Recursive Proof Composition from Accumulation Schemes

Benedikt Bünz[1], Alessandro Chiesa[2], Pratyush Mishra[2] and Nicholas Spooner[2]

[1] Stanford University
benedikt@cs.stanford.edu
[2] UC Berkeley
{alexch,pratyush,nick.spooner}@berkeley.edu

Abstract. Recursive proof composition has been shown to lead to powerful primitives such as incrementally-verifiable computation (IVC) and proof-carrying data (PCD). All existing approaches to recursive composition take a succinct non-interactive argument of knowledge (SNARK) and use it to prove a statement about its own verifier. This technique requires that the verifier run in time sublinear in the size of the statement it is checking, a strong requirement that restricts the class of SNARKs from which PCD can be built. This in turn restricts the efficiency and security properties of the resulting scheme.
Bowe, Grigg, and Hopwood (ePrint 2019/1021) outlined a novel approach to recursive composition, and applied it to a particular SNARK construction which does *not* have a sublinear-time verifier. However, they omit details about this approach and do not prove that it satisfies any security property. Nonetheless, schemes based on their ideas have already been implemented in software.
In this work we present a collection of results that establish the theoretical foundations for a generalization of the above approach. We define an *accumulation scheme* for a non-interactive argument, and show that this suffices to construct PCD, even if the argument itself does not have a sublinear-time verifier. Moreover we give constructions of accumulation schemes for SNARKs, which yield PCD schemes with novel efficiency and security features.

Keywords: succinct arguments; proof-carrying data; recursive proof composition

1 Introduction

Proof-carrying data (PCD) [CT10] is a 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 succinctly verified. PCD supports computations defined on (possibly infinite) directed acyclic graphs, with messages passed along directed edges. Verification is facilitated by attaching to each message a succinct proof of correctness. This is a generalization of the notion of *incrementally-verifiable computation* (IVC) due to [Val08], which can be viewed as PCD for the path graph (i.e., for automata). PCD has found applications in enforcing language semantics [CTV13], verifiable MapReduce computations [CTV15], image authentication [NT16], succinct blockchains [Co17; KB20; BMRS20], and others.

Recursive composition. Prior to this work, the only known method for constructing PCD was from *recursive composition* of succinct non-interactive arguments (SNARGs) [BCCT13; BCTV14; COS20]. This method informally works as follows. 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). The efficiency and security properties of the resulting PCD scheme correspond to those of a single invocation of the SNARK.

Limitations of recursion. Recursion as realized in prior work requires proving a statement that contains a description of the SNARK verifier. In particular, for efficiency, we must ensure that the statement we are proving (essentially) *does not grow* with the number of recursion steps t. For example, if the representation of the verifier were to grow even *linearly* with the statement it is verifying, then the size of the statement to be checked would grow *exponentially* in t. Therefore, prior works have achieved efficiency by focusing on SNARKs which admit sublinear-time verification: either SNARKs for machine computations [BCCT13] or preprocessing SNARKs for circuit computations [BCTV14; COS20]. Requiring sublinear-time verification significantly restricts our choice of SNARK, which limits what we can achieve for PCD.

In addition to the above asymptotic considerations, recursion raises additional considerations concerning concrete efficiency. All SNARK constructions require that statements be encoded as instances of some particular (algebraic) NP-complete problem, and difficulties often arise when encoding the SNARK verifier itself as such an instance. The most well-known example of this is in recursive composition of pairing-based SNARKs, since the verifier performs operations over a finite field that is necessarily different from the field supported "natively" by the NP-complete problem [BCTV14]. This type of problem also appears when recursing SNARKs whose verifiers make heavy use of cryptographic hash functions [COS20].

A new technique. Bowe, Grigg, and Hopwood [BGH19] suggest an exciting novel approach to recursive composition that replaces the SNARK verifier in the circuit with a simpler algorithm. This algorithm does not itself verify the previous proof π_{t-1}. Instead, it adds the proof to an *accumulator* for verification at the end. The accumulator must not grow in size. A key contribution of [BGH19] is to sketch a mechanism by which this might be achieved for a particular SNARK construction. While they prove this SNARK construction secure, they do not include definitions or proofs of security for their recursive technique. Nonetheless, practitioners have already built software based on these ideas [Halo19; Pickles20].

1.1 Our contributions

In this work we provide a collection of results that establish the theoretical foundations for the above approach. We introduce the cryptographic object, an *accumulation scheme*, that enables this technique, and prove that it suffices for constructing PCD. We then provide generic tools for building accumulation schemes, as well as several concrete

instantiations. Our framework establishes the security of schemes that are already being used by practitioners, and we believe that it will simplify and facilitate further research in this area.

Accumulation schemes. We introduce the notion of an *accumulation scheme* for a predicate $\Phi\colon X \to \{0,1\}$. This formalizes, and generalizes, an idea outlined in [BGH19]. An accumulation scheme is best understood in the context of the following process. Consider an infinite stream $\mathsf{q}_1, \mathsf{q}_2, \ldots$ with each $\mathsf{q}_i \in X$. We augment this stream with *accumulators* acc_i as follows: at time i, the *accumulation prover* receives $(\mathsf{q}_i, \mathsf{acc}_{i-1})$ and computes acc_i; the *accumulation verifier* receives $(\mathsf{q}_i, \mathsf{acc}_{i-1}, \mathsf{acc}_i)$ and checks that acc_{i-1} and q_i were correctly accumulated into acc_i (if not, the process ends). Then at any time t, the *decider* can validate acc_t, which establishes that, for *all* $i \in [t]$, $\Phi(\mathsf{q}_i) = 1$. All three algorithms are stateless. To avoid trivial constructions, we want (i) the accumulation verifier to be more efficient than Φ, and (ii) the size of an accumulator (and hence the running time of the three algorithms) does not grow over time. Accumulation schemes are powerful, as we demonstrate next.

Recursion from accumulation. We say that a SNARK has an accumulation scheme if the predicate corresponding to its verifier has an accumulation scheme (so X is a set of instance-proof pairs). We show that any SNARK having an accumulation scheme where the *accumulation verifier* is sublinear can be used to build a proof-carrying data (PCD) scheme, *even if the SNARK verifier is not itself sublinear*. This broadens the class of SNARKs from which PCD can be built. Similarly to [COS20], we show that if the SNARK and accumulation scheme are post-quantum secure, so is the PCD scheme. (Though it remains an open question whether there are non-trivial accumulation schemes for post-quantum SNARKs.)

Theorem 1 (informal)**.** *There is an efficient transformation that compiles any SNARK with an efficient accumulation scheme into a PCD scheme. If the SNARK and its accumulation scheme are zero knowledge, then the PCD scheme is also zero knowledge. Additionally, if the SNARK and its accumulation scheme are post-quantum secure then the PCD scheme is also post-quantum secure.*

The above theorem holds in the standard model (where all parties have access to a common reference string, but no oracles). Since our construction makes non-black-box use of the accumulation scheme verifier, the theorem does not carry over to the random oracle model (ROM). It remains an intriguing open problem to determine whether or not SNARKs in the ROM imply PCD in the ROM (and if the latter is even possible).

Note that we require a suitable definition of zero knowledge for an accumulation scheme. This is not trivial, and our definition is informed by what is required for Theorem 1 and what our constructions achieve.

Proof-carrying data is a powerful primitive: it implies IVC and, further assuming collision-resistant hash functions, also efficient SNARKs for machine computations. Hence, Theorem 1 may be viewed as an extension of the "bootstrapping" theorem of [BCCT13] to certain non-succinct-verifier SNARKs.

See Section 2.1 for a summary of the ideas behind Theorem 1, and the full version for technical details.

Accumulation from accumulation. Given the above, a natural question is: where do accumulation schemes for SNARKs come from? In [BGH19] it was informally observed that a specific SNARK construction, based on the hardness of the discrete logarithm problem, has an accumulation scheme. To show this, [BGH19] first observe that the verifier in the SNARK construction is sublinear *except for* the evaluation of a certain predicate (checking an opening of a polynomial commitment [KZG10]), then outline a construction which is essentially an accumulation scheme for that predicate.

We prove that this idea is a special case of a general paradigm for building accumulation schemes for SNARKs.

Theorem 2 (informal)**.** *There is an efficient transformation that, given a SNARK whose verifier is succinct when given oracle access to a "simpler" predicate, and an accumulation scheme for that predicate, constructs an accumulation scheme for the SNARK. Moreover, this transformation preserves zero knowledge and post-quantum security of the accumulation scheme.*

The construction underlying Theorem 2 is black-box. In particular, if both the SNARK and the accumulation scheme for the predicate are secure with respect to an oracle, then the resulting accumulation scheme for the SNARK is secure with respect to that oracle.

See Section 2.3 for a summary of the ideas behind Theorem 2, and the full version for technical details.

Accumulating polynomial commitments. Several works [MBKM19; GWC19; CHM+20] have constructed SNARKs whose verifiers are succinct relative to a specific predicate: checking the opening of a *polynomial commitment* [KZG10]. We prove that two natural polynomial commitment schemes possess accumulation schemes in the random oracle model: $\mathsf{PC}_{\mathsf{DL}}$, a scheme based on the security of discrete logarithms [BCC+16; BBB+18; WTS+18]; and $\mathsf{PC}_{\mathsf{AGM}}$, a scheme based on knowledge assumptions in bilinear groups [KZG10; CHM+20].

Theorem 3 (informal)**.** *In the random oracle model, there exist (zero knowledge) accumulation schemes for* $\mathsf{PC}_{\mathsf{DL}}$ *and* $\mathsf{PC}_{\mathsf{AGM}}$ *that achieve the efficiency outlined in the table below (n denotes the number of evaluation proofs, and d denotes the degree of committed polynomials).*

polynomial commitment	assumption	cost to check evaluation proofs	cost to check an accumulation step	cost to check final accumulator	accumulator size
$\mathsf{PC}_{\mathsf{DL}}$	DLOG + RO	$\Theta(nd)$ $\mathbb{G}$ mults.	$\Theta(n \log d)$ $\mathbb{G}$ mults.	$\Theta(d)$ $\mathbb{G}$ mults.	$\Theta(\log d)$ $\mathbb{G}$
$\mathsf{PC}_{\mathsf{AGM}}$	AGM + RO	$\Theta(n)$ pairings	$\Theta(n)$ $\mathbb{G}_1$ mults.	1 pairing	2 $\mathbb{G}_1$

For both schemes the cost of checking that an accumulation step was performed correctly is *much less* than the cost of checking an evaluation proof. We can apply Theorem 2 to combine either of these accumulation schemes for polynomial commitments with any of the aforementioned predicate-efficient SNARKs, which yields concrete accumulation schemes for these SNARKs with the same efficiency benefits.

We remark that our accumulation scheme for $\mathsf{PC}_{\mathsf{DL}}$ is a variation of a construction presented in [BGH19], and so our result establishes the security of a type of construction used by practitioners.

We sketch the constructions underlying Theorem 3 in Section 2.4, and provide details in the full version of our paper.

New constructions of PCD. By combining our results, we (heuristically) obtain constructions of PCD that achieve new properties. Namely, starting from either $\mathsf{PC}_{\mathsf{DL}}$ or $\mathsf{PC}_{\mathsf{AGM}}$, we can apply Theorem 2 to a suitable SNARK to obtain a SNARK with an accumulation scheme in the random oracle model. Then we can instantiate the random oracle, obtaining a SNARK and accumulation scheme with *heuristic* security in the standard (CRS) model, to which we apply Theorem 1 to obtain a corresponding PCD scheme. Depending on whether we started with $\mathsf{PC}_{\mathsf{DL}}$ or $\mathsf{PC}_{\mathsf{AGM}}$, we get a PCD scheme with different features, as summarized below.

- *From* $\mathsf{PC}_{\mathsf{DL}}$*: PCD based on discrete logarithms.* We obtain a PCD scheme in the *uniform reference string* model (i.e., without secret parameters) and small argument sizes. In contrast, prior PCD schemes require structured reference strings [BCTV14] or have larger argument sizes [COS20]. Moreover, our PCD scheme can be efficiently instantiated from any cycle of elliptic curves [SS11]. In contrast, prior PCD schemes with small argument size use cycles of pairing-friendly elliptic curves [BCTV14; CCW19], which are more expensive.
- *From* $\mathsf{PC}_{\mathsf{AGM}}$*: lightweight PCD based on bilinear groups.* The recursive statement inside this PCD scheme does not involve checking any pairing computations, because pairings are deferred to a verification that occurs *outside* the recursive statement. In contrast, the recursive statements in prior PCD schemes based on pairing-based SNARKs were more expensive because they checked pairing computations [BCTV14].

Note again that our constructions of PCD are *heuristic* as they involve instantiating the random oracle of certain SNARK constructions with an appropriate hash function. This is because Theorem 3 is proven in the random oracle model, but Theorem 1 is explicitly *not* (as is the case for all prior IVC/PCD constructions [Val08; BCCT13; BCTV14; COS20]). There is evidence that this limitation might be inherent [CL20].

Open problem: accumulation in the standard model. All known constructions of accumulation schemes for non-interactive arguments make use of either random oracles (as in our constructions) or knowledge assumptions (e.g., the "trivial" construction from succinct-verifier SNARKs). A natural question, then, is whether there exist constructions of accumulation schemes for non-interactive arguments, or any other interesting predicate, from standard assumptions, or any assumptions which are not known to imply SNARKs. A related question is whether there is a black-box impossibility for accumulation schemes similar to the result for SNARGs of [GW11].

1.2 Related work

Below we survey prior constructions of IVC/PCD.

PCD from SNARKs. Bitansky, Canetti, Chiesa, and Tromer [BCCT13] proved that recursive composition of SNARKs for machine computations implies PCD for constant-depth graphs, and that this in turn implies IVC for polynomial-time machine computations. From the perspective of concrete efficiency, however, one can achieve more efficient recursive composition by using *preprocessing* SNARKs for circuits rather than

SNARKs for machines [BCTV14; COS20]; this observation has led to real-world applications [Co17; BMRS20]. The features of the PCD scheme obtained from recursion depend on the features of the underlying preprocessing SNARK. Below we summarize the features of the two known constructions.

- *PCD from pairing-based SNARKs.* Ben-Sasson, Chiesa, Tromer, and Virza [BCTV14] used pairing-based SNARKs with a special algebraic property to achieve efficient recursive composition with very small argument sizes (linear in the security parameter λ). The use of pairing-based SNARKs has two main downsides. First, they require sampling a *structured reference string* involving secret values ("toxic waste") that, if revealed, compromise security. Second, the verifier performs operations over a finite field that is necessarily different from the field supported "natively" by the statement it is checking. To avoid expensive simulation of field arithmetic, the construction uses *pairing-friendly cycles of elliptic curves*, which severely restricts the choice of field in applications and requires a large base field for security.
- *PCD from IOP-based SNARKs.* Chiesa, Ojha, and Spooner [COS20] used a holographic IOP to construct a preprocessing SNARK that is unconditionally secure in the (quantum) random oracle model, which heuristically implies a post-quantum preprocessing SNARK in the *uniform reference string* model (i.e., without toxic waste). They then proved that any post-quantum SNARK leads to a post-quantum PCD scheme via recursive composition. The downside of this construction is that, given known holographic IOPs, the argument size is larger, currently at $O(\lambda^2 \log^2 N)$ bits for circuits of size N.

IVC from homomorphic encryption. Naor, Paneth, and Rothblum [NPR19] obtain a notion of IVC by using somewhat homomorphic encryption and an information-theoretic object called an "incremental PCP". The key feature of their scheme is that security holds under falsifiable assumptions.

There are two drawbacks, however, that restrict the use of the notion of IVC that their scheme achieves.

First, the computation to be verified must be *deterministic* (this appears necessary for schemes based on falsifiable assumptions given known impossibility results [GW11]). Second, and more subtly, completeness holds only in the case where intermediate proofs were honestly generated. This means that the following attack may be possible: an adversary provides an intermediate proof that verifies, but it is impossible for honest parties to generate new proofs for subsequent computations. Our construction of PCD achieves the stronger condition that completeness holds so long as intermediate proofs verify, ruling out this attack.

Both nondeterministic computation and the stronger completeness notion (achieved by all SNARK-based PCD schemes) are necessary for many of the applications of IVC/PCD.

2 Techniques

2.1 PCD from arguments with accumulation schemes

We summarize the main ideas behind Theorem 1, which obtains proof-carrying data (PCD) from any succinct non-interactive argument of knowledge (SNARK) that has an accumulation scheme. For the sake of exposition, in this section we focus on the special case of IVC, which can be viewed as repeated application of a circuit F. Specifically, we wish to check a claim of the form "$F^T(z_0) = z_T$" where F^T denotes F composed with itself T times.

Prior work: recursion from succinct verification. Recall that in previous approaches to efficient recursive composition [BCTV14; COS20], at each step i we prove a claim of the form "$z_i = F(z_{i-1})$, and there exists a proof π_{i-1} that attests to the correctness of z_{i-1}". This claim is expressed using a circuit R which is the conjunction of F with a circuit representing the SNARK verifier; in particular, the size of the claim is at least the size of the verifier circuit. If the size of the verifier circuit grows linearly (or more) with the size of the claim being checked, then verifying the final proof becomes more costly than the original computation.

For this reason, these works focus on SNARKs with *succinct verification*, where the verifier runs in time *sublinear* in the size of the claim. In this case, the size of the claim essentially *does not grow* with the number of recursive steps, and so checking the final proof costs roughly the same as checking a single step.

Succinct verification is a seemingly paradoxical requirement: the verifier does not even have time to *read* the circuit R. One way to sidestep this issue is *preprocessing*: one designs an algorithm that, at the beginning of the recursion, computes a small cryptographic digest of R, which the recursive verifier can use instead of reading R directly. Because this preprocessing need only be performed once for the given R in an offline phase, it has almost no effect on the performance of each recursive step (in the later online phase).

A new paradigm: IVC from accumulation. Even allowing for preprocessing, succinct verification remains a strong requirement, and there are many SNARKs that are not known to satisfy it (e.g., [BCC+16; BBB+18; AHIV17; BCG+17; BCR+19]). Bowe, Grigg, and Hopwood [BGH19] suggested a further relaxation of succinctness that appears to still suffice for recursive composition: a type of "post-processing". Their observation is as follows: if a SNARK is such that we can efficiently "defer" the verification of a claim in a way that does not grow in cost with the number of claims to be checked, then we can hope to achieve recursive composition by deferring the verification of all claims to the end.

