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

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.
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 arguments) that has a *split accumulation scheme*, which is a weak form of accumulation that we introduce.
Moreover, we construct a transparent non-interactive argument of knowledge for R1CS whose split accumulation is verifiable via a (small) *constant number of group and field operations*. Our construction is proved secure in the random oracle model based on the hardness of discrete logarithms, and it leads, via the random oracle heuristic and our result above, to concrete efficiency improvements for PCD.
Along the way, we construct a split accumulation scheme for Hadamard products under Pedersen commitmentsand 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

1 Introduction

Proof-carrying data (PCD) [CT10] is a powerful cryptographic primitive that enables mutually distrustful parties to perform distributed computations that run indefinitely, while ensuring that the correctness of every intermediate state of the computation can be verified efficiently. 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

⋆ The full version of this paper is available online[BCL+20].

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 is achievable 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], which in turn led to 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 has constant size (improving over the logarithmic-size verifier of prior accumulation schemes in this setting). (4) We implement and evaluate constructions from this paper and from [BCMS20].

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+1} from the input q_i and the old accumulator acc_i; the verifier can check that the triple $(\mathsf{q}_i, \mathsf{acc}_i, \mathsf{acc}_{i+1})$ is a valid accumulation step, much more efficiently than running Φ on q_i. At any time, the decider can validate acc_{i+1}, 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 $\mathsf{qx}_1, \mathsf{qx}_2, \ldots$ the goal is to establish that there exist witnesses $\mathsf{qw}_1, \mathsf{qw}_2, \ldots$ such that $\Phi(\mathsf{qx}_i, \mathsf{qw}_i) = 1$ 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 pf). We refer to this notion as *split accumulation for relations*, and refer to (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.

See Section 2.1 for more on atomic vs. split accumulation, and the full version for formal definitions.

(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 where the *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. Similarly to [BCMS20], if the NARK and accumulation scheme are post-quantum secure, so is the PCD scheme. (It remains an open question whether there are non-trivial post-quantum instantiations of these.)

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. Additionally, if the NARK and its accumulation scheme are post-quantum secure then the PCD scheme is also post-quantum secure.*

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 the full version for technical details.

(3) NARK with split accumulation based on DL. 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 only component that is not used as a black box, and thus typically determines concrete efficiency. Towards this end, we present a (zero knowledge) NARK with (zero knowledge) 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 (zero knowledge) NARK for R1CS and a corresponding (zero knowledge) 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*
$O(\mathsf{M})\ \mathbb{G}$	$O(\mathsf{M})\ \mathbb{G}$	$O(1)\ \mathbb{G}$	$O(\mathsf{M})\ \mathbb{G}$	$O(1)\ \mathbb{G}$	$O(\mathsf{M})\ \mathbb{G}$	$\lvert\mathsf{acc}.\mathbb{x}\rvert = O(1)\ \mathbb{G} + O(1)\ \mathