition 3.1. *A* **transcript** T *is a directed acyclic graph where each vertex $u\in V(\mathsf{T})$ is labeled by local data $z^{(u)}_{\mathsf{loc}}$ and each edge $e\in E(\mathsf{T})$ is labeled by a message $z^{(e)}\neq\bot$. The* **output** *of a transcript* T*, denoted* $\mathsf{o}(\mathsf{T})$*, is $z^{(e)}$ where $e=(u,v)$ is the lexicographically-first edge such that v is a sink.*

Definition 3.2. *A vertex $u\in V(\mathsf{T})$ is* **φ-compliant** *for $\varphi\in\mathsf{F}$ if for all outgoing edges $e=(u,v)\in E(\mathsf{T})$:*

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

We say that T *is* **φ-compliant** *if all of its vertices are φ-compliant.*

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

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

Knowledge soundness. We say that $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ has knowledge soundness (with respect to auxiliary input distribution $\mathcal{D}$) if for every expected polynomial-time (non-uniform) adversary $\tilde{\mathbb{P}}$, there exists an expected polynomial-time extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ such that for every set Z,

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

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

Zero knowledge. We say that $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ has (statistical) zero knowledge if there exists a probabilistic polynomial-time simulator $\mathbb{S}$ such that for all honest adversaries $\mathcal{A}$ the distributions below are statistically close:

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

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

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

3.3 Instantiating the random oracle

Almost all of the results in this paper are proved in the *random oracle model*, and so we give definitions which include random oracles. The single exception is our construction of proof-carrying data, in Section 5.1. We do not know how to build PCD schemes which are secure in the random oracle model from any standard assumption. Instead, we show that assuming the existence of a non-interactive argument with security in the standard (CRS) model, we obtain a PCD scheme which is also secure in the standard (CRS) model.

For this reason, the definition of PCD above is stated in the standard model (without oracles). We do not explicitly define non-interactive arguments in the standard model; the definition is easily obtained by removing the random oracle from the definition presented in Section 3.1.

3.4 Post-quantum security

The definitions of both non-interactive arguments (in the standard model) and proof-carrying data can be strengthened, in a straightforward way, to express post-quantum security. In particular, we replace "polynomial-size circuit" and "polynomial-time algorithm" with their quantum analogues. Since we do not prove post-quantum security of any construction in the random oracle model, we do not discuss the quantum random oracle model.

3.5 Commitment schemes

A commitment scheme $\mathsf{CM} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit})$ enables one to create binding commitments to messages.

- $\mathsf{CM.Setup}$, on input a *message format* L, outputs public parameters pp; this specifies a message universe $\mathcal{M}_{\mathsf{pp}}$ and a commitment universe $\mathcal{C}_{\mathsf{pp}}$.
- $\mathsf{CM.Trim}$, on input public parameters pp and a *trim specification* ℓ, outputs a commitment key ck containing a description of a message space $\mathcal{M}_{\mathsf{ck}} \subseteq \mathcal{M}_{\mathsf{pp}}$ (corresponding to ℓ).
- $\mathsf{CM.Commit}$, on input a commitment key ck, a message $m \in \mathcal{M}_{\mathsf{ck}}$ and randomness ω, outputs a commitment $C \in \mathcal{C}_{\mathsf{pp}}$.

CM is binding if, for every message format L such that $|L| = \mathrm{poly}(\lambda)$, and for every efficient adversary $\mathcal{A}$, the following holds.

$$\Pr\left[\begin{array}{c} m_1 \in \mathcal{M}_{\mathsf{ck}_1}\,,\, m_2 \in \mathcal{M}_{\mathsf{ck}_2}\,,\, m_1 \neq m_2 \\ \wedge \\ \mathsf{CM.Commit}(\mathsf{ck}_1, m_1;\, \omega_1) = \mathsf{CM.Commit}(\mathsf{ck}_2, m_2;\, \omega_2) \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{CM.Setup}^{\rho}(1^{\lambda}, L) \\ ((\ell_1, m_1, \omega_1), (\ell_2, m_2, \omega_2)) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}) \\ \mathsf{ck}_1 \leftarrow \mathsf{CM.Trim}^{\rho}(\mathsf{pp}, \ell_1) \\ \mathsf{ck}_2 \leftarrow \mathsf{CM.Trim}^{\rho}(\mathsf{pp}, \ell_2) \end{array}\right] \leq \mathrm{negl}(\lambda)\ .$$

Note that $m_1 \neq m_2$ is well-defined since $\mathcal{M}_{\mathsf{ck}_1}, \mathcal{M}_{\mathsf{ck}_2} \subseteq \mathcal{M}_{\mathsf{pp}}$.

3.6 Polynomial commitments

A polynomial commitment scheme is a cryptographic primitive that enables a sender to commit to a polynomial p over a field $\mathbb{F}$ and then later prove the correct evaluation of the polynomial at a desired point. In more detail, a polynomial commitment scheme PC is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ with the following syntax and properties:

- $\mathsf{PC.Setup}(1^{\lambda}, D) \to \mathsf{pp}$. On input a security parameter λ (in unary), and a maximum degree bound $D \in \mathbb{N}$, $\mathsf{PC.Setup}$ samples public parameters $\mathsf{pp}_{\mathsf{PC}}$. The parameters contain the description of a finite field $\mathbb{F}$ (which has size that is super-polynomial in λ).

- $\mathsf{PC.Trim}(\mathsf{pp}, d) \to (\mathsf{ck}, \mathsf{rk})$. On input public parameters $\mathsf{pp}_{\mathsf{PC}}$, and supported degree d, $\mathsf{PC.Trim}$ deterministically computes a key pair $(\mathsf{ck}, \mathsf{rk})$ that is specialized to d.

- $\mathsf{PC.Commit}(\mathsf{ck}, p; \omega) \to C$. On input ck, a univariate polynomial p over the field $\mathbb{F}$ such that $\deg(p) \leq \mathsf{ck}.d$, $\mathsf{PC.Commit}$ outputs a commitment C to the polynomial p. The randomness ω is used if the commitment C is hiding.

- $\mathsf{PC.Open}(\mathsf{ck}, p, C, z; \omega) \to \pi$. On input the commitment key ck, a univariate polynomial p over the field $\mathbb{F}$, a commitment C to p, an evaluation point z, and commitment randomness ω, $\mathsf{PC.Open}$ outputs an evaluation proof π.

- $\mathsf{PC.Check}(\mathsf{rk}, C, z, v, \pi) \to b$. On input the receiver key rk, a commitment C, an evaluation point z, a claimed evaluation v, and an evaluation proof π, $\mathsf{PC.Check}$ if π attests that the polynomial p committed in C has degree at most $\mathsf{rk}.d$, and evaluates to v at z.

A polynomial commitment scheme PC must be such that $(\mathsf{PC.Setup}, \mathsf{PC.Trim}, \mathsf{PC.Commit})$ is a (binding) commitment scheme for bounded-degree polynomials over a field. The message format L is equal to the maximum degree bound D; the message universe is the set of polynomials over some field $\mathbb{F}$ of degree at most D. The trim specification ℓ is equal to the supported degree d; the corresponding message space is the set of polynomials over $\mathbb{F}$ of degree at most d.

In this work we will not use any property of the $\mathsf{PC.Open}$ and $\mathsf{PC.Check}$ algorithms; their inclusion here is merely for consistency with previous definitions.

4 Split accumulation schemes for relations

Let $\Phi\colon (\{0,1\}^*)^3 \to \{0,1\}$ be a relation predicate. Let $\mathcal{H}$ be a randomized oracle algorithm, which outputs predicate parameters pp_Φ.

An **accumulation scheme for** $(\Phi, \mathcal{H})$ is a tuple of algorithms $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ of which $\mathrm{I}, \mathrm{P}, \mathrm{V}$ have access to the same random oracle ρ. The algorithms have the following syntax and properties.

Syntax. The algorithms comprising AS have the following syntax:

- *Generator:* On input a security parameter λ (in unary), G samples and outputs public parameters pp.
- *Indexer:* On input public parameters pp, predicate parameters pp_Φ (generated by $\mathcal{H}$), and a predicate index i_Φ, I deterministically computes and outputs a triple $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk})$ consisting of an accumulator proving key apk, an accumulator verification key avk, and a decision key dk.[5]
- *Accumulation prover:* On input the accumulator proving key apk, inputs $[\mathsf{q}_i = (\mathsf{q}.\mathbb{x}_i, \mathsf{q}.\mathbb{w}_i)]_{i=1}^n$, and old accumulators $[\mathsf{acc}_j = (\mathsf{acc}.\mathbb{x}_j, \mathsf{acc}.\mathbb{w}_j)]_{j=1}^m$, P outputs a new accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ and a proof π_V for the accumulation verifier.
- *Accumulation verifier:* On input the accumulator verification key avk, input instances $[\mathsf{q}_i.\mathbb{x}]_{i=1}^n$, accumulator instances $[\mathsf{acc}_j.\mathbb{x}]_{j=1}^m$, a new accumulator instance $\mathsf{acc}.\mathbb{x}$, and a proof π_V, V outputs a bit indicating whether $\mathsf{acc}.\mathbb{x}$ correctly accumulates $[\mathsf{q}_i]_{i=1}^n$ and $[\mathsf{acc}_j]_{j=1}^m$.
- *Decider:* On input the decision key dk, and an accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$, D outputs a bit indicating whether acc is a valid accumulator.

These algorithms must satisfy two properties, *completeness* and *knowledge soundness*, defined below. We additionally define a notion of zero knowledge that we will rely on to achieve zero knowledge PCD (see Section 5).

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

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

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

Knowledge soundness. We say that $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ has *knowledge error* $k(\lambda)$ if for every (non-uniform) adversary $\tilde{\mathcal{P}}$ running in expected polynomial time there exists an extractor E running in expected

[5] We remark that in some schemes it is important, for the sake of efficiency, for the indexer I to have oracle access to the predicate parameters pp_Φ and predicate index i_Φ, rather than reading them in full. All of our constructions and statements extend, in a straightforward way, to this case.

polynomial time such that for every set Z:

$$\Pr\left[\begin{array}{c}(\mathsf{pp},\mathsf{pp}_\Phi,\mathsf{ai},\mathsf{i}_\Phi,\mathsf{acc},[\mathsf{q}_i.\mathbb{x}]_{i=1}^n,[\mathsf{acc}_j.\mathbb{x}]_{j=1}^m,\mathsf{ao})\in Z\\ \wedge\\ \forall\, j\in[m],\ \mathrm{D}(\mathsf{dk},\mathsf{acc}_j)=1\\ \forall\, i\in[n],\ \Phi(\mathsf{pp}_\Phi,\mathsf{i}_\Phi,\mathsf{q}_i)=1\end{array}\middle|\begin{array}{r}\rho\leftarrow\mathcal{U}(\lambda)\\ \mathsf{pp}\leftarrow\mathrm{G}(1^\lambda)\\ \mathsf{pp}_\Phi\leftarrow\mathcal{H}(1^\lambda)\\ \mathsf{ai}\leftarrow\mathcal{D}(1^\lambda)\\ \begin{pmatrix}\mathsf{i}_\Phi & \mathsf{acc} & \mathsf{ao}\\ [\mathsf{q}_i]_{i=1}^n & [\mathsf{acc}_j]_{j=1}^m\end{pmatrix}\leftarrow\mathrm{E}_{\tilde{\mathcal{P}}}(\mathsf{pp},\mathsf{pp}_\Phi,\mathsf{ai})\end{array}\right]$$

$$\geq\Pr\left[\begin{array}{c}(\mathsf{pp},\mathsf{pp}_\Phi,\mathsf{ai},\mathsf{i}_\Phi,\mathsf{acc},[\mathsf{q}_i.\mathbb{x}]_{i=1}^n,[\mathsf{acc}_j.\mathbb{x}]_{j=1}^m,\mathsf{ao})\in Z\\ \wedge\\ \mathrm{V}^\rho(\mathsf{avk},[\mathsf{q}_i.\mathbb{x}]_{i=1}^n,[\mathsf{acc}_j.\mathbb{x}]_{j=1}^m,\mathsf{acc}.\mathbb{x},\pi_{\mathrm{V}})=1\\ \mathrm{D}(\mathsf{dk},\mathsf{acc})=1\end{array}\middle|\begin{array}{r}\rho\leftarrow\mathcal{U}(\lambda)\\ \mathsf{pp}\leftarrow\mathrm{G}(1^\lambda)\\ \mathsf{pp}_\Phi\leftarrow\mathcal{H}^\rho(1^\lambda)\\ \mathsf{ai}\leftarrow\mathcal{D}(1^\lambda)\\ \begin{pmatrix}\mathsf{i}_\Phi & \mathsf{acc} & \pi_{\mathrm{V}} & \mathsf{ao}\\ [\mathsf{q}_i.\mathbb{x}]_{i=1}^n & & [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m\end{pmatrix}\leftarrow\tilde{\mathcal{P}}^\rho(\mathsf{pp},\mathsf{pp}_\Phi,\mathsf{ai})\\ (\mathsf{apk},\mathsf{avk},\mathsf{dk})\leftarrow\mathrm{I}^\rho(\mathsf{pp},\mathsf{pp}_\Phi,\mathsf{i}_\Phi)\end{array}\right]-k(\lambda)\ .$$

If $k(\lambda)$ is negligible then we say AS satisfies knowledge soundness.

Zero knowledge. There exists a polynomial-time simulator S such that for every polynomial-size "honest" adversary $\mathcal{A}$ (see below) the following distributions are (statistically/computationally) indistinguishable:

$$\left\{(\rho,\mathsf{pp},\mathsf{pp}_\Phi,\mathsf{i}_\Phi,\mathsf{acc})\middle|\begin{array}{r}\rho\leftarrow\mathcal{U}(\lambda)\\ \mathsf{pp}\leftarrow\mathrm{G}(1^\lambda)\\ \mathsf{pp}_\Phi\leftarrow\mathcal{H}(1^\lambda)\\ (\mathsf{i}_\Phi,[\mathsf{q}_i]_{i=1}^n,[\mathsf{acc}_j]_{j=1}^m)\leftarrow\mathcal{A}^\rho(\mathsf{pp},\mathsf{pp}_\Phi)\\ (\mathsf{apk},\mathsf{avk},\mathsf{dk})\leftarrow\mathrm{I}^\rho(\mathsf{pp},\mathsf{pp}_\Phi,\mathsf{i}_\Phi)\\ (\mathsf{acc},\pi_{\mathrm{V}})\leftarrow\mathrm{P}^\rho(\mathsf{apk},[\mathsf{q}_i]_{i=1}^n,[\mathsf{acc}_j]_{j=1}^m)\end{array}\right\}$$

and

$$\left\{(\rho[\mu],\mathsf{pp},\mathsf{pp}_\Phi,\mathsf{i}_\Phi,\mathsf{acc})\middle|\begin{array}{r}\rho\leftarrow\mathcal{U}(\lambda)\\ (\mathsf{pp},\tau)\leftarrow\mathrm{S}^\rho(1^\lambda)\\ \mathsf{pp}_\Phi\leftarrow\mathcal{H}(1^\lambda)\\ (\mathsf{i}_\Phi,[\mathsf{q}_i]_{i=1}^n,[\mathsf{acc}_j]_{j=1}^m)\leftarrow\mathcal{A}^\rho(\mathsf{pp},\mathsf{pp}_\Phi)\\ (\mathsf{acc},\mu)\leftarrow\mathrm{S}^\rho(\mathsf{pp}_\Phi,\tau,\mathsf{i}_\Phi)\end{array}\right\}\ .$$

Here $\mathcal{A}$ is *honest* if it outputs, with probability 1, a tuple $(\mathsf{i}_\Phi,[\mathsf{q}_i]_{i=1}^n,[\mathsf{acc}_j]_{j=1}^m)$ such that $\Phi(\mathsf{pp}_\Phi,\mathsf{i}_\Phi,\mathsf{q}_i)=1$ and $\mathrm{D}(\mathsf{dk},\mathsf{acc}_j)=1$ for all $i\in[n]$ and $j\in[m]$. Note that the simulator S is *not* required to simulate the accumulation verifier proof π_{V}.

4.1 Accumulation schemes for certain predicates

We conclude by specializing the definition of an accumulation scheme to the case of predicates induced by the verifier in a non-interactive argument (Definition 4.1) and in a polynomial commitment scheme (Definition 4.2).

Definition 4.1 (accumulation for ARG)**.** *A non-interactive argument system $\mathsf{ARG}=(\mathcal{G},\mathcal{I},\mathcal{P},\mathcal{V})$ has an accumulation scheme if the pair $(\Phi_{\mathcal{V}},\mathcal{H}_{\mathsf{ARG}}:=\mathcal{G})$ has an accumulation scheme, where $\Phi_{\mathcal{V}}$ is defined below:*

$\Phi_{\mathcal{V}}(\mathsf{pp}_\Phi=\mathsf{pp},\mathsf{i}_\Phi=\mathbb{i},\mathsf{q}.\mathbb{x}=(\mathbb{x},\pi.\mathbb{x}),\mathsf{q}.\mathbb{w}=\pi.\mathbb{w})$*:*

1. $(\mathrm{ipk},\mathrm{ivk})\leftarrow\mathcal{I}(\mathsf{pp},\mathbb{i})$.
2. *Output* $\mathcal{V}(\mathrm{ivk},\mathbb{x},(\pi.\mathbb{x},\pi.\mathbb{w}))$.

Definition 4.2 (accumulation for PC)**.** *A polynomial commitment scheme* $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ *has an accumulation scheme if, for every* $D(\lambda) = \mathrm{poly}(\lambda)$*, the pair* $(\Phi_{\mathsf{PC}}, \mathcal{H}_{\mathsf{PC},D})$ *defined below has an accumulation scheme.*

$\Phi_{\mathsf{PC}}(\mathsf{pp}_\Phi = \mathsf{pp}_{\mathsf{PC}}, \mathsf{i}_\Phi = d, q = ((C, z, v), \pi))$*:*
1. $(\mathsf{ck}, \mathsf{rk}) \leftarrow \mathsf{PC.Trim}(\mathsf{pp}_{\mathsf{PC}}, d)$.
2. *Output* $\mathsf{PC.Check}(\mathsf{rk}, C, z, v, \pi)$.

$\mathcal{H}_{\mathsf{PC},D}(1^\lambda)$*:*
Output $\mathsf{pp}_{\mathsf{PC}} \leftarrow \mathsf{PC.Setup}(1^\lambda, D(\lambda))$.

5 PCD from arguments of knowledge with split accumulation

We formally restate and then prove Theorem 1, which provides a construction of proof-carrying data (PCD) from any NARK that has a split accumulation scheme with certain efficiency properties.

First, we provide some notation for these properties.

Definition 5.1. *Let* $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ *be an accumulation scheme for a non-interactive argument (see Definition 4.1). We denote by* $\mathrm{V}^{(\lambda,m,N,k)}$ *the circuit corresponding to the computation of the accumulation verifier* V*, for security parameter* λ*, when checking the accumulation of* m *instance-proof pairs and accumulators, on an index of size at most* N*, where each instance is of size at most* k*.*

We denote by $\mathsf{v}(\lambda, m, N, k)$ *the size of the circuit* $\mathrm{V}^{(\lambda,m,N,k)}$*, by* $|\mathsf{avk}(\lambda, m, N)|$ *the size of the accumulator verification key* avk*, and by* $|\mathsf{acc}.\mathbb{x}(\lambda, m, N)|$ *the size of an accumulator instance.*

Note that here we have specified that the size of $\mathsf{acc}.\mathbb{x}$ is bounded by a function of λ, m, N; in particular, it *may not* depend on the number of instances accumulated.

When we invoke the accumulation verifier in our construction of PCD, an instance will consist of an accumulator verification key, an accumulator *instance*, and some additional data of size ℓ. Thus the size of the accumulation verifier circuit used in the scheme is given by

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

The notion of "sublinear verification" which is important here is that v^* is sublinear in N. The following theorem shows that when this is the case, this accumulation scheme can be used to construct PCD.

Theorem 5.2. *There exists a polynomial-time transformation* T *such that if* $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *is a NARK for circuit satisfiability and* AS *is a split accumulation scheme for* ARG *then* $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V}) := \mathrm{T}(\mathsf{ARG}, \mathsf{AS})$ *is a PCD scheme for constant-depth compliance predicates, provided*

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

Moreover:

- *If* ARG *and* AS *are secure against quantum adversaries, then* PCD *is secure against quantum adversaries.*
- *If* ARG *and* AS *are (post-quantum) zero knowledge, then* PCD *is (post-quantum) zero knowledge.*
- *If the size of the predicate* $\varphi \colon \mathbb{F}^{(m+2)\ell} \to \mathbb{F}$ *is* $f = \omega(\alpha(\lambda, m, \ell)^{1/\epsilon})$ *then:*
  - *the cost of running* $\mathbb{I}$ *is equal to the cost of running both* $\mathcal{I}$ *and* I *on an index of size* $f + o(f)$*;*
  - *the cost of running* $\mathbb{P}$ *is equal to the cost of accumulating* m *instance-proof pairs using* P*, and running* $\mathcal{P}$*, on an index of size* $f + o(f)$ *and instance of size* $o(f)$*;*
  - *the cost of running* $\mathbb{V}$ *is equal to the cost of running both* $\mathcal{V}$ *and* D *on an index of size* $f + o(f)$ *and an instance of size* $o(f)$*.*

This last point gives the conditions for a *sublinear additive* recursive overhead; i.e., when the *additional* cost of proving that φ is satisfied recursively is asymptotically smaller than the cost of proving that φ is satisfied locally. Note that the smaller the compliance predicate φ, the more efficient the accumulation scheme has to be in order to achieve this.

5.1 Construction

Let $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ be a non-interactive argument for circuit satisfiability, and let $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ be an accumulation scheme for ARG. Below we construct a PCD scheme $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$.

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

$R_{\mathrm{V},\varphi}^{(\lambda,N,k)}\big((\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), (z_{\mathsf{loc}}, [z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \pi_{\mathrm{V}})\big)$:

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

$$\mathrm{V}^{(\lambda,m,N,k)}(\mathsf{avk}, [(\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i.\mathbb{x}]_{i=1}^m, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \mathsf{acc}.\mathbb{x}, \pi_{\mathrm{V}}) = 1 \ .$$

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

Below we denote by "q_i" the query contents $((\mathsf{avk}, z_i, \mathsf{acc}_i), \pi_i)$, and use "$\mathsf{q}_i.\mathbb{x}$" to denote query instances $((\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i.\mathbb{x})$ and "$\mathsf{q}_i.\mathbb{w}$" to denote the corresponding query witnesses $((\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{w}), \pi_i.\mathbb{w})$.

Above, $\mathrm{V}^{(\lambda,m,N,k)}$ refers to the circuit representation of V with input size appropriate for security parameter λ, number of instance-proof pairs and accumulators m, index size N, and instance size k.

Next we describe the generator $\mathbb{G}$, indexer $\mathbb{I}$, prover $\mathbb{P}$, and verifier $\mathbb{V}$ of the PCD scheme.

- $\mathbb{G}(1^\lambda)$: Sample $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$ and $\mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{G}(1^\lambda)$, and output $\mathbb{pp} := (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}})$.
- $\mathbb{I}(\mathbb{pp}, \varphi)$:
  1. Compute the integer $N := N(\lambda, |\varphi|, m, \ell)$, where N is defined in Lemma 5.3 below.
  2. Construct the circuit $R := R_{\mathrm{V},\varphi}^{(\lambda,N,k)}$ where $k := |\mathsf{avk}(\lambda, N)| + \ell + |\mathsf{acc}(\lambda, N)|$.
  3. Compute the index key pair $(\mathrm{ipk}, \mathrm{ivk}) \leftarrow \mathcal{I}(\mathsf{pp}, R)$ for the circuit R for the NARK.
  4. Compute the index key triple $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{i}_\Phi = (\mathsf{pp}, R))$ for the accumulator.
  5. Output $\mathsf{ipk} := (\mathrm{ipk}, \mathsf{apk})$ and $\mathsf{ivk} := (\mathrm{ivk}, \mathsf{dk}, \mathsf{avk})$.
- $\mathbb{P}(\mathsf{ipk}, z, z_{\mathsf{loc}}, [z_i, (\pi_i, \mathsf{acc}_i)]_{i=1}^m)$:
  1. If $z_i = \bot$ for all $i \in [m]$ then set $(\mathsf{acc}, \pi_{\mathrm{V}}) \leftarrow \mathrm{P}(\mathsf{apk}, \bot)$.
  2. If $z_i \neq \bot$ for some $i \in [m]$ then compute $(\mathsf{acc}, \pi_{\mathrm{V}}) \leftarrow \mathrm{P}(\mathsf{apk}, [\mathsf{q}_i]_{i=1}^m, [\mathsf{acc}_i]_{i=1}^m)$.
  3. Compute $\pi \leftarrow \mathcal{P}\big(\mathrm{ipk}, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), (z_{\mathsf{loc}}, [z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \pi_{\mathrm{V}})\big)$.
  4. Output (π, acc).
- $\mathbb{V}(\mathsf{ivk}, z, (\pi, \mathsf{acc}))$: Accept if both $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), \pi)$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ accept.

5.2 Completeness

Let $\mathcal{A}$ be any adversary that causes the completeness condition of PCD to be satisfied with probability p. We construct an adversary $\mathcal{B}$, as follows, that causes the completeness condition of AS to be satisfied with probability at most p.

$\mathcal{B}(\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}})$:

1. Set $\mathbb{pp} := (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}})$ and compute $(\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i, \mathsf{acc}_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathbb{pp})$.
2. Run $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)})$.
3. Output $(R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, [\mathsf{q}_i]_{i=1}^m, [\mathsf{acc}_i]_{i=1}^m)$.

Suppose that $\mathcal{A}$ outputs $(\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i, \mathsf{acc}_i]_{i=1}^m)$ such that the completeness precondition is satisfied, but $\mathbb{V}(\mathsf{ivk}, z, (\pi, \mathsf{acc})) = 0$. Then, by construction of $\mathbb{V}$, it holds that either $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), \pi) = 0$ or $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 0$. If $z_i = \perp$ for all i, then by perfect completeness of ARG both of these algorithms output 1; hence there exists i such that $z_i \neq \perp$. Hence it holds that for all i, $\mathbb{V}(\mathsf{ivk}, z_i, (\pi_i, \mathsf{acc}_i)) = 1$, whence for all i, $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i) = \Phi_{\mathcal{V}}(\mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, (\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i) = 1$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}_i) = 1$.

If $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), \pi) = 0$, then, by perfect completeness of ARG, we know that $R_{\mathrm{V},\varphi}^{(\lambda,N,k)}$ rejects $\big((\mathsf{avk}, z, \mathsf{acc}), (z_{\mathsf{loc}}, [z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}]_{i=1}^m), \pi_{\mathrm{V}}\big)$, and so $\mathrm{V}(\mathsf{avk}, [\mathsf{q}_i.\mathbb{x}]_{i=1}^m, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \mathsf{acc}.\mathbb{x}) = 0$. Otherwise, $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 0$.