In the remainder of this section, we will give an overview of the proof of Theorem 1, our construction of PCD from SNARKs that have this "post-processing" property. We note that this relaxation of requirements is useful because, as suggested in [BGH19], it leads to new constructions of PCD with desirable properties (see discussion at the end of Section 1.1). In fact, some of these efficiency features are already being exploited by practitioners working on recursing SNARKs [Halo19; Pickles20].

The specific property we require, which we discuss more formally in the next section, is that the SNARK has an *accumulation scheme*. This is a generalization of the idea

described in [BGH19]. Informally, an accumulation scheme consists of three algorithms: an accumulation prover, an accumulation verifier, and a decider. The accumulation prover is tasked with taking an instance-proof pair (z, π) and a previous accumulator acc, and producing a new accumulator $\mathsf{acc}^\star$ that "includes" the new instance. The accumulation verifier, given $((z, \pi), \mathsf{acc}, \mathsf{acc}^\star)$, checks that $\mathsf{acc}^\star$ was computed correctly (i.e., that it accumulates (z, π)) into acc). Finally the decider, given a single accumulator acc, performs a single check that simultaneously ensures that *every* instance-proof pair accumulated in acc verifies.[3]

Given such an accumulation scheme, we can construct IVC as follows. Given a previous instance z_i, proof π_i, and accumulator acc_i, the IVC prover first accumulates (z_i, π_i) with acc_i to obtain a new accumulator acc_{i+1}. The IVC prover also 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. The final IVC proof then consists of (π_T, acc_T). The IVC verifier checks such a proof by running the SNARK verifier on π_T and the accumulation scheme decider on acc_T.

Why does this achieve IVC? Throughout the computation 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. Clearly if this holds at time T then the IVC verifier successfully checks the entire computation. Observe that if we were able to prove that "$z_{i+1} = F(z_i)$, π_i is a valid proof, and acc_i is a valid accumulator", by applying the invariant we would be able to conclude that the computation is correct up to step $i + 1$. Unfortunately we are not able to prove this directly, for two reasons: (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.

From sketch to proof. In the full version of our paper, 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 now informally discuss the properties of our PCD construction.

[3] We remark that the notion of an accumulation scheme is *distinct* from the notion of a cryptographic accumulator for a set (e.g., an RSA accumulator), which provides a succinct representation of a large set while supporting membership queries.

– *Efficiency requirements.* Observe that the statement to be proved includes only the *accumulation verifier*, and so the *only* efficiency requirement for obtaining PCD is that this algorithm run in time sublinear in the size of the circuit R. This implies, in particular, that an accumulator must be of size sublinear in the size of R, and hence must not grow with each accumulation step. The SNARK verifier and the decider algorithm need only be efficient in the usual sense (i.e., polynomial-time).
– *Soundness.* We prove that the PCD scheme is sound provided that the SNARK is knowledge sound (i.e., is an adaptively-secure argument of knowledge) and the accumulation scheme is sound (see Section 2.2 for more on what this means). We stress that in both cases security should be in the standard (CRS) model, without any random oracles (as in prior PCD constructions).
– *Zero knowledge.* We prove that the PCD scheme is zero knowledge, if the underlying SNARK and accumulation scheme are both zero knowledge (for this part we also formulate a suitable notion of zero knowledge for accumulation schemes as discussed shortly in Section 2.2).
– *Post-quantum security.* We also prove that if both the SNARK and accumulation scheme are *post-quantum* secure, then so is the resulting PCD scheme. Here by post-quantum secure we mean that the relevant security properties continue to hold even against polynomial-size *quantum* circuits, as opposed to just polynomial-size *classical* circuits.

2.2 Accumulation schemes

A significant contribution of this work is formulating a general notion of an accumulation scheme. An accumulation scheme for a non-interactive argument as described above is a particular instance of this definition; in subsequent sections we will apply the definition in other settings.

We first give an informal definition that captures the key features of an accumulation scheme. For clarity this is stated for the (minimal) case of a single predicate input q and a single "old" accumulator acc; we later extend this in the natural way to n predicate inputs and m "old" accumulators.

Definition 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 all efficiently-generated accumulators* $\mathsf{acc}, \mathsf{acc}^\star$ *and predicate inputs* $\mathsf{q} \in X$*, 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* $\mathrm{D}(\mathsf{acc}) = 1$.

An accumulation scheme for a SNARK is an accumulation scheme for the predicate induced by the argument verifier; in this case the predicate input q consists of an instance-proof pair $(\mathbb{x}, \pi)$. Note that the completeness requirement does not place any restriction on how the previous accumulator acc is generated; we require that completeness holds

for any acc the decider D determines to be valid, and any q for which the predicate Φ holds. This is needed to obtain a similarly strong notion of completeness for PCD, required for applications where accumulation is done by multiple parties that do not trust one another.

Zero knowledge. For our PCD application, the notion of zero knowledge for an accumulation scheme that we use is the following: one can sample a "fake" accumulator that is indistinguishable from a real accumulator $\mathsf{acc}^\star$, *without knowing anything* about the old accumulator acc and predicate input q that were accumulated in $\mathsf{acc}^\star$. The existence of the accumulation verifier V complicates matters here: if the adversary knows acc and q, then it is easy to distinguish a real accumulator from a fake one using V. We resolve this issue by modifying Definition 1 to have the accumulation prover P produce a *verification proof* π_{V} in addition to the new accumulator $\mathsf{acc}^\star$. Then V uses π_{V} in verifying the accumulator, but π_{V} is *not* required for subsequent accumulation. In our application, the simulator then does *not* have to simulate π_{V}. This avoids the problem described: even if the adversary knows acc and q, unless π_{V} is correct, V can simply reject, as it would for a "fake" accumulator. Our informal definition is as follows.

Definition 2. *An accumulation scheme for Φ is* **zero knowledge** *if there exists an efficient simulator S such that for all accumulators acc and inputs $\mathsf{q} \in X$ such that $\mathrm{D}(\mathsf{acc}) = 1$ and $\Phi(\mathsf{q}) = 1$, the distribution of $\mathsf{acc}^\star$ when $(\mathsf{acc}^\star, \pi_{\mathrm{V}}) \leftarrow \mathrm{P}(\mathsf{acc}, \mathsf{q})$ is computationally indistinguishable from $\mathsf{acc}^\star \leftarrow \mathrm{S}(1^\lambda)$.*

Predicate specification. The above informal definitions omit many important details; we now highlight some of these. Suppose that, as required for IVC/PCD, we have some fixed circuit R for which we want to accumulate pairs $(\mathbb{x}_i, \pi_i)$, where π_i is a SNARK proof that there exists $\mathbb{w}_i$ such that $R(\mathbb{x}_i, \mathbb{w}_i) = 1$. In this case the predicate corresponding to the verifier depends not only on the pair $(\mathbb{x}_i, \pi_i)$, but also on the circuit R, as well as the public parameters of the argument scheme pp and (often) a random oracle ρ.

Moreover, each of these inputs has different security and efficiency considerations. The security of the SNARK (and the accumulation scheme) can only be guaranteed with high probability over public parameters drawn by the generator algorithm of the SNARK, and over the random oracle. The circuit R may be chosen adversarially, but cannot be part of the input q because it is too large; it must be fixed at the beginning.

These considerations lead us to define an accumulation scheme with respect to both a predicate $\Phi\colon \mathcal{U}(*) \times (\{0,1\}^*)^3 \to \{0,1\}$ and a *predicate-specification algorithm* $\mathcal{H}$. We then adapt Definition 1 to hold for the predicate $\Phi(\rho, \mathsf{pp}_\Phi, \mathsf{i}_\Phi, \cdot)$ where ρ is a random oracle, pp_Φ is output by $\mathcal{H}$, and i_Φ is chosen adversarially. In our SNARK example, $\mathcal{H}$ is equal to the SNARK generator, i_Φ is the circuit R, and $\Phi(\rho, \mathsf{pp}, R, (\mathbb{x}, \pi)) = \mathcal{V}^\rho(\mathsf{pp}, R, \mathbb{x}, \pi)$.

Remark 1 (helped verification). We compare accumulation schemes for SNARKs with the notion of "helped verification" [MBKM19]. In a SNARK with helped verification, an untrusted party known as the *helper* can, given n proofs, produce an auxiliary proof that enables checking the n proofs at lower cost than that of checking each proof individually. This batching capability can be viewed as a special case of accumulation, as it applies to

n "fresh" proofs only; there is no notion of batching "old" accumulators. It is unclear whether the weaker notion of helped verification alone suffices to construct IVC/PCD schemes.

2.3 Constructing arguments with accumulation schemes

A key ingredient in our construction of PCD is a SNARK that has an accumulation scheme (see Section 2.1). Below we summarize the ideas behind Theorem 2, by explaining how to construct accumulation schemes for SNARKs whose verifier is succinct relative to an oracle predicate $\Phi_\circ$ that itself has an accumulation scheme.

Predicate-efficient SNARKs. We call a SNARK ARG *predicate-efficient* with respect to a predicate $\Phi_\circ$ if its verifier $\mathcal{V}$ operates as follows: (i) run a fast "inner" verifier $\mathcal{V}_{\mathsf{pe}}$ to produce a bit b and query set Q; (ii) accept iff $b = 1$ and for all $\mathsf{q} \in Q$, $\Phi_\circ(\mathsf{q}) = 1$. In essence, $\mathcal{V}$ can be viewed as a circuit with "oracle gates" for $\Phi_\circ$.[4] The aim is for $\mathcal{V}_{\mathsf{pe}}$ to be significantly more efficient than $\mathcal{V}$; that is, the queries to $\Phi_\circ$ capture the "expensive" part of the computation of $\mathcal{V}$.

As noted in Section 1.1, one can view recent SNARK constructions [MBKM19; GWC19; CHM+20] as being predicate-efficient with respect to a "polynomial commitment" predicate. We discuss how to construct accumulation schemes for these predicates below in Section 2.4.

Accumulation scheme for predicate-efficient SNARKs. Let ARG be a SNARK that is predicate-efficient with respect to a predicate $\Phi_\circ$, and let $\mathsf{AS}_\circ$ be an accumulation scheme for $\Phi_\circ$. To check n proofs, instead of directly invoking the SNARK verifier $\mathcal{V}$, we can first run $\mathcal{V}_{\mathsf{pe}}$ n times to generate n query sets for $\Phi_\circ$, and then, instead of invoking $\Phi_\circ$ on each of these sets, we can accumulate these queries using $\mathsf{AS}_\circ$. Below we sketch the construction of an accumulation scheme $\mathsf{AS}_{\mathsf{ARG}}$ for ARG based on this idea.

To accumulate n instance-proof pairs $[(\mathbb{x}_i, \pi_i)]_{i=1}^n$ starting from an old accumulator acc, the accumulation prover $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{P}$ first invokes the inner verifier $\mathcal{V}_{\mathsf{pe}}$ on each $(\mathbb{x}_i, \pi_i)$ to generate a query set Q_i for $\Phi_\circ$, accumulates their union $Q = \cup_{i=1}^n Q_i$ into acc using $\mathsf{AS}_\circ.\mathrm{P}$, and finally outputs the resulting accumulator $\mathsf{acc}^\star$. To check that $\mathsf{acc}^\star$ indeed accumulates $[(\mathbb{x}_i, \pi_i)]_{i=1}^n$ into acc, the accumulation verifier $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{V}$ first checks, for each i, whether the inner verifier $\mathcal{V}_{\mathsf{pe}}$ accepts $(\mathbb{x}_i, \pi_i)$, and then invokes $\mathsf{AS}_\circ.\mathrm{V}$ to check whether $\mathsf{acc}^\star$ correctly accumulates the query set $Q = \cup_{i=1}^n Q_i$. Finally, to decide whether $\mathsf{acc}^\star$ is a valid accumulator, the accumulation scheme decider $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{D}$ simply invokes $\mathsf{AS}_\circ.\mathrm{D}$.

From sketch to proof. The foregoing sketch omits details required to construct a scheme that satisfies the "full" definition of accumulation schemes as stated in the full version of our paper. For instance, as noted in Section 2.3, the predicate $\Phi_\circ$ may be be an oracle predicate, and could depend on the public parameters of the SNARK ARG. We handle this by requiring that the accumulation scheme for $\Phi_\circ$ uses the SNARK generator $\mathcal{G}$ as its predicate specification algorithm. We also show that zero knowledge and post-quantum security are preserved. See the full version of our paper for a formal treatment of these issues, along with security proofs.

[4] This is not precisely the case, because the verifier is required to reject immediately if it ever makes a query q with $\Phi_\circ(\mathsf{q}) = 0$.