Now consider the completeness experiment for AS with adversary $\mathcal{B}$. Since $\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}$ are drawn identically to the PCD experiment, the distribution of the output of $\mathcal{A}$ is identical. Hence in particular it holds that for all i, $\Phi_{\mathcal{V}}(\mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, (\mathsf{avk}, z_i, \mathsf{acc}_i), \pi_i) = 1$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}_i) = 1$. By the above, it holds that either $\mathrm{V}(\mathsf{avk}, [\mathsf{q}_i.\mathbb{x}]_{i=1}^m, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \mathsf{acc}) = 0$ or $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 0$, and so $\mathcal{B} := (\mathcal{B}_1, \mathcal{B}_2)$ causes the completeness condition for AS to be satisfied with probability at most p.

5.3 Knowledge soundness

Since the extracted transcript T will be a tree, we find it convenient to associate the label $z^{(u,v)}$ of the unique outgoing edge of a node u with the node u itself, so that the node u is labelled with $(z^{(u)}, z_{\mathsf{loc}}^{(u)})$. For the purposes of the proof we also associate with each node u a NARK proof $\pi^{(u)}$ and an accumulator $\mathsf{acc}^{(u)}$, so that the full label for a node is $(z^{(u)}, z_{\mathsf{loc}}^{(u)}, \pi^{(u)}, \mathsf{acc}^{(u)})$. It is straightforward to transform such a transcript into one that satisfies Definition 3.1.

Given a malicious prover $\tilde{\mathbb{P}}$, we will define an extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ that satisfies knowledge soundness. In the process we construct a sequence of extractors $\mathbb{E}_1, \ldots, \mathbb{E}_d$ for $d := d(\varphi)$ (the depth of φ); $\mathbb{E}_j$ outputs a tree of depth $j+1$. Let $\mathbb{E}_0(\mathbb{pp}, \mathsf{ai})$ run $(\varphi, \mathsf{o}, \pi, \mathsf{acc}) \leftarrow \tilde{\mathbb{P}}(\mathbb{pp}, \mathsf{ai})$ and output (φ, T_0), where T_0 is a single node labeled with $(\mathsf{o}, \pi, \mathsf{acc})$. Let $l_{\mathsf{T}}(j)$ denote the vertices of T at depth j; $l_{\mathsf{T}}(0) := \varnothing$ and $l_{\mathsf{T}}(1)$ is the singleton containing the root.

Now we define the extractor $\mathbb{E}_j$ inductively for each $j \in [d]$. Suppose we have already constructed $\mathbb{E}_{j-1}$. We construct a NARK prover $\tilde{\mathcal{P}}_j$ as follows:

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

1. Compute $(\varphi, \mathsf{T}_{j-1}, \mathsf{ao}) \leftarrow \mathbb{E}_{j-1}((\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}), \mathsf{ai})$.
2. For each vertex $v \in l_{\mathsf{T}_{j-1}}(j)$, denote its label by $(z^{(v)}, \pi^{(v)}, \mathsf{acc}^{(v)})$.
3. Run the argument indexer $(\mathrm{ipk}, \mathrm{ivk}) \leftarrow \mathcal{I}(\mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)})$. Run the accumulator indexer $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)})$.
4. Output

$$(\vec{\mathrm{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{ao}') := \Big(\vec{R}, (\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x})_{v \in l_{\mathsf{T}_{j-1}}(j)}, (\pi^{(v)})_{v \in l_{\mathsf{T}_{j-1}}(j)}, (\varphi, \mathsf{T}_{j-1}, \mathsf{ao})\Big)$$

where $\vec{R}$ is the vector $(R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, \ldots, R_{\mathrm{V},\varphi}^{(\lambda,N,k)})$ of the appropriate length.

Next let $\mathcal{E}_{\tilde{\mathcal{P}}_j}$ be the extractor that corresponds to $\tilde{\mathcal{P}}_j$, via the knowledge soundness of the non-interactive argument ARG. We now define an accumulation scheme prover P_j as follows:

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

1. Run the extractor $(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\mathbb{w}}, \mathsf{ao}') \leftarrow \mathcal{E}_{\tilde{\mathcal{P}}_j}(\mathsf{pp}, (\mathsf{pp}_{\mathsf{AS}}, \mathsf{ai}))$.
2. Parse the auxiliary output ao' as $(\varphi, \mathsf{T}', \mathsf{ao})$. If T' is not a transcript of depth j, abort.
3. For each vertex $v \in l_{\mathsf{T}'}(j)$,
   - obtain $\mathsf{acc}^{(v)}$ from T';
   - obtain the local data $z^{(v)}_{\mathsf{loc}}$, input messages $\left(z_i^{(v)}, \pi_i^{(v)}.\mathbb{x}, \mathsf{acc}_i^{(v)}.\mathbb{x}\right)_{i\in[m]}$ and accumulation proof $\pi_{\mathrm{V}}^{(v)}$ from $\mathbb{w}^{(v)}$;
   - append $z^{(v)}_{\mathsf{loc}}$ to the label of v in T';
   - let $S_j := \{v \in l_{\mathsf{T}'}(j) : \exists i,\ z_i^{(v)} \neq \perp\}$; attach m children to each $v \in S_j$, where the i-th child is labeled with $z_i^{(v)}$.
4. Output $\left(\left(\mathbb{i}^{(v)}, \mathsf{acc}^{(v)}, \pi_{\mathrm{V}}^{(v)}, [\mathsf{q}_i^{(v)}.\mathbb{x} = \left((\mathsf{avk}, z_i^{(v)}, \mathsf{acc}_i^{(v)}.\mathbb{x}), \pi_i^{(v)}.\mathbb{x}\right)]_{i=1}^m, [\mathsf{acc}_i^{(v)}.\mathbb{x}]_{i=1}^m\right)_{v\in S_j}, (\varphi, \mathsf{T}', \mathsf{ao})\right)$.

Let $\mathrm{E}_{\tilde{\mathrm{P}}_j}$ be the extractor corresponding to $\tilde{\mathrm{P}}_j$, by the knowledge soundness guarantee of AS. Finally, we define the extractor $\mathbb{E}_j$ as follows:

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

1. Run the extractor $\left(\left(\mathbb{i}^{(v)}, \mathsf{acc}^{(v)}, [\mathsf{q}_i^{(v)}]_{i=1}^m, [\mathsf{acc}_i^{(v)}]_{i=1}^m\right)_{v\in S_j}, \mathsf{ao}'\right) \leftarrow \mathrm{E}_{\tilde{\mathrm{P}}_j}(\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}, \mathsf{ai})$.
2. Parse the auxiliary output ao' as $(\varphi, \mathsf{T}', \mathsf{ao})$. If T' is not a transcript of depth j, abort. Let $S_j := \{v \in l_{\mathsf{T}'}(j) : \exists i,\ z_i^{(v)} \neq \perp\}$.
3. Parse each $\mathsf{q}_i^{(v)}$ as $(\mathsf{avk}^{(v)}, z_i^{(v)}, \mathsf{acc}_i^{(v)}.\mathbb{x}, \pi_i^{(v)})$.
4. Output $(\varphi, \mathsf{T}_j, \mathsf{ao})$ where T_j is the transcript constructed from T' by adding, for each vertex $v \in S_j$, $(\pi_i^{(v)}, \mathsf{acc}_i^{(v)})$ to the label of its i-th child.

The extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ is equal to $\mathbb{E}_d$.

We now show that $\mathbb{E}_{\tilde{\mathbb{P}}}$ runs in expected polynomial time and that it outputs a transcript that is φ-compliant.

Running time of the extractor. It follows from the extraction guarantees of ARG and AS that $\mathbb{E}_j$ runs in expected time polynomial in the expected running time of $\mathbb{E}_{j-1}$. Hence if $d(\varphi)$ is a constant, $\mathbb{E}_{\tilde{\mathbb{P}}} = \mathbb{E}_{d(\varphi)}$ runs in expected polynomial time.

Correctness of the extractor. Fix a set Z, and suppose that $\tilde{\mathbb{P}}$'s output falls in Z, and causes $\mathbb{V}$ to accept, with probability μ. We show by induction that, for all $j \in \{0, \ldots, d\}$, the transcript T_j output by $\mathbb{E}_j$ is φ-compliant up to depth j, and that for all $v \in \mathsf{T}_j$, both $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x}), \pi^{(v)})$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}^{(v)})$ accept, and that $(\mathbb{pp}, \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}_j), \mathsf{ao}) \in Z$ and $\varphi \in \mathsf{F}$, with probability $\mu - \mathrm{negl}(\lambda)$.

For $j = 0$ the statement holds by assumption.

Now suppose that $(\varphi, \mathsf{T}_{j-1}) \leftarrow \mathbb{E}_{j-1}(\mathbb{pp}, \mathsf{ai})$ is such that T_{j-1} is φ-compliant up to depth $j-1$, and that both $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x}), \pi^{(v)})$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}^{(v)})$ accept for all $v \in \mathsf{T}_{j-1}$, with probability $\mu - \mathrm{negl}(\lambda)$.

Let $(\vec{\mathbb{i}}, (\mathsf{avk}_v, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x})_v, (\pi^{(v)})_v, (\varphi, \mathsf{T}'), \vec{\mathbb{w}}) \leftarrow \mathcal{E}_{\tilde{\mathcal{P}}_j}(\mathsf{pp}, (\mathsf{pp}_{\mathsf{AS}}, \mathsf{ai}))$.

We let $(\mathsf{pp}, (\mathsf{pp}_{\mathsf{AS}}, \mathsf{ai}), \vec{\mathbb{i}}, (\mathsf{avk}_v, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x})_v, (\varphi, \mathsf{T}', \mathsf{ao})) \in Z'$ if and only if, for $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, R^{(\lambda,N,k)}_{\mathrm{V},\varphi})$:

- $((\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}), \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}'), \mathsf{ao}) \in Z$ and $\varphi \in \mathsf{F}$,
- $\mathbb{i}^{(v)} = R^{(\lambda,N,k)}_{\mathrm{V},\varphi}$ and $\mathsf{avk}_v = \mathsf{avk}$ for all v,
- T' is φ-compliant up to depth $j-1$,
- $\mathrm{D}(\mathsf{dk}, \mathsf{acc}^{(v)})$ accepts for all $v \in \mathsf{T}'$, and

- for $v \in l_{\mathsf{T}'}(j)$, v is labeled in T' with $(z^{(v)}, \pi^{(v)}, \mathsf{acc}^{(v)})$.

By knowledge soundness, with probability $\mu - \mathrm{negl}(\lambda)$, $(\mathsf{pp}, (\mathsf{pp}_{\mathsf{AS}}, \mathsf{ai}), \vec{\mathbb{i}}, (\mathrm{ivk}_v, z^{(v)})_v, (\varphi, \mathsf{T}')) \in Z'$ and for every vertex $v \in l_{\mathsf{T}'}(j)$, $(R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, (\mathsf{avk}_v, z^{(v)}, \mathsf{acc}^{(v)}), \mathbb{w}^{(v)}) \in \mathcal{R}_{\mathrm{R1CS}}$. Here we use Z' and the auxiliary output in the knowledge soundness definition of ARG to ensure consistency between the values $z^{(v)}$ and T', and to ensure that T' is φ-compliant and that the decider accepts.

Consider some $v \in l_{\mathsf{T}'}(j)$. Since $(R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, (\mathsf{avk}^{(v)}, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x}), \mathbb{w}^{(v)}) \in \mathcal{R}_{\mathrm{R1CS}}$, we obtain from $\mathbb{w}^{(v)}$ either

- local data $z_{\mathsf{loc}}^{(v)}$, input messages $\left(z_i^{(v)}, \pi_i^{(v)}.\mathbb{x}, \mathsf{acc}_i^{(v)}.\mathbb{x}\right)_{i\in[m]}$ and proof π_{V} such that $\varphi(z^{(v)}, z_{\mathsf{loc}}, z_1, \ldots, z_m)$ accepts and the accumulation verifier $\mathrm{V}^{(\lambda,N,k)}(\mathsf{avk}^{(v)}, [\mathsf{q}_i^{(v)}.\mathbb{x}]_{i=1}^m, [\mathsf{acc}_i^{(v)}.\mathbb{x}]_{i=1}^m, \mathsf{acc}^{(v)}, \pi_{\mathrm{V}}^{(v)})$ accepts, where $\mathsf{q}_i^{(v)}.\mathbb{x} := ((\mathsf{avk}^{(v)}, z_i^{(v)}, \mathsf{acc}_i^{(v)}.\mathbb{x}), \pi_i^{(v)}.\mathbb{x})$; or
- local data $z_{\mathsf{loc}}^{(v)}$ such that $\varphi(z^{(v)}, z_{\mathsf{loc}}^{(v)}, \bot, \ldots, \bot)$ accepts.

In both cases we append $z_{\mathsf{loc}}^{(v)}$ to the label of v. In the latter case, v has no children and so is φ-compliant by the base case condition. In the former case we label the children of v with $(z_i, \pi_i, \mathsf{acc}_i)$, and so v is φ-compliant.

We define $(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}, (\mathbb{i}^{(v)}, \mathsf{acc}^{(v)}, [\mathsf{q}_i^{(v)}.\mathbb{x}]_{i=1}^m, [\mathsf{acc}_i^{(v)}.\mathbb{x}]_{i=1}^m)_v, (\varphi, \mathsf{T}', \mathsf{ao})) \in Z''$ if and only if

- $((\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}), \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}'), \mathsf{ao}) \in Z$ and $\varphi \in \mathsf{F}$,
- $\mathbb{i}^{(v)} = R_{\mathrm{V},\varphi}^{(\lambda,N,k)}$ for all v,
- T' is φ-compliant up to depth j,
- for all v, $\mathsf{q}_i^{(v)}.\mathbb{x} = ((\mathsf{avk}, z_i^{(v)}, \mathsf{acc}_i^{(v)}.\mathbb{x}), \pi_i^{(v)})$ where $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi)$, and
- for $u \in l_{\mathsf{T}'}(j+1)$, where u is the i-th child of $v \in l_{\mathsf{T}'}(j)$, u is labeled in T' with $z_i^{(v)}$.