From predicate-efficient SNARKs to PCD. In order to build an accumulation scheme $\mathsf{AS}_{\mathsf{ARG}}$ that suffices for PCD, ARG and $\mathsf{AS}_{\circ}$ must satisfy certain efficiency properties. In particular, when verifying satisfiability for a circuit of size N, the running time of $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{V}$ must be sublinear in N, which means in turn that the running times of $\mathcal{V}_{\mathsf{pe}}$ and $\mathsf{AS}_{\circ}.\mathrm{V}$, as well as the size of the query set Q, must be sublinear in N. Crucially, however, $\mathsf{AS}_{\circ}.\mathrm{D}$ need only run in time polynomial in N.

2.4 Accumulation schemes for polynomial commitments

As noted in Section 2.3, several SNARK constructions (e.g., [MBKM19; GWC19; CHM+20]) are predicate-efficient with respect to an underlying *polynomial commitment*, which means that constructing an accumulation scheme for the latter leads (via Theorem 2) to an accumulation scheme for the whole SNARK.

Informally, a polynomial commitment scheme (PC 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. An accumulation scheme for a PC scheme thus accumulates claims of the form "C commits to p such that $p(z) = v$" for arbitrary polynomials p and evaluation points z.

In this section, we explain the ideas behind Theorem 3, by sketching how to construct (zero knowledge) accumulation schemes for two popular (hiding) polynomial commitment schemes.

- In Section 2.4.1, we sketch our accumulation scheme for $\mathsf{PC}_{\mathsf{DL}}$, a polynomial commitment scheme derived from [BCC+16; BBB+18; WTS+18] that is based on the hardness of discrete logarithms.
- In Section 2.4.2, we sketch our accumulation scheme for $\mathsf{PC}_{\mathsf{AGM}}$, a polynomial commitment scheme based on knowledge assumptions over bilinear groups [KZG10; CHM+20].

In each case, the running time of the accumulation verifier will be sublinear in the degree of the polynomial, and the accumulator itself will not grow with the number of accumulation steps. This allows the schemes to be used, in conjunction with a suitable predicate-efficient SNARK, to construct PCD.

We remark that each of our accumulation schemes is proved secure in the random oracle model by invoking a useful lemma about "zero-finding games" for committed polynomials. Security also requires that the random oracle used for an accumulation scheme for a PC scheme is domain-separated from the random oracle used by the PC scheme itself. See the full version for details.

2.4.1 Accumulation scheme for $\mathsf{PC}_{\mathsf{DL}}$

We sketch our accumulation scheme for $\mathsf{PC}_{\mathsf{DL}}$. For univariate polynomials of degree less than d, $\mathsf{PC}_{\mathsf{DL}}$ achieves evaluation proofs of size $O(\lambda \log d)$ in the random oracle model, and assuming the hardness of the discrete logarithm problem in a prime order group $\mathbb{G}$. In particular, there are no secret parameters (so-called "toxic waste"). However, $\mathsf{PC}_{\mathsf{DL}}$ has poor verification complexity: checking an evaluation proof requires $\Omega(d)$ scalar multiplications in $\mathbb{G}$. Bowe, Grigg, and Hopwood [BGH19] suggested a way to amortize this cost across a batch of n proofs. Below we show that their idea leads to an

accumulation scheme for $\mathsf{PC}_{\mathsf{DL}}$ with an accumulation verifier that uses only $O(n \log d)$ scalar multiplications instead of the naive $\Theta(n \cdot d)$, and with an accumulator of size $O(\log d)$ elements in $\mathbb{G}$.

Summary of $\mathsf{PC}_{\mathsf{DL}}$. The committer and receiver both sample (consistently via the random oracle) a list of group elements $\{G_0, G_1, \ldots, G_d\} \in \mathbb{G}^{d+1}$ in a group $\mathbb{G}$ of prime order q (written additively). A commitment to a polynomial $p(X) = \sum_{i=0}^{d} a_i X^i \in \mathbb{F}_q^{\leq d}[X]$ is then given by $C := \sum_{i=0}^{d} a_i G_i$. To prove that the committed polynomial p evaluates to v at a given point $z \in \mathbb{F}_q$, it suffices to prove that the triple (C, z, v) satisfies the following NP statement:

$$\exists\, a_0, \ldots, a_d \in \mathbb{F} \text{ s.t. } v = \textstyle\sum_{i=0}^{d} a_i z^i \text{ and } C = \sum_{i=0}^{d} a_i G_i \ .$$

This is a special case of an *inner product argument* (IPA), as defined in [BCC+16], which proves the inner product of two committed vectors. The receiver simply verifies this inner product argument to check the evaluation. The fact that the vector $(1, z, \ldots, z^d)$ is known to the verifier and has a certain structure is exploited in the accumulation scheme that we describe below.

Accumulation scheme for the IPA. Our accumulation scheme relies on a special structure of the IPA verifier: it generates $O(\log d)$ challenges using the random oracle, then performs cheap checks requiring $O(\log d)$ field and group operations, and finally performs an expensive check requiring $\Omega(d)$ scalar multiplications. This latter check asserts consistency between the challenges and a group element U contained in the proof. Hence, the IPA verifier is succinct *barring the expensive check*, and so constructing an accumulation scheme for the IPA reduces to the task of constructing an accumulation scheme for the expensive check involving U.

To do this, we rely on an idea of Bowe, Grigg, and Hopwood [BGH19], which itself builds on an observation in [BBB+18]. Namely, letting $(\xi_1, \ldots, \xi_{\log_2 d})$ be the protocol's challenges, U can be viewed as a commitment to the polynomial $h(X) := \prod_{i=0}^{\log_2(d)-1} (1 + \xi_{\log_2(d)-i} X^{2^i}) \in \mathbb{F}_q^{\leq d}[X]$. This polynomial has the special property that it can be evaluated at any point in just $O(\log d)$ field operations (exponentially smaller than its degree d). This allows transforming the expensive check on U into a check that is amenable to batching: instead of directly checking that U is a commitment to h, one can instead check that the polynomial committed inside U agrees with h at a challenge point z sampled via the random oracle.

We leverage this idea as follows. When accumulating evaluation claims about multiple polynomials $p_1, \ldots, p_n$, applying the foregoing transformation results in n checks of the form "check that the polynomial contained in U_i evaluates to $h_i(z)$ at the point z". Because these are all claims for the correct evaluation of the polynomials h_i at *the same point* z, we can accumulate them via standard homomorphic techniques. We now summarize how we apply this idea to construct our accumulation scheme $\mathsf{AS} = (\mathrm{P}, \mathrm{V}, \mathrm{D})$ for $\mathsf{PC}_{\mathsf{DL}}$.