Let $\left(\left(\mathbb{i}^{(v)}, \mathsf{acc}^{(v)}, [\mathsf{q}_i^{(v)}]_{i=1}^m, [\mathsf{acc}_i^{(v)}]_{i=1}^m\right)_{v\in S_j}, \mathsf{ao}'\right) \leftarrow \mathrm{E}_{\tilde{\mathrm{P}}_j}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai})$. By the knowledge soundness guarantee of the accumulation scheme, $(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}, (\mathbb{i}^{(v)}, \mathsf{acc}^{(v)}, [\mathsf{q}_i^{(v)}.\mathbb{x}]_{i=1}^m, [\mathsf{acc}_i^{(v)}.\mathbb{x}]_{i=1}^m)_v, \mathsf{ao}') \in Z''$, and it holds that for all descendants u of v in T_j, $\mathrm{D}(\mathsf{dk}, \mathsf{acc}^{(u)})$ accepts and $\Phi_{\mathcal{V}}(\mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, (\mathsf{avk}, z^{(u)}, \mathsf{acc}^{(u)}.\mathbb{x}), \pi_{\mathsf{in}}^{(u)}) = \mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z^{(u)}, \mathsf{acc}^{(u)}.\mathbb{x}), \pi_{\mathsf{in}}^{(u)})$ accepts, with probability $\mu - \mathrm{negl}(\lambda)$; this completes the inductive step.

Hence by induction, $(\varphi, \mathsf{T}, \mathsf{ao}) \leftarrow \mathbb{E}(\mathbb{pp}, \mathsf{ai})$ has φ-compliant T, $(\mathbb{pp}, \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}), \mathsf{ao}) \in Z$, and $\varphi \in \mathsf{F}$, with probability $\mu - \mathrm{negl}(\lambda)$.

5.4 Efficiency

The efficiency argument follows from Lemma 5.3 and is essentially identical to that of [BCMS20], and so we will not repeat it. We note only that the quantity v^* (i) describes the size of the *accumulation* verifier, which in particular need not read the entire NARK proof, which may be large, and (ii) is a function of the size of the accumulator *instance* alone; the accumulator witness may be large.

Lemma 5.3. *Suppose that for every security parameter $\lambda \in \mathbb{N}$, arity m, and message size $\ell \in \mathbb{N}$ the ratio of accumulation verifier circuit size to index size $\mathsf{v}^*(\lambda, m, N, \ell)/N$ is monotone decreasing in N. Then there exists a size function $N(\lambda, f, m, \ell)$ such that*

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

Moreover if for some $\epsilon > 0$ and some increasing function α it holds that, for all N, λ, m, ℓ sufficiently large,

$$\mathsf{v}^*(\lambda, m, N, \ell) \leq N^{1-\epsilon}\alpha(\lambda, m, \ell)$$

then, for all λ, m, ℓ sufficiently large, $N(\lambda, f, m, \ell) \leq O(f + \alpha(\lambda, m, \ell)^{1/\epsilon})$.

6 Accumulating Pedersen polynomial commitments

We construct a split accumulation scheme for the Pedersen polynomial commitment scheme. This polynomial commitment scheme has modest efficiency properties: commitments are succinct, but evaluation proofs (and the time to verify them) are linear in the degree of the committed polynomial.

Despite this, our split accumulation scheme achieves strong efficiency guarantees: checking the accumulation of n evaluation claims only requires the accumulation verifier V to add $O(n)$ commitments (*regardless* of the degree of the committed polynomials) and only requires the decider D to check one evaluation proof.

In more detail, we prove the following (more general) theorem for a polynomial commitment scheme $\mathsf{PC}_{\mathsf{HC}}$ derived from any $\mathbb{F}_q$-linear hash function HC. The Pedersen commitment [Ped92] is an $\mathbb{F}_q$-linear homomorphic commitment where the commitment space is a group $\mathbb{G}$ of order q, and is computationally binding provided the discrete logarithm problem is hard in $\mathbb{G}$.

Theorem 6.1. *The scheme* $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ *constructed in Section 6.1 is an accumulation scheme in the random oracle model for the polynomial commitment scheme* $\mathsf{PC}_{\mathsf{HC}}$ *described in Fig. 4.* AS *achieves the following efficiency:*

- Generator: $\mathrm{G}(1^\lambda)$ *runs in time* $O(\lambda)$.
- Indexer: *the time of* $\mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi = d)$ *is dominated by the time to compute* $\mathsf{HC.Trim}$ *with degree bound* d.
- Accumulation prover: *the time of* $\mathrm{P}^\rho(\mathsf{apk}, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m)$ *is dominated by the time to commit to* $n + m$ *polynomials of degree* d. *The output accumulator* acc *consists of*
  - *an accumulator instance* $\mathsf{acc}.\mathbb{x}$ *consisting of a commitment and two field elements, and*
  - *an accumulator witness* $\mathsf{acc}.\mathbb{w}$ *consisting of a polynomial of degree less than* d.

  The output proof π *consists of* n *commitments and* $2n + 2m$ *field elements.*
- Accumulation verifier: *the time of* $\mathrm{V}^\rho(\mathsf{avk}, [\mathsf{q}_i.\mathbb{x}]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, \mathsf{acc}.\mathbb{x}, \pi)$ *is dominated by* $O(n + m)$ *field additions/multiplications and* $O(n + m)$ *group scalar multiplications.*
- Decider: *the time of* $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ *is dominated by the time to compute a commitment to a polynomial of degree less than* d.

Definition 6.2. *An* $\mathbb{F}_q$*-linear* **homomorphic commitment** $\mathsf{HC} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit})$ *for prime* q *is a binding commitment scheme (Section 3.6) with the following additional properties: the message space is* $\mathbb{F}_q^n$*, and there exists an efficient operation* $+$ *on the commitment space such that*

$$\forall\, \vec{a}, \vec{b} \in \mathbb{F}_q^n\,,\ \mathsf{HC.Commit}(\vec{a}) + \mathsf{HC.Commit}(\vec{b}) = \mathsf{HC.Commit}(\vec{a} + \vec{b})\ .$$

$\mathsf{PC}_{\mathsf{HC}}$. We use HC to construct a homomorphic polynomial commitment scheme $\mathsf{PC}_{\mathsf{HC}}$:

- *Setup:* On input $\lambda, D \in \mathbb{N}$, output $\mathsf{pp}_{\mathsf{CM}} \leftarrow \mathsf{HC.Setup}(1^\lambda, D + 1)$.
- *Trim:* On input $\mathsf{pp}_{\mathsf{CM}}$ and $d \in \mathbb{N}$, check that $d \leq D$, set $\mathsf{ck}_{\mathsf{HC}} \leftarrow \mathsf{HC.Trim}(\mathsf{pp}_{\mathsf{CM}}, d + 1)$, and output $(\mathsf{ck} := (\mathsf{ck}_{\mathsf{HC}}, d), \mathsf{rk} := (\mathsf{ck}_{\mathsf{HC}}, d))$.
- *Commit:* On input ck and $p \in \mathbb{F}_q[X]$ of degree at most $\mathsf{ck}.d$, output $C \leftarrow \mathsf{HC.Commit}(\mathsf{ck}.\mathsf{ck}_{\mathsf{HC}}, p)$.
- *Open:* On input (ck, p, C, z), output $\pi := p$.
- *Check:* On input $(\mathsf{rk}, (C, z, v), \pi = p)$, check that $C = \mathsf{HC.Commit}(\mathsf{rk}.\mathsf{ck}_{\mathsf{HC}}, p)$, $p(z) = v$, and $\deg(p) \leq \mathsf{rk}.d$.

Completeness of $\mathsf{PC}_{\mathsf{HC}}$ follows from that of HC, while extractability follows from the binding property of HC.

Figure 4: $\mathsf{PC}_{\mathsf{HC}}$ is a homomorphic polynomial commitment scheme based on a linear homomorphic commitment.

6.1 Accumulation scheme for $\mathsf{PC}_{\mathsf{HC}}$

We describe the accumulation scheme $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ for the homomorphic polynomial commitment scheme $\mathsf{PC}_{\mathsf{HC}}$. First, we describe the form of inputs $\mathsf{q} = (\mathsf{q}.\mathbb{x}, \mathsf{q}.\mathbb{w})$ and accumulators $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$. Both $\mathsf{q}.\mathbb{x}$ and $\mathsf{acc}.\mathbb{x}$ are tuples of the form (C, z, v), while both $\mathsf{q}.\mathbb{w}$ and $\mathsf{acc}.\mathbb{w}$ consist of a polynomial p (allegedly, committed inside C and such that $p(z) = v$ and $\deg(p) < d$). Jumping ahead, we note that since the decider D is equal to the predicate Φ_{PC}, there is no distinction between inputs and prior accumulators, and so it suffices to accumulate inputs only.

Generator. The generator G receives as input $\mathsf{pp} := 1^\lambda$ and outputs 1^λ. (In other words, G does not have to create additional public parameters beyond those used by HC.)

Indexer. On input the accumulator parameters pp, predicate parameters $\mathsf{pp}_\Phi = \mathsf{pp}_{\mathsf{PC}}$, and a predicate index $\mathsf{i}_\Phi = d$, the indexer I computes the committer and receiver keys $(\mathsf{ck}, \mathsf{rk}) := \mathsf{PC}_{\mathsf{HC}}.\mathsf{Trim}(\mathsf{pp}_{\mathsf{PC}}, d)$, and then outputs the accumulator proving key $\mathsf{apk} := (\mathsf{ck}, d)$, the accumulator verification key $\mathsf{avk} := d$, and the decision key $\mathsf{dk} := \mathsf{rk}$.

Accumulation prover. On input accumulation proving key apk and inputs $[\mathsf{q}_i]_{i=1}^n$, P works as below.

$\mathrm{P}^\rho(\mathsf{apk} = (\mathsf{ck}, d), [\mathsf{q}_i]_{i=1}^n)$:
1. For each i in $[n]$:
   (a) Parse the input instance $\mathsf{q}.\mathbb{x}_i$ as (C_i, z_i, v_i) and the input witness $\mathsf{q}.\mathbb{w}_i$ as $\pi_i = p_i \in \mathbb{F}^{\leq \mathsf{ck}.d}[X]$.
   (b) Compute the witness polynomial $w_i(X) := \frac{p_i(X) - v_i}{X - z_i}$.
   (c) Compute a commitment to $w_i(X)$: $W_i := \mathsf{PC}_{\mathsf{HC}}.\mathsf{Commit}(\mathsf{ck}, w_i)$.
2. Compute the challenge $z := \rho(d, [(C_i, z_i, v_i, W_i)]_{i=1}^n)$.
3. For each i in $[n]$, compute the evaluations $y_i := p_i(z)$ and $y'_i := w_i(z)$.
4. Compute the challenge $\alpha := \rho(z, [(y_i, y'_i)]_{i=1}^n)$.
5. Compute the linear combination $p := \sum_{i=1}^n \alpha^{i-1} \cdot p_i + \sum_{i=1}^n \alpha^{n+i-1} \cdot w_i$.
6. Compute the evaluation $v := p(z)$.
7. Compute the linear combination $C := \sum_{i=1}^n \alpha^{i-1} \cdot C_i + \sum_{i=1}^n \alpha^{n+i-1} \cdot W_i$.
8. Set the accumulator $\mathsf{acc} := (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ where $\mathsf{acc}.\mathbb{x} := (C, z, v)$ and $\mathsf{acc}.\mathbb{w} := p$.
9. Set the proof $\pi_{\mathrm{V}} := [(W_i, y_i, y'_i)]_{i=1}^n$.
10. Output $(\mathsf{acc}, \pi_{\mathrm{V}})$.

Accumulation verifier. On input the accumulator verification key avk, input instances $[\mathsf{q}_i.\mathbb{x}]_{i=1}^n$, a new accumulator instance $\mathsf{acc}.\mathbb{x}$, and a proof π_{V}, V works as below.

$\mathrm{V}^\rho(\mathsf{avk} = d, [\mathsf{q}_i.\mathbb{x}]_{i=1}^n, \mathsf{acc}.\mathbb{x}, \pi_{\mathrm{V}})$:
1. Parse $\mathsf{acc}.\mathbb{x}$ as (C, z, v), and π_{V} as $[(W_i, y_i, y'_i)]_{i=1}^n$.
2. For each $i \in [n]$, parse $\mathsf{q}.\mathbb{x}_i$ as (C_i, z_i, v_i).
3. Check that $z = \rho(d, [(C_i, z_i, v_i, W_i)]_{i=1}^n)$.
4. For each $i \in [n]$, check that $y_i - v_i = y'_i \cdot (z - z_i)$.
5. Compute $\alpha := \rho(z, [(y_i, y'_i)]_{i=1}^n)$
6. Check that $v = \sum_{i=1}^n \alpha^{i-1} \cdot y_i + \sum_{i=i}^n \alpha^{n+i-1} \cdot y'_i$.
7. Check that $C = \sum_{i=1}^n \alpha^{i-1} \cdot C_i + \sum_{i=1}^n \alpha^{n+i-1} \cdot W_i$.

Decider. On input the decision key $\mathsf{dk} = \mathsf{rk}$ and an accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x} = (C, z, v), \mathsf{acc}.\mathbb{w} = \pi)$, D checks that $\mathsf{PC}_{\mathsf{HC}}.\mathsf{Check}(\mathsf{rk}, C, z, v, \pi) = 1$.

6.2 Proof of Theorem 6.1

We prove Theorem 6.1 by first proving that AS constructed above is complete and satisfies the claimed efficiency properties, and then, in Section 6.3, by proving that AS achieves knowledge soundness.

Completeness. Recalling that we need only accumulate predicate inputs, we demonstrate that the following simplified completeness property holds for every (unbounded) adversary $\mathcal{A}$:

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

We prove this directly. For each $i \in [n]$, since

$$\Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{q}_i = (C_i, z_i, v_i), \pi_i = p_i) = \mathsf{PC}_{\mathsf{HC}}.\mathsf{Check}(\mathsf{rk}, C_i, z_i, v_i, p_i) = 1 \ ,$$

we know that $C_i = \mathsf{PC}_{\mathsf{HC}}.\mathsf{Commit}(\mathsf{ck}, p_i)$ and $p_i(z_i) = v_i$; this implies that each witness polynomial $w_i(X) = \frac{p_i(X) - v_i}{X - z_i}$ is indeed a polynomial of degree $d - 1$.

Together with the fact that the accumulation prover P behaves honestly, the foregoing facts imply that C is a well-formed commitment to $p = \sum_{i=1}^n \alpha^i p_i + \sum_{i=1}^n \alpha^{n+i} w_i$, and that $p(z) = v$, as required.

Efficiency. We now analyze the efficiency of our accumulation scheme.

- *Generator:* $\mathrm{G}(1^\lambda)$ outputs 1^λ, and hence runs in time $O(\lambda)$.
- *Indexer:* $\mathrm{I}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi)$ invokes $\mathsf{PC}_{\mathsf{HC}}.\mathsf{Trim}$, and hence runs in time $O_\lambda(d)$.
- *Accumulation prover:* $\mathrm{P}^\rho(\mathsf{apk} = \mathsf{ck}, [\mathsf{q}_i = (\mathsf{q}.\mathbb{x}_i, \mathsf{q}.\mathbb{w}_i)]_{i=1}^n)$ computes a commitment to the degree $\deg(p_i) - 1$ witness polynomial w_i for each input $\mathsf{q}.\mathbb{x}_i = (C_i, z_i, v_i)$. The time to generate these n commitments dominates the running time of P.
- *Accumulator size:* The accumulator instance $\mathsf{acc}.\mathbb{x}$ consists of a polynomial commitment C, an evaluation point z and an evaluation claim v. The accumulator witness $\mathsf{acc}.\mathbb{w}$ is a polynomial of degree d. (Separately, the accumulation proof π_V contains $O(n)$ group and field elements; this efficiency measure is not so important.)
- *Accumulation verifier:* $\mathrm{V}^\rho(\mathsf{avk}, [\mathsf{q}_i.\mathbb{x}]_{i=1}^n, \mathsf{acc}.\mathbb{x}, \pi_\mathrm{V})$ computes a random linear combination between $2n$ commitments, and hence its running time is as claimed.
- *Decider:* $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ simply invokes $\mathsf{PC}_{\mathsf{HC}}.\mathsf{Check}$, which in turn invokes $\mathsf{PC}_{\mathsf{HC}}.\mathsf{Commit}$, and checks that the output matches the accumulator.

6.3 Knowledge soundness

Since we do not distinguish queries and prior accumulators, and since our predicate does not access the random oracle, we can simplify the knowledge soundness condition. In particular, it suffices to demonstrate

that for every expected polynomial-time adversary $\tilde{\mathcal{P}}$ and auxiliary input distribution $\mathcal{D}$ there exists an expected polynomial-time extractor $\mathcal{E}$ such that for every set Z the following condition holds:

$$\Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{acc}, [\mathsf{q}_i.\mathbb{x}]_{i=1}^n, \mathsf{ai}, \mathsf{ao}) \in Z \\ \wedge \\ \forall\, i \in [n],\ \Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{q}_i) = 1 \end{array} & \begin{array}{r} \mathsf{pp} \leftarrow \mathrm{G}(1^\lambda) \\ \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ \begin{pmatrix} \mathsf{i}_\Phi & \mathsf{acc} \\ [\mathsf{q}_i]_{i=1}^n & \mathsf{ao} \end{pmatrix} \leftarrow \mathcal{E}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}) \end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{acc}, [\mathsf{q}_i.\mathbb{x}]_{i=1}^n, \mathsf{ai}, \mathsf{ao}) \in Z \\ \wedge \\ \mathrm{V}^\rho(\mathsf{avk}, [\mathsf{q}_i.\mathbb{x}]_{i=1}^n, \mathsf{acc}.\mathbb{x}, \pi_\mathrm{V}) = 1 \\ \mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 1 \end{array} & \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathrm{G}(1^\lambda) \\ \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ \begin{pmatrix} \mathsf{i}_\Phi & \mathsf{acc} & \mathsf{ao} \\ [\mathsf{q}_i.\mathbb{x}]_{i=1}^n & \pi_\mathrm{V} & \end{pmatrix} \leftarrow \tilde{\mathcal{P}}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \end{array}\right] - \mathrm{negl}(\lambda)\ .$$

The remainder of this subsection is dedicated to proving the above statement. We first establish some useful notation for algorithms with access to oracles. We then give two key technical tools: an expected-time forking lemma with negligible loss, and an expected-time variant of the zero-finding game lemma of [BCMS20]. Finally, we apply these tools to prove the knowledge soundness property.

6.3.1 Notation for oracle algorithms

Let A be a t-query oracle algorithm with access to an oracle $\rho\colon \{0,1\}^* \to \{0,1\}^\lambda$. For $\vec{h} = (h_1, \ldots, h_t) \in (\{0,1\}^\lambda)^t$, we denote by $(q, y; \Gamma) \leftarrow A^{\vec{h}}(x; r)$ the following procedure: run A on input x and randomness r, and answer the i-th query q_i of A to its oracle with h_i for each i; then, when A outputs (q, y), if $q = q_k$ for some k, let k be the smallest such, and output $(q, y; \Gamma = [(q_i, h_i)]_{i=1}^k)$; otherwise, output $(q, y; \bot)$. Note that Γ is *truncated* to (and includes) the first appearance of the query q. We will write $(q, y; \Gamma) \leftarrow A^\rho(x; r)$ to denote the same procedure where h_i is defined (adaptively) to be $\rho(q_i)$ for each i.

We assume without loss of generality that A makes no oracle query more than once; in particular, we can interpret Γ as partial function $\{0,1\}^* \rightharpoonup \{0,1\}^\lambda$. For q in the support of Γ, we denote by $\Gamma[q \mapsto x]$ the partial function identical to Γ except that $\Gamma(q) = x$. If the randomness r is omitted it is assumed to be chosen uniformly.

6.3.2 An expected-time forking lemma

We give an expected-time forking lemma that is suitable for our setting. Crucially, the lemma gives a guarantee for extracting from expected-time adversaries, which will be important for extracting recursively; and it suffers only a negligible loss in success probability.

Lemma 6.3. *Let A be a t-query oracle algorithm running in expected time t_A, G be a parameter generation algorithm, and p be a t-time computable predicate. For any set Z, define*

$$\delta_Z := \Pr\left[\begin{array}{c|r} \Gamma \neq \bot \wedge (\mathsf{pp}, y) \in Z \\ \wedge\, \mathsf{p}(\mathsf{pp}, q, y, \Gamma) = 1 \end{array} & \begin{array}{r} \mathsf{pp} \leftarrow G(1^\lambda) \\ \vec{h} \leftarrow (\{0,1\}^\lambda)^t \\ (q, y; \Gamma) \leftarrow A^{\vec{h}}(\mathsf{pp}) \end{array}\right]\ .$$

Then there exists a t-query oracle algorithm B running in expected time $O(tN \cdot t_A)$ such that for all Z,

$$\Pr\left[\begin{array}{c|c} \Gamma \neq \bot \wedge (\mathsf{pp}, y_1) \in Z & \mathsf{pp} \leftarrow G(1^\lambda) \\ \wedge \forall j \in [N],\ \mathsf{p}(\mathsf{pp}, q, y_j, \Gamma[q \mapsto g_j]) = 1 & \vec{h} \leftarrow (\{0,1\}^\lambda)^t \\ \wedge \forall j \neq k \in [N],\ g_j \neq g_k & (q, [g_i, y_i]_{i=1}^N; \Gamma) \leftarrow B^{\vec{h}}(\mathsf{pp}, 1^N) \end{array}\right] \geq \delta_Z - \frac{2N\sqrt{t}}{2^{\lambda/2}} \ .$$

Proof. The algorithm $B^{\vec{h}}$ on input $(\mathsf{pp}, 1^N)$ operates as follows.

1. Draw $r \leftarrow \{0,1\}^R$.
2. Run $A^{\vec{h}}(x; r)$ until it terminates and outputs $(q, y; \Gamma)$. If $\Gamma = \bot$ or $p(x, q, \Gamma(q), y) = 0$, output $\bot$. Otherwise let $i = |\Gamma|$; in particular, $h_i = \Gamma(q)$.
3. Set $g_1 := h_i$ and $y_1 := y$.
4. Set $K := 1$ and repeat the following until $K = N$:
   (a) Draw $h'_i, \ldots, h'_t \leftarrow \{0,1\}^\lambda$, and run $A^{h_1, \ldots, h_{i-1}, h'_i, \ldots, h'_t}(x; r)$ until it terminates and outputs $(q', y'; \Gamma')$.
   (b) If $q' = q$ and $\mathsf{p}(\mathsf{pp}, q, y', \Gamma[q \mapsto h'_i]) = 1$, update $K \leftarrow K + 1$ and set $g_K := h'_i$ and $y_K := y'$.
5. Output $(q, g_1, y_1, \ldots, g_N, y_N)$.

Let $S_i := \{(\vec{h}, r) : (q, y; D) \leftarrow A^{\vec{h}}(x; r) \wedge p(x, q, y, D) = 1\}$, and let $S := \bigcup_{i=1}^t S_i$. Define

$$p_i(h_1, \ldots, h_{i-1}; r) := |\{h'_i, \ldots, h'_t \in \{0,1\}^\lambda : (h_1, \ldots, h_{i-1}, h'_i, \ldots, h'_t, r) \in S_i\}| / 2^{\lambda(t-i+1)} \ .$$

Observe that if B samples r such that $(\vec{h}, r) \in S_i$ then the probability that a single iteration of Step 4 increments K is $p_i(h_1, \ldots, h_i; r)$.

We see that B halts almost surely: if B does not terminate in Step 2 then $p_i(h_1, \ldots, h_{i-1}; r) \neq 0$. We also observe that the probability that B terminates in Step 2, or that $(\mathsf{pp}, y_1) \notin Z$, is at most $1 - \delta_Z$.

We now bound the expected running time of B. Let T_A, T_B be random variables denoting the running time of A, B respectively, and let $t(h_1, \ldots, h_{i-1}; r)$ denote the expected running time of $A^{\vec{h}}(x; r)$ over uniformly random $h_i, \ldots, h_t$. For $(\vec{h}, r) \in S_i$, the number of iterations between $K \leftarrow k$ and $K \leftarrow k+1$, which we denote $X^{(k)}_{\vec{h},r}$, is geometrically distributed with parameter $p_i(h_1, \ldots, h_{i-1}; r) \neq 0$. We denote the *time* between these increments of K by $T^{(k)}_{\vec{h},r}$ and note that for $(\vec{h}; r) \in S_i$, $E[T^k_{\vec{h},r} \mid X^{(k)}_{\vec{h},r} = j] = j \cdot t(h_1, \ldots, h_{i-1}; r)$ by linearity. Hence by law of total expectation,

$$\begin{aligned} E[T_B] &= \frac{1}{2^{\lambda t + R}} \sum_{i=1}^{t} \sum_{(\vec{h},r) \in S_i} \sum_{k=1}^{N} \sum_{j=1}^{\infty} \Pr[X^{(k)}_{\vec{h},r} = j] \cdot E[T^{(k)}_{\vec{h},r} \mid X^{(k)}_{\vec{h},r} = j] \\ &= \frac{N}{2^{\lambda t + R}} \sum_{i=1}^{t} \sum_{(\vec{h},r) \in S_i} \sum_{j=1}^{\infty} (1 - p_i(h_1, \ldots, h_{i-1}; r))^{j-1} \cdot p_i(h_1, \ldots, h_{i-1}; r) \cdot j \cdot t(h_1, \ldots, h_{i-1}; r) \\ &= \frac{N}{2^{\lambda t + R}} \sum_{i=1}^{t} \sum_{(\vec{h},r) \in S_i} t(h_1, \ldots, h_{i-1}; r) / p_i(h_1, \ldots, h_{i-1}; r) \\ &\leq N \cdot \sum_{i=1}^{t} \sum_{h_1, \ldots, h_{i-1}, r} \frac{t(h_1, \ldots, h_{i-1}; r)}{2^{\lambda(i-1)+R}} = tN \cdot E[T_A] \ . \end{aligned}$$

where the inequality follows because for all functions $f(h_1, \ldots, h_{i-1}; r)$ into $\mathbb{R}$,

$$\sum_{(\vec{h},r) \in S_i} f(h_1, \ldots, h_{i-1}; r) = 2^{\lambda(t-i+1)} \cdot \sum_{\substack{h_1, \ldots, h_{i-1}, r \\ p_i(h_1, \ldots, h_{i-1}; r) \neq 0}} f(h_1, \ldots, h_{i-1}; r) \cdot p_i(h_1, \ldots, h_{i-1}; r) \ .$$

It can be shown similarly that the expected number of iterations is at most tN, and so the probability that B performs more than $\sqrt{t} \cdot 2^{\lambda/2}$ iterations is at most $N\sqrt{t} \cdot 2^{-\lambda/2}$. Conditioned on this, the probability that in any iteration we draw h'_i such that $h'_i = g_j$ for any $j < K$ is at most $N\sqrt{t}2^{\lambda/2} \cdot 2^{-\lambda} = N\sqrt{t} \cdot 2^{-\lambda/2}$. Applying a union bound completes the proof. □

6.3.3 Zero-finding games

The following lemma, due to [BCMS20], bounds the probability that applying the random oracle to a commitment to a polynomial yields a zero of that polynomial. We refer to this as a *zero-finding game*. Here we have adapted the lemma to expected-time adversaries; the proof is essentially unchanged.