Accumulators in our accumulation scheme have the same form as the instances to be accumulated: they are tuples of the form (C, z, v, π) where π is an evaluation proof for the claim "$p(z) = v$" and p is the polynomial committed in C. For simplicity, below we consider the case of accumulating one old accumulator $\mathsf{acc} = (C_1, z_1, v_1, \pi_1)$ and one instance (C_2, z_2, v_2, π_2) into a new accumulator $\mathsf{acc}^\star = (C, z, v, \pi)$.

Accumulation prover P*:* compute the new accumulator $\mathsf{acc}^\star = (C, z, v, \pi)$ from the old accumulator $\mathsf{acc} = (C_1, z_1, v_1, \pi_1)$ and the instance (C_2, z_2, v_2, π_2) as follows.

- Compute U_1, U_2 from π_1, π_2 respectively. As described above, these elements can be viewed as commitments to polynomials h_1, h_2 defined by the challenges derived from π_1, π_2.
- Use the random oracle ρ to compute the random challenge $\alpha := \rho([(h_1, U_1), (h_2, U_2)])$.
- Compute $C := U_1 + \alpha U_2$, which is a polynomial commitment to $p(X) := h_1(X) + \alpha h_2(X)$.
- Compute the challenge point $z := \rho(C, p)$, where p is uniquely represented via the tuple $([h_1, h_2], \alpha)$.
- Construct an evaluation proof π for the claim "$p(z) = v$". (This step is the only expensive one.)
- Output the new accumulator $\mathsf{acc}^\star := (C, z, v, \pi)$.

Accumulation verifier V*:* to check that the new accumulator $\mathsf{acc}^\star = (C, z, v, \pi)$ was correctly generated from the old accumulator $\mathsf{acc} = (C_1, z_1, v_1, \pi_1)$ and the instance (C_2, z_2, v_2, π_2), first compute the challenges α and z from the random oracle as above, and then check that (a) (C_1, z_1, v_1, π_1) and (C_2, z_2, v_2, π_2) pass the cheap checks of the IPA verifier, (b) $C = U_1 + \alpha U_2$, and (c) $h_1(z) + \alpha h_2(z) = v$.

Decider D*:* on input the (final) accumulator $\mathsf{acc}^\star = (C, z, v, \pi)$, check that π is a valid evaluation proof for the claim that the polynomial committed inside C evaluates to v at the point z.

This construction achieves the efficiency summarized in Theorem 3.

We additionally achieve zero knowledge accumulation for the hiding variant of $\mathsf{PC}_{\mathsf{DL}}$. Informally, the accumulation prover randomizes $\mathsf{acc}^\star$ by including a new random polynomial h_0 in the accumulation step. This ensures that the evaluation claim in $\mathsf{acc}^\star$ is for a random polynomial, thus hiding all information about the original evaluation claims. To allow the accumulation verifier to check that this randomization was performed correctly, the prover includes h_0 in an auxiliary proof π_{V}.

In the full version, we show how to extend the above accumulation scheme to accumulate any number of old accumulators and instances. Our security proof for the resulting accumulation scheme relies on the hardness of zero-finding games, and the security of $\mathsf{PC}_{\mathsf{DL}}$.

2.4.2 Accumulation scheme for $\mathsf{PC}_{\mathsf{AGM}}$

We sketch our accumulation scheme $\mathsf{AS} = (\mathrm{P}, \mathrm{V}, \mathrm{D})$ for $\mathsf{PC}_{\mathsf{AGM}}$. Checking an evaluation proof in $\mathsf{PC}_{\mathsf{AGM}}$ requires 1 pairing, and so checking n evaluation proofs requires n pairings. AS improves upon this as follows: the accumulation verifier V only performs $O(n)$ scalar multiplications in $\mathbb{G}_1$ in order to check the accumulation of n evaluation proofs, while the decider D performs only a single pairing in order to check the resulting accumulator. This is much cheaper: it reduces the number of pairings from n to 1, and also defers this single pairing to the end of the accumulation (the decider). In particular, when instantiating the PCD construction outlined in Section 2.1 with a $\mathsf{PC}_{\mathsf{AGM}}$-based SNARK and our accumulation scheme for $\mathsf{PC}_{\mathsf{AGM}}$, we can eliminate *all* pairings from the circuit being verified in the PCD construction.

Below we explain how standard techniques for batching pairings using random linear combinations [CHM+20] allow us to realize an accumulation scheme for $\mathsf{PC}_{\mathsf{AGM}}$ with these desirable properties.

Summary of $\mathsf{PC}_{\mathsf{AGM}}$. The committer key ck and receiver key rk for a given maximum degree bound D are group elements from a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, G, H, e)$: $\mathsf{ck} := \{G, \beta G, \ldots, \beta^D G\} \in \mathbb{G}_1^{D+1}$ consists of group elements encoding powers of a random field element β, while $\mathsf{rk} := (G, H, \beta H) \in \mathbb{G}_1 \times \mathbb{G}_2^2$.

A commitment to a polynomial $p \in \mathbb{F}_q^{\leq D}[X]$ is the group element $C := p(\beta)G \in \mathbb{G}_1$. To prove that p evaluates to v at a given point $z \in \mathbb{F}_q$, the sender computes a "witness polynomial" $w(X) := (p(X) - v)/(X - z)$, and outputs the evaluation proof $\pi := w(\beta)G \in \mathbb{G}_1$. The receiver can check this proof by checking the pairing equation $e(C - vG, H) = e(\pi, \beta H - zH)$. This pairing equation is the focus of our accumulation scheme below. (This summary omits details about degree enforcement and about hiding.)