Lemma 6.4 ([BCMS20])**.** *Let $F\colon \mathbb{N} \to \mathbb{N}$, and $\mathsf{CM} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit})$ be a commitment scheme. Fix a number of variables $M \in \mathbb{N}$ and maximum degree $N \in \mathbb{N}$. Then for every family of (possibly inefficient) functions $\{f_{\mathsf{pp}}\}_{\mathsf{pp}}$ and fields $\{\mathbb{F}_{\mathsf{pp}}\}_{\mathsf{pp}}$ where $f_{\mathsf{pp}}\colon \mathcal{M}_{\mathsf{pp}} \to \mathbb{F}_{\mathsf{pp}}^{\leq N}[X_1, \ldots, X_M]$ and $|\mathbb{F}_{\mathsf{pp}}| \geq F(\lambda)$, and for every message format L and t-query expected polynomial time oracle algorithm $\mathcal{A}$, the following holds.*

$$\Pr\left[\begin{array}{c|r} & \rho \leftarrow \mathcal{U}(\lambda) \\ & \mathsf{pp} \leftarrow \mathsf{CM.Setup}(1^\lambda, L) \\ p \not\equiv 0 & (\ell, \mathfrak{p} \in \mathcal{M}_{\mathsf{ck}}, \omega) \leftarrow \mathcal{A}^\rho(\mathsf{pp}) \\ \wedge & \mathsf{ck} \leftarrow \mathsf{CM.Trim}(\mathsf{pp}, \ell) \\ p(\boldsymbol{z}) = 0 & C \leftarrow \mathsf{CM.Commit}(\mathsf{ck}, \mathfrak{p}; \omega) \\ & \boldsymbol{z} \in \mathbb{F}_{\mathsf{pp}}^N \leftarrow \rho(C) \\ & p \leftarrow f_{\mathsf{pp}}(\mathfrak{p}) \end{array}\right] \leq \sqrt{(t+1) \cdot \frac{MN}{F(\lambda)}} + \mathrm{negl}(\lambda) \ .$$

Remark 6.5. For Lemma 6.4 to hold, the algorithms of CM *must not* have access to the random oracle ρ used to generate the challenge point $\boldsymbol{z}$. The lemma is otherwise black-box with respect to CM, and so CM itself may use other oracles. The lemma continues to hold when $\mathcal{A}$ has access to these additional oracles. We use this fact later to justify the security of domain separation.

6.3.4 Proof of knowledge soundness

We formally present the extractor below and then analyze why it satisfies the knowledge soundness definition. Let $A^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai})$ be the algorithm which runs $(\mathsf{i}_\Phi, [\mathsf{q}_i.\mathbb{x}]_{i=1}^n, \mathsf{acc} = (C, z, v), \pi_{\mathrm{V}} = [(W_i, y_i, y'_i)]_{i=1}^n, \mathsf{ao}) \leftarrow \tilde{\mathrm{P}}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai})$, sets $q := (z, [y_i, y'_i]_{i=1}^n)$, and outputs $(q, (\mathsf{i}_\Phi, [\mathsf{q}_i.\mathbb{x}]_{i=1}^n, \mathsf{acc}, \pi_{\mathrm{V}}))$. Define the forking lemma predicate:

$\mathsf{p}(\mathsf{pp}, \mathsf{pp}_\Phi, q, y, \Gamma)$:
1. If Γ contains a collision, output 0.
2. Parse q as $(z, [(y_i, y'_i)]_{i=1}^n)$, and y as $(\mathsf{i}_\Phi = d, [\mathsf{q}_i.\mathbb{x}]_{i=1}^n, \mathsf{acc}, \pi_{\mathrm{V}})$. If z does not match the evaluation point in $\mathsf{acc}.\mathbb{x}$, or $[(y_i, y'_i)]_{i=1}^n$ does not match π_{V}, output 0.
3. Compute $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, d)$.
4. Run $\mathrm{V}(\mathsf{avk}, [\mathsf{q}_i.\mathbb{x}]_{i=1}^n, \mathsf{acc}.\mathbb{x}, \pi_{\mathrm{V}})$, answering its oracle queries according to Γ. If V asks a query outside of the support of Γ, or rejects, output 0.
5. Run $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$; if it rejects, output 0.
6. Otherwise, output 1.

We may assume without loss of generality that A queries the oracle at the points $(d, [(C_i, z_i, v_i, W_i)]_{i=1}^n)$ and q. The probability that A finds a collision or an inversion[6] is $O(t^2/2^{-\lambda})$, and so the output of A satisfies p

[6] An *inversion* is a query whose answer is equal to the first λ bits of any previous query, including itself.

with probability $\delta - \mathrm{negl}(\lambda)$. Let B be the algorithm given by applying Lemma 6.3 to A, p.

For the purposes of the proof, we present the extractor as an algorithm with access to a random oracle; the final extractor can then be obtained by simulating the oracle.

$\mathrm{E}^{\rho}_{\tilde{\mathcal{P}}}(\mathsf{pp}, \mathsf{pp}_\Phi)$:

1. Run $(q, \alpha_1, y_1, \ldots, \alpha_{2n}, y_{2n}; \Gamma) \leftarrow B^{\rho}(\mathsf{pp}, 1^{2n})$.
2. For $j \in [2n]$, parse y_j as $(\mathsf{i}_\Phi^{(j)} = d^{(j)}, [\mathsf{q}_i.\mathbb{x}^{(j)}]_{i=1}^n, \pi_{\mathrm{V}}^{(j)}, \mathsf{acc}^{(j)})$ and $\mathsf{acc}^{(j)}$ as $((G^{(j)}, z^{(j)}, v^{(j)}), s^{(j)})$.
3. Set $\vec{s} := \begin{pmatrix} s^{(1)} \\ \vdots \\ s^{(2n)} \end{pmatrix}$, and set $A := \begin{pmatrix} 1 & \alpha_1 & \ldots & \alpha_1^{2n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_j & \ldots & \alpha_j^{2n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{2n} & \ldots & \alpha_{2n}^{2n-1} \end{pmatrix}$.
4. If A is invertible, compute $(\vec{p} \| \vec{w}) := A^{-1} \cdot \vec{s}$; otherwise, abort.
5. Output $\left(\mathsf{i}_\Phi, \mathsf{acc}^{(1)}, [\mathsf{q}_i = (\mathsf{q}_i.\mathbb{x}^{(1)}, p_i)]_{i=1}^n, \pi_{\mathrm{V}}^{(1)}\right)$.

By Lemma 6.3, $\mathrm{E}_{\tilde{\mathrm{P}}}$ runs in expected polynomial time and with probability $\delta - \mathrm{negl}(\lambda)$ we have: $(\mathsf{pp}, \mathsf{pp}_\Phi, y_1) \in Z$, $\forall j \neq k \in [2n]$, $\alpha_j \neq \alpha_k$ and $\forall j \in [2n]$, $\mathsf{p}(\mathsf{pp}, \mathsf{pp}_\Phi, q, y_j, \Gamma[q \mapsto \alpha_j]) = 1$; call this event E.

Conditioned on E, we observe the following. First, since the α_j are distinct, A is invertible. Next, let $(z, [(y_i, y_i')]_{i=1}^n) := q$; note that $z^{(j)} = z$ and $(d^{(j)}, [\mathsf{q}_i.\mathbb{x}^{(j)}, W_i^{(j)}]_{i=1}^n) = \Gamma^{-1}(z) = (d^{(1)}, [\mathsf{q}_i.\mathbb{x}^{(1)}, W_i^{(1)}]_{i=1}^n)$ for all j, whence also $\pi_{\mathrm{V}}^{(j)} = \pi_{\mathrm{V}}^{(1)}$ for all j. For the remainder of the proof we therefore omit the superscripts on $d, z, \mathsf{q}_i.\mathbb{x} = (C_i, z_i, v_i)$, and $\pi_{\mathrm{V}} = [(W_i, y_i, y_i')]_{i=1}^n$. It is immediate that the knowledge soundness predicate p is satisfied. We now show a sequence of two claims. The first shows that the extracted polynomials $\vec{p}, \vec{w}$ are openings of the corresponding commitments, and that their evaluations at z are as claimed.

Claim 6.6. *Conditioned on E, for each $i \in [n]$, $C_i = \mathsf{PC}_{\mathsf{HC}}.\mathsf{Commit}(\mathsf{ck}, p_i)$, $W_i = \mathsf{PC}_{\mathsf{HC}}.\mathsf{Commit}(\mathsf{ck}, w_i)$, $\deg(p_i) \leq d$, $\deg(w_i) \leq d$, $p_i(z) = y_i$ and $w_i(z) = y_i'$.*

Proof. Let $\vec{C} := (C_1, \ldots, C_n)$ and $\vec{W} := (W_1, \ldots, W_n)$. Let $\vec{y} := (y_1, \ldots, y_n)$ and $\vec{y}' := (y_1', \ldots, y_n')$. Then for each j, let $((G^{(j)}, z, v^{(j)}), p^{(j)}) := \mathsf{acc}^{(j)}$, and let $\vec{G} := (G^{(1)}, \ldots, G^{(2n)})$ and $\vec{v} := (v^{(1)}, \ldots, v^{(2n)})$.

Since the verifier accepts $(\mathsf{avk}, [\mathsf{q}_i.\mathbb{x}]_{i=1}^n, \mathsf{acc}^{(j)}.\mathbb{x}, \pi_{\mathrm{V}})$ for all j, it holds that $\vec{v} = A(\vec{y} \| \vec{y}')$, and $\vec{G} = A(\vec{C} \| \vec{W})$. Moreover, since the decider accepts $(\mathsf{avk}, \mathsf{acc}^{(j)})$ for all j, it holds for all j that $G^{(j)} = \mathsf{PC}_{\mathsf{HC}}.\mathsf{Commit}(\mathsf{ck}, s^{(j)})$, $s^{(j)}(z) = v^{(j)}$ and $\deg(s^{(j)}) \leq d$, from which the degree bounds on p_i, w_i follow directly.

Using the homomorphic property of $\mathsf{PC}_{\mathsf{HC}}$, and because $(\vec{C} \| \vec{W}) = A^{-1}\vec{G}$, it holds that $C_i = \sum_j A_{i,j}^{-1} G^{(j)} = \mathsf{PC}_{\mathsf{HC}}.\mathsf{Commit}(\mathsf{ck}, \sum_j A_{i,j}^{-1} s^{(j)}) = \mathsf{PC}_{\mathsf{HC}}.\mathsf{Commit}(\mathsf{ck}, p_i)$. Similarly, $W_i = \mathsf{PC}_{\mathsf{HC}}.\mathsf{Commit}(\mathsf{ck}, w_i)$. In addition, since $(\vec{y}, \vec{y}') = A^{-1}\vec{v}$, and $p^{(j)}(z) = v^{(j)}$, we have that $p_i(z) = \sum_j A_{i,j}^{-1} v^{(j)} = y_i$, and $w_i(z) = \sum_j A_{n+i,j}^{-1} v^{(j)} = y_i'$. □

The second claim shows that the evaluations of the p_i on the original query points z_i are as claimed.

Claim 6.7. *With probability at least $\delta - \mathrm{negl}(\lambda)$, it holds that for all $i \in [n]$, $p_i(z_i) = v_i$.*

Proof. Consider a slight modification to $\mathrm{E}_{\tilde{\mathrm{P}}}$ which also outputs $\vec{w}$. From the previous claim, if E occurs, then for each i, the tuple (C_i, z_i, v_i, W_i) constitutes a binding commitment to the polynomial $p_i(X) - v_i - w_i(X)(X - z_i)$ of degree at most $d + 1$, and $p_i(z) = y_i, w_i(z) = y_i'$. Since the verifier accepts the output

of $\mathrm{E}_{\tilde{\mathrm{P}}}$, we have that $z = \Gamma(d, [(C_i, z_i, v_i, W_i)]_{i=1}^n)$, and that $y_i - v_i = y_i'(z - z_i)$. By Lemma 6.4, with probability $\delta - \mathrm{negl}(\lambda)$, $p_i(X) - v_i - w_i(X)(X - z_i)$ is the zero polynomial, and so $p_i(z_i) = v_i$. $\square$

Together these claims establish that with probability at least $\delta - \mathrm{negl}(\lambda)$, for all $i \in [n]$ it holds that $\mathsf{PC}_{\mathsf{HC}}.\mathsf{Check}(\mathsf{rk}, (C_i, z_i, v_i), p_i) = 1$, which completes the proof of knowledge soundness.

7 Implementation

We contribute a generic and modular implementation of proof-carrying data based on accumulation schemes. Our implementation includes several components of independent interest.

Framework for accumulation. We design and implement a generic framework for accumulation schemes that supports arbitrary predicates/relations. The main interface is a Rust trait that defines the behavior of any (atomic or split) accumulation scheme. We implement this trait for accumulation schemes for popular polynomial commitments (and thus for popular SNARKs that rely on PC schemes, such as [CHMMVW20]). In more detail, we implemented:

- the atomic accumulation scheme $\mathsf{AS}_{\mathsf{AGM}}$ in [BCMS20] for the PC scheme $\mathsf{PC}_{\mathsf{AGM}}$;
- the atomic accumulation scheme $\mathsf{AS}_{\mathsf{IPA}}$ in [BCMS20] for the PC scheme $\mathsf{PC}_{\mathsf{IPA}}$;
- the split accumulation scheme $\mathsf{AS}_{\mathsf{HC}}$ in Section 6 for $\mathsf{PC}_{\mathsf{HC}}$.

Our framework also provides a generic trait for defining R1CS constraints for the verifier of an accumulation scheme. We use this trait to implement R1CS constraints for all these accumulation schemes.

PCD from accumulation. We provide a generic construction of PCD from accumulation, which simultaneously supports the case of atomic accumulation from [BCMS20] and the case of split accumulation from Section 5. Our code builds on and extends an existing PCD library that offers a generic "PCD" trait.[7] We instantiate this PCD trait via a modular construction, which takes as ingredients any NARK (as defined by an appropriate trait), accumulation scheme for that NARK that implements the accumulation trait (from above), and constraints for the accumulation verifier. We use our concrete instantiations of these ingredients to construct PCD that is based on atomic accumulation of $\mathsf{PC}_{\mathsf{AGM}}$ and $\mathsf{PC}_{\mathsf{IPA}}$, and split accumulation of $\mathsf{PC}_{\mathsf{HC}}$.

Cycles of elliptic curves. All PCD constructions in our implementation rely on the technique of cycles of elliptic curves [BCTV14]: PCD based on $\mathsf{PC}_{\mathsf{AGM}}$ uses cycles of pairing-friendly curves, while PCD based on $\mathsf{PC}_{\mathsf{IPA}}$ and $\mathsf{PC}_{\mathsf{HC}}$ uses cycles of standard curves. For all of these, we rely on existing implementations from the `arkworks` ecosystem:[8] for pairing-friendly cycles we use the MNT cycle of curves (low security and high security variants), while for standard cycles we use the Pasta cycle of curves [Hop20].

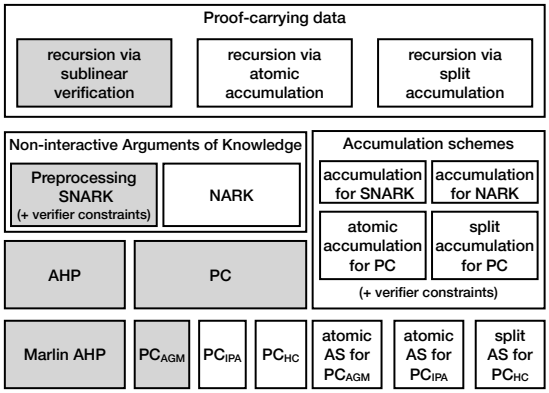

Figure 5: Diagram illustrating components in our implementation. The gray boxes denote components that exist in prior libraries, while the white boxes denote components contributed in this work.

[7] `https://github.com/arkworks-rs/pcd`
[8] `https://github.com/arkworks-rs/curves`

8 Evaluation

We perform an evaluation focused on understanding the tradeoffs of atomic versus split accumulation in the discrete logarithm setting.[9] We compare two routes for recursion:

- recursion based on the atomic accumulation scheme $\mathsf{AS}_{\mathsf{IPA}}$ in [BCMS20] for the PC scheme $\mathsf{PC}_{\mathsf{IPA}}$;
- recursion based on the split accumulation scheme $\mathsf{AS}_{\mathsf{HC}}$ in Section 6 for $\mathsf{PC}_{\mathsf{HC}}$.

We proceed as follows. In Section 8.1 we compare the two polynomial commitment schemes $\mathsf{PC}_{\mathsf{IPA}}$ and $\mathsf{PC}_{\mathsf{HC}}$, and in Section 8.2 we compare the two corresponding accumulation schemes $\mathsf{AS}_{\mathsf{IPA}}$ and $\mathsf{AS}_{\mathsf{HC}}$.

Experimental setup. All experiments are performed on a machine with an Intel Xeon 6136 CPU at 3.0 GHz. The reported numbers are for schemes instantiated over the 255-bit prime-order Pallas curve in the Pasta cycle [Hop20]; results for the Vesta curve in that cycle would be similar.

8.1 Comparing polynomial commitments based on DLs

We compare the performance of $\mathsf{PC}_{\mathsf{IPA}}$ and $\mathsf{PC}_{\mathsf{HC}}$ in Table 2, reporting experiments for an illustrative choice of polynomial degree d. In both PC schemes all operations (commit, open, check) are linear in the degree d, though for $\mathsf{PC}_{\mathsf{HC}}$ opening is concretely much cheaper than $\mathsf{PC}_{\mathsf{IPA}}$ (primarily because $\mathsf{PC}_{\mathsf{HC}}$ has a trivial opening procedure). The main difference between the two PC schemes is that an evaluation proof in $\mathsf{PC}_{\mathsf{HC}}$ is $O(d)$ field elements while an evaluation proof in $\mathsf{PC}_{\mathsf{IPA}}$ is $O(\log d)$ group elements; this asymptotic difference is apparent in the reported numbers (the proof size for $\mathsf{PC}_{\mathsf{HC}}$ is significantly larger than for $\mathsf{PC}_{\mathsf{IPA}}$). We also report lines of code to realize the same abstract PC scheme trait, to support the (intuitive) claim that $\mathsf{PC}_{\mathsf{HC}}$ is a much simpler primitive than $\mathsf{PC}_{\mathsf{IPA}}$.

PC scheme	Commit	Open	Check	$\lvert C\rvert$	$\lvert\pi\rvert$	LoC
$\mathsf{PC}_{\mathsf{IPA}}$	8.0 s	106.6 s	8.2 s	33 B	1.4 kB $O(\log d)$ $\mathbb{G}$	1120
$\mathsf{PC}_{\mathsf{HC}}$	8.1 s	0.43 s	8.3 s	33 B	33.5 MB $O(d)$ $\mathbb{F}$	608

Table 2: Comparison between the PC schemes $\mathsf{PC}_{\mathsf{IPA}}$ and $\mathsf{PC}_{\mathsf{HC}}$ for polynomials of degree $d = 2^{20}$.

8.2 Comparing accumulation schemes based on DLs

We compare the performance of $\mathsf{AS}_{\mathsf{IPA}}$ and $\mathsf{AS}_{\mathsf{HC}}$ in Table 3, reporting experiments for an illustrative choice of polynomial degree d. We focus on the special case where the accumulation scheme is used to accumulate one new polynomial evaluation claim into one old accumulator to obtain a new accumulator. Our experiments indicate that $\mathsf{AS}_{\mathsf{HC}}$ is cheaper than $\mathsf{AS}_{\mathsf{IPA}}$ across all metrics *except for accumulator size*, and more generally that performance is consistent with the asymptotic comparison from Table 1. In more detail:

- While prover time (per claim) in both $\mathsf{AS}_{\mathsf{IPA}}$ and $\mathsf{AS}_{\mathsf{HC}}$ are linear in the degree d, our experiments show that $\mathsf{AS}_{\mathsf{IPA}}$ is concretely much more expensive than $\mathsf{AS}_{\mathsf{HC}}$.
- Decider time in both $\mathsf{AS}_{\mathsf{IPA}}$ and $\mathsf{AS}_{\mathsf{HC}}$ are linear in the degree d, and our experiments show that the two schemes have similar concrete performance.
- Verifier time (per claim) in $\mathsf{AS}_{\mathsf{IPA}}$ is logarithmic while in $\mathsf{AS}_{\mathsf{HC}}$ it is constant, and our experiments confirm that $\mathsf{AS}_{\mathsf{IPA}}$ is concretely significantly more expensive than $\mathsf{AS}_{\mathsf{HC}}$.
- Verifier constraint cost is *much* higher for $\mathsf{AS}_{\mathsf{IPA}}$, even though both schemes use the same underlying constraint gadget libraries.

[9] The pairing setting is also part of our implementation, as described in Section 7, but we do not include an evaluation for it here.

- The size of an atomic accumulator for $\mathsf{AS}_{\mathsf{IPA}}$ is logarithmic, and amounts to a few kilobytes; in contrast an accumulator for $\mathsf{AS}_{\mathsf{HC}}$ is much larger, but is split into a short instance part (106 bytes) and a long witness part (33.5 megabytes).

Overall the expensive parts of $\mathsf{AS}_{\mathsf{HC}}$ are exactly where intended (a large accumulation witness part) in exchange for a very cheap verifier and a very short accumulation instance part; all other metrics are comparable to (and concretely better than for) $\mathsf{AS}_{\mathsf{IPA}}$.

scheme	P	V	D	|acc|		|V|	LoC	
				|x|	|w|		native	constraints
$\mathsf{AS}_{\mathsf{IPA}}$	117.6 s	14 ms	8.3 s	1.58 kB	0	1006×10^3	664	1232
$\mathsf{AS}_{\mathsf{HC}}$	25.2 s	2 ms	8.1 s	106 B	33.5 MB	61×10^3	571	395

Table 3: Comparison between the accumulation schemes $\mathsf{AS}_{\mathsf{IPA}}$ and $\mathsf{AS}_{\mathsf{HC}}$ for polynomials of degree $d = 2^{20}$, when accumulating one old accumulator and one evaluation claim into a new accumulator.

In Figure 6, we also compare $|\mathrm{V}|$ (the constraint cost of V) in both $\mathsf{AS}_{\mathsf{HC}}$ and $\mathsf{AS}_{\mathsf{IPA}}$ as we accumulate polynomial evaluation claims of degree d in the range 2^{10} to 2^{20}. As expected, the cost for $\mathsf{AS}_{\mathsf{HC}}$ is a small constant, whereas the cost of $\mathsf{AS}_{\mathsf{IPA}}$ grows logarithmically (and is concretely much larger).

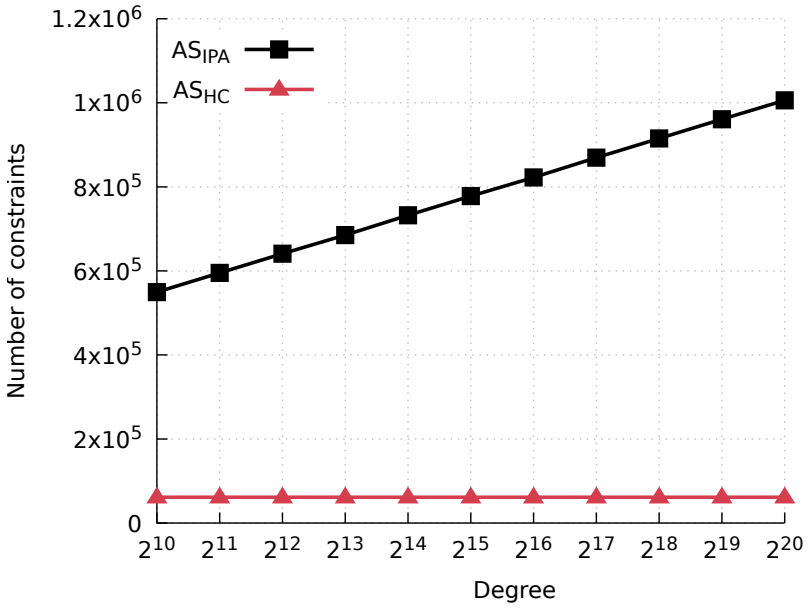

Figure 6: Comparison of the constraint cost of the accumulation verifier V in $\mathsf{AS}_{\mathsf{IPA}}$ and $\mathsf{AS}_{\mathsf{HC}}$ when varying the degree of the accumulated polynomial from 2^{10} to 2^{20}.

A Split accumulation scheme for R1CS

We describe a split accumulation scheme $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ for the NARK for R1CS in Figure 3. As a subroutine we use an accumulation scheme $\mathsf{AS}_{\mathsf{HC}} = (\mathrm{G}_{\mathsf{HC}}, \mathrm{I}_{\mathsf{HC}}, \mathrm{P}_{\mathsf{HC}}, \mathrm{V}_{\mathsf{HC}}, \mathrm{D}_{\mathsf{HC}})$ for the Pedersen PC scheme $\mathsf{PC}_{\mathsf{HC}}$ (e.g., the one we construct in Section 6). We use domain separation on the given random oracle ρ for different tasks: we use ρ_{HC} to denote the oracle used for one invocation of $\mathsf{AS}_{\mathsf{HC}}$; ρ_{NARK} to denote the oracle used to run the NARK for R1CS; and ρ_{AS} to denote the random oracle used by AS for other tasks.

Predicate inputs. The predicate to accumulate is the NARK verifier, with the following split in a predicate input q obtained from an R1CS instance x and proof π:

- The instance part of q consists of the R1CS input x, the first prover message $(C_w, C_A, C_B, C_C, C_q, C_w)$, and the claimed evaluations sent in the second prover message $(v_A, v_B, v_C, v_q, v_H)$. This amounts to 6 group elements and $|x| + 5$ field elements (which is short).
- The witness part of q consists of the openings sent in the second prover message (w, z_A, z_B, z_C, q). This amounts to $O(n + m)$ field elements (which is proportional to the dimensions of the R1CS matrices).