Accumulation scheme. We construct an accumulation scheme $\mathsf{AS} = (\mathrm{P}, \mathrm{V}, \mathrm{D})$ for $\mathsf{PC}_{\mathsf{AGM}}$ by relying on standard techniques for batching pairing equations. Suppose that we wish to simultaneously check the validity of n instances $[(C_i, z_i, v_i, \pi_i)]_{i=1}^n$. First, rewrite the pairing check for the i-th instance as follows:

$$e(C_i - v_i G, H) = e(\pi_i, \beta H - z_i H) \iff e(C_i - v_i G + z_i \pi_i, H) = e(\pi_i, \beta H) \ . \quad (1)$$

After the rewrite, the $\mathbb{G}_2$ inputs to both pairings do not depend on the claim being checked. This allows batching the pairing checks by taking a random linear combination with respect to a random challenge $r := \rho([C_i, z_i, v_i, \pi_i]_{i=1}^n)$ computed from the random oracle, resulting in the following combined equation:

$$e(\textstyle\sum_{i=1}^n r^i (C_i - v_i G + z_i \pi_i), H) = e(\sum_{i=1}^n r^i \pi_i, \beta H) \ . \quad (2)$$

We now have a pairing equation involving an "accumulated commitment" $C^\star := \sum_{i=1}^n r^i (C_i - v_i G + z_i \pi_i)$ and an "accumulated proof" $\pi^\star := \sum_{i=1}^n r^i \pi_i$. This observation leads to the accumulation scheme below.

An accumulator in AS consists of a commitment-proof pair $(C^\star, \pi^\star)$, which the decider D validates by checking that $e(C^\star, H) = e(\pi^\star, \beta H)$. Moreover, observe that by Eq. (1), checking the validity of a claimed evaluation (C, z, v, π) within $\mathsf{PC}_{\mathsf{AGM}}$ corresponds to checking that the "accumulator" $(C - vG + z\pi, \pi)$ is accepted by the decider D. Thus we can restrict our discussion to accumulating *accumulators*.

The accumulation prover P, on input a list of old accumulators $[\mathsf{acc}_i]_{i=1}^n = [(C_i^\star, \pi_i^\star)]_{i=1}^n$, computes a random challenge $r := \rho([\mathsf{acc}_i]_{i=1}^n)$, constructs $C^\star := \sum_{i=1}^n r^i C_i^\star$ and $\pi^\star := \sum_{i=1}^n r^i \pi_i^\star$, and outputs the new accumulator $\mathsf{acc}^\star := (C^\star, \pi^\star) \in \mathbb{G}_1^2$. To check that $\mathsf{acc}^\star$ accumulates $[\mathsf{acc}_i]_{i=1}^n$, the accumulation verifier V simply invokes P and checks that its output matches the claimed new accumulator $\mathsf{acc}^\star$.

To achieve zero knowledge accumulation, the accumulation prover randomizes $\mathsf{acc}^\star$ by including in it an extra "old" accumulator corresponding to a random polynomial, which statistically hides the accumulated claims. To allow the accumulation verifier to check that this randomization was performed correctly, the prover includes this old accumulator in an auxiliary proof π_{V}.

This construction achieves the efficiency summarized in Theorem 3.

In the full version of our paper, we show how to extend the above accumulation scheme to account for additional features of $\mathsf{PC}_{\mathsf{AGM}}$ (degree enforcement and hiding). Our security proof for the resulting accumulation scheme relies on the hardness of zero-finding games (see full version).

Acknowledgements

The authors thank William Lin for pointing out an error in a prior version of the construction of $\mathsf{PC}_{\mathsf{DL}}$, and Github user 3for for pointing out errors in a prior version of the construction of $\mathsf{PC}_{\mathsf{AGM}}$. This research was supported in part by: the Berkeley Haas Blockchain Initiative and a donation from the Ethereum Foundation. Benedikt Bünz performed part of the work while visiting the Simons Institute for the Theory of Computing.

References

[AHIV17] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup". In: CCS '17.

[BBB+18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: S&P '18.

[BCC+16] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting". In: EUROCRYPT '16.

[BCCT13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. "Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data". In: STOC '13.

[BCG+17] J. Bootle, A. Cerulli, E. Ghadafi, J. Groth, M. Hajiabadi, and S. K. Jakobsen. "Linear-Time Zero-Knowledge Proofs for Arithmetic Circuit Satisfiability". In: ASIACRYPT '17.

[BCR+19] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. "Aurora: Transparent Succinct Arguments for R1CS". In: EUROCRYPT '19.

[BCTV14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. "Scalable Zero Knowledge via Cycles of Elliptic Curves". In: CRYPTO '14.

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

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

[CCW19] A. Chiesa, L. Chua, and M. Weidner. "On Cycles of Pairing-Friendly Elliptic Curves". In: *SIAM Journal on Applied Algebra and Geometry* (2019).

[CHM+20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: EUROCRYPT '20.

[CL20] A. Chiesa and S. Liu. "On the Impossibility of Probabilistic Proofs in Relativized Worlds". In: ITCS '20.

[Co17] O(1) Labs. "Coda Cryptocurrency". `https://codaprotocol.com/`.

[COS20] A. Chiesa, D. Ojha, and N. Spooner. "Fractal: Post-Quantum and Transparent Recursive Proofs from Holography". In: EUROCRYPT '20.

[CT10] A. Chiesa and E. Tromer. "Proof-Carrying Data and Hearsay Arguments from Signature Cards". In: ICS '10.

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

[CTV15] A. Chiesa, E. Tromer, and M. Virza. “Cluster Computing in Zero Knowledge”. In: EUROCRYPT ’15.

[GW11] C. Gentry and D. Wichs. “Separating Succinct Non-Interactive Arguments From All Falsifiable Assumptions”. In: STOC ’11.

[GWC19] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. “PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge”. ePrint Report 2019/953.

[Halo19] S. Bowe, J. Grigg, and D. Hopwood. *Halo*. 2019. URL: `https://github.com/ebfull/halo`.

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

[KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. “Constant-Size Commitments to Polynomials and Their Applications”. In: ASIACRYPT ’10.

[MBKM19] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. “Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updateable Structured Reference Strings”. In: CCS ’19.

[NPR19] M. Naor, O. Paneth, and G. N. Rothblum. “Incrementally Verifiable Computation via Incremental PCPs”. In: TCC ’19.

[NT16] A. Naveh and E. Tromer. “PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations”. In: S&P ’16.

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

[SS11] J. H. Silverman and K. E. Stange. “Amicable Pairs and Aliquot Cycles for Elliptic Curves”. In: *Experimental Mathematics* (2011).

[Val08] P. Valiant. “Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency”. In: TCC ’08.

[WTS+18] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. “Doubly-Efficient zkSNARKs Without Trusted Setup”. In: S&P ’18.