Accumulator. The format of an accumulator acc is as follows:

- The instance part of acc consists of $\mathsf{acc}.\mathbb{x} = (\mathsf{acc}_{\mathsf{HC}}.\mathbb{x}, x, C_w, C_A, C_B, C_C)$.
- The witness part of acc consists of $\mathsf{acc}.\mathbb{w} = (\mathsf{acc}_{\mathsf{HC}}.\mathbb{w}, w, z_A, z_B, z_C)$.

Generator. The generator G runs G_{HC} as a subroutine and outputs its output.

Indexer. The indexer I receives as input accumulation public parameters pp (output by G), predicate public parameters $\mathsf{pp}_\Phi = \mathsf{pp}_{\mathsf{NARK}}$ (they are the public parameters of the NARK per Definition 4.1), predicate index $\mathsf{i}_\Phi = (A, B, C)$ (it is the index of the relation verified by the NARK per Definition 4.1), and works as follows:

- Hash the coefficient matrices: $\sigma := \rho_{\mathsf{AS}}(A, B, C)$ and $\tau := \rho_{\mathsf{NARK}}(A, B, C)$.
- Set the degree bound to be $d := m$, the number of rows in each R1CS coefficient matrix.
- Trim the public parameters of $\mathsf{PC}_{\mathsf{HC}}$ for degree d: $\mathsf{ck} \leftarrow \mathsf{CM.Trim}(\mathsf{pp}_{\mathsf{NARK}}, d+1)$. (In the NARK the public parameters are the public parameters for the commitment scheme.)
- Invokes $\mathrm{I}_{\mathsf{HC}}(\mathsf{pp}, \mathsf{pp}_\Phi, d)$ to obtain $(\mathsf{apk}_{\mathsf{HC}}, \mathsf{avk}_{\mathsf{HC}}, \mathsf{dk}_{\mathsf{HC}})$.
- Output $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \big((\sigma, \tau, \mathsf{apk}_{\mathsf{HC}}), (\sigma, \tau, \mathsf{avk}_{\mathsf{HC}}), ((A, B, C), \mathsf{ck}, \mathsf{dk}_{\mathsf{HC}})\big)$.

Decider. The decider D receives as input the decision key $\mathsf{dk} = ((A, B, C), \mathsf{ck}, \mathsf{dk}_{\mathsf{HC}})$ and an accumulator acc (both instance and witness parts) and determines the validity of the accumulator as follows.

1. Set $z := (\mathsf{acc}.x, \mathsf{acc}.w)$.
2. Check that $\mathsf{acc}.C_w = \mathsf{CM.Commit}(\mathsf{ck}, \mathsf{acc}.w)$.
3. Check that $\mathsf{acc}.C_A = \mathsf{CM.Commit}(\mathsf{ck}, \mathsf{acc}.z_A)$, $\mathsf{acc}.C_B = \mathsf{CM.Commit}(\mathsf{ck}, \mathsf{acc}.z_B)$, $\mathsf{acc}.C_C = \mathsf{CM.Commit}(\mathsf{ck}, \mathsf{acc}.z_C)$.
4. Check that $\mathsf{acc}.z_A = A \cdot z$, $\mathsf{acc}.z_B = B \cdot z$, $\mathsf{acc}.z_C = Cz$.
5. Check that $\mathrm{D}_{\mathsf{HC}}(\mathsf{dk}_{\mathsf{HC}}, \mathsf{acc.acc}_{\mathsf{HC}}) = 1$.

Prover. The prover P, given a random oracle ρ, receives as input the accumulator proving key $\mathsf{apk} = (\sigma, \tau, \mathsf{apk}_{\mathsf{HC}})$, inputs $[\mathsf{q}_i = (\mathsf{q}.\mathbb{x}_i, \mathsf{q}.\mathbb{w}_i)]_{i=1}^n$, and old accumulators $[\mathsf{acc}_j = (\mathsf{acc}.\mathbb{x}_j, \mathsf{acc}.\mathbb{w}_j)]_{j=1}^m$, and outputs a new accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ and a proof π_{V} that are computed as follows.

1. Compute $\alpha := \rho_{\mathsf{AS}}(\sigma, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, [\mathsf{q}_i.\mathbb{x}]_{i=1}^n)$.
2. For each $i \in [n]$, compute a combined polynomial evaluation instance $\mathsf{q}_{\mathsf{HC},i}.\mathbb{x} := (C, \gamma, v)$ and witness $\mathsf{q}_{\mathsf{HC},i}.\mathbb{w} := f$ where

$$\mathsf{q}_{\mathsf{HC},i}.\mathbb{x}.C := \mathsf{q}_i.C_A + \alpha \cdot \mathsf{q}_i.C_B + \alpha^2 \cdot \mathsf{q}_i.C_C + \alpha^3 \cdot \mathsf{q}_i.C_q + \alpha^4 \cdot \mathsf{q}_i.C_H \ ,$$

$$\mathsf{q}_{\mathsf{HC},i}.\mathbb{x}.\gamma := \rho_{\mathsf{NARK}}(\tau, \mathsf{q}_i.x, \mathsf{q}_i.\{C_w, C_A, C_B, C_C, C_q, C_H\}) \ ,$$
$$\mathsf{q}_{\mathsf{HC},i}.\mathbb{x}.v := \mathsf{q}_i.v_A + \alpha \cdot \mathsf{q}_i.v_B + \alpha^2 \cdot \mathsf{q}_i.v_C + \alpha^3 \cdot \mathsf{q}_i.v_q + \alpha^4 \cdot \mathsf{q}_i.v_H \ ,$$
$$\mathsf{q}_{\mathsf{HC},i}.\mathbb{w}.f := \mathsf{q}_i.z_A + \alpha \cdot \mathsf{q}_i.z_B + \alpha^2 \cdot \mathsf{q}_i.z_C + \alpha^3 \cdot \mathsf{q}_i.q + \alpha^4 \cdot \mathsf{q}_i.V_H \ .$$

3. Accumulate all polynomial evaluations: $(\mathsf{acc}_{\mathsf{HC}}, \pi_{\mathrm{V},\mathsf{HC}}) := \mathrm{P}^{\rho_{\mathsf{HC}}}(\mathsf{apk}_{\mathsf{HC}}, [\mathsf{acc}_j.\mathsf{acc}_{\mathsf{HC}}]_{j=1}^m, [\mathsf{q}_{\mathsf{HC},i}]_{i=1}^n)$.
4. Accumulate the linear computation for the instance part:

$$\mathsf{acc}.x := \textstyle\sum_{j=1}^m \alpha^j \cdot \mathsf{acc}_j.x + \sum_{i=1}^n \alpha^{m+i} \cdot \mathsf{q}_i.x \ ,$$
$$\mathsf{acc}.C_w := \textstyle\sum_{j=1}^m \alpha^j \cdot \mathsf{acc}_j.C_w + \sum_{i=1}^n \alpha^{m+i} \cdot \mathsf{q}_i.C_w \ ,$$
$$\mathsf{acc}.C_A := \textstyle\sum_{j=1}^m \alpha^j \cdot \mathsf{acc}_j.C_A + \sum_{i=1}^n \alpha^{m+i} \cdot \mathsf{q}_i.C_A \ ,$$
$$\mathsf{acc}.C_B := \textstyle\sum_{j=1}^m \alpha^j \cdot \mathsf{acc}_j.C_B + \sum_{i=1}^n \alpha^{m+i} \cdot \mathsf{q}_i.C_B \ ,$$
$$\mathsf{acc}.C_C := \textstyle\sum_{j=1}^m \alpha^j \cdot \mathsf{acc}_j.C_C + \sum_{i=1}^n \alpha^{m+i} \cdot \mathsf{q}_i.C_C \ .$$

5. Accumulate the linear computation for the witness part:

$$\mathsf{acc}.w := \textstyle\sum_{j=1}^m \alpha^j \cdot \mathsf{acc}_j.w + \sum_{i=1}^n \alpha^{m+i} \cdot \mathsf{q}_i.w \ ,$$
$$\mathsf{acc}.z_A := \textstyle\sum_{j=1}^m \alpha^j \cdot \mathsf{acc}_j.z_A + \sum_{i=1}^n \alpha^{m+i} \cdot \mathsf{q}_i.z_A \ ,$$
$$\mathsf{acc}.z_B := \textstyle\sum_{j=1}^m \alpha^j \cdot \mathsf{acc}_j.z_B + \sum_{i=1}^n \alpha^{m+i} \cdot \mathsf{q}_i.z_B \ ,$$
$$\mathsf{acc}.z_C := \textstyle\sum_{j=1}^m \alpha^j \cdot \mathsf{acc}_j.z_C + \sum_{i=1}^n \alpha^{m+i} \cdot \mathsf{q}_i.z_C \ .$$

6. Set $\mathsf{acc}.\mathbb{x} := (\mathsf{acc}_{\mathsf{HC}}.\mathbb{x}, x, C_w, C_A, C_B, C_C)$ and $\mathsf{acc}.\mathbb{w} := (\mathsf{acc}_{\mathsf{HC}}.\mathbb{w}, w, z_A, z_B, z_C)$.
7. Output the new accumulator $\mathsf{acc} := (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ and accumulation proof $\pi_{\mathrm{V}} := \pi_{\mathrm{V},\mathsf{HC}}$.

Verify. The verifier V, given a random oracle ρ, receives as input the accumulator verification key $\mathsf{avk} = (\sigma, \tau, \mathsf{avk}_{\mathsf{HC}})$, input instances $[\mathsf{q}_i.\mathbb{x}]_{i=1}^n$, accumulator instances $[\mathsf{acc}_j.\mathbb{x}]_{j=1}^m$, a new accumulator instance $\mathsf{acc}.\mathbb{x}$, and a proof π_{V}, and determines whether to accept or reject as follows.

1. Compute $\alpha \in \mathbb{F}$ as in Item 1 of the accumulation prover.
2. For each $i \in [n]$, compute the polynomial evaluation instance $\mathsf{q}_{\mathsf{HC},i}.\mathbb{x}$ as in Item 2 of the accumulation prover (excluding the witness computations) and check that

$$\mathsf{q}_i.v_q \cdot \mathsf{q}_i.v_H = \mathsf{q}_i.v_A \cdot \mathsf{q}_i.v_B - \mathsf{q}_i.v_C \ .$$

3. Verify accumulation of polynomial evaluations: $\mathrm{V}_{\mathsf{HC}}^{\rho_{\mathsf{HC}}}(\mathsf{avk}_{\mathsf{HC}}, [\mathsf{acc}_j.\mathsf{acc}_{\mathsf{HC}}]_{j=1}^m, [\mathsf{q}_{\mathsf{HC},i}]_{i=1}^n, \mathsf{acc}_{\mathsf{HC}}, \pi_{\mathrm{V},\mathsf{HC}}) = 1$.
4. Check the accumulation of linear computations by performing the assignments in Item 4 of the accumulation prover as equality checks (between the new accumulator instance and the input instances and old accumulator instances).

Acknowledgements

This research was supported in part by the Ethereum Foundation, NSF, DARPA, a grant from ONR, and the Simons Foundation. Nicholas Spooner was supported by DARPA under Agreement No. HR00112020023.

References

[BBBPWM18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P '18. 2018, pp. 315–334.

[BCCGP16] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting". In: *Proceedings of the 35th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '16. 2016, pp. 327–357.

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

[BCMS20] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. "Proof-Carrying Data from Accumulation Schemes". In: TCC '20 (2020).

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

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

[BDFG20] D. Boneh, J. Drake, B. Fisch, and A. Gabizon. *Halo Infinite: Recursive zk-SNARKs from any Additive Polynomial Commitment Scheme*. Cryptology ePrint Archive, Report 2020/1536. 2020.

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

[BMMTV19] B. Bünz, M. Maller, P. Mishra, N. Tyagi, and P. Vesely. *Proofs for Inner Pairing Products and Applications*. Cryptology ePrint Archive, Report 2019/1177. 2019.

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

[BN06] M. Bellare and G. Neven. "Multi-signatures in the plain public-Key model and a general forking lemma". In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS '06. 2006, pp. 390–399.

[BR93] M. Bellare and P. Rogaway. "Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. CCS '93. 1993, pp. 62–73.

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

[CHMMVW20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '20. 2020, pp. 738–768.

[COS20] A. Chiesa, D. Ojha, and N. Spooner. "Fractal: Post-Quantum and Transparent Recursive Proofs from Holography". In: *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '20. 2020, pp. 769–793.

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

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

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

[GT20] A. Ghoshal and S. Tessaro. *Tight State-Restoration Soundness in the Algebraic Group Model*. Cryptology ePrint Archive, Report 2020/1351. 2020.

[Halo20] S. Bowe, J. Grigg, and D. Hopwood. *Halo2*. 2020. URL: `https://github.com/zcash/halo2`.

[Hop20] D. Hopwood. *The Pasta Curves for Halo 2 and Beyond*. `https://electriccoin.co/blog/the-pasta-curves-for-halo-2-and-beyond/`. 2020.

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

[KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. “Constant-Size Commitments to Polynomials and Their Applications”. In: *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT ’10. 2010, pp. 177–194.

[Lin03] Y. Lindell. “Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation”. In: *Journal of Cryptology* 16.3 (2003), pp. 143–184.

[Mina] O(1) Labs. *Mina Cryptocurrency*. `https://minaprotocol.com/`. 2017.

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

[Pas03] R. Pass. “On Deniability in the Common Reference String and Random Oracle Model”. In: *Proceedings of the 23rd Annual International Cryptology Conference*. CRYPTO ’03. 2003, pp. 316–337.

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

[Pickles20] O(1) Labs. *Pickles*. URL: `https://github.com/o1-labs/marlin`.

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

[WTSTW18] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. “Doubly-Efficient zkSNARKs Without Trusted Setup”. In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P ’18. 2018, pp. 926–943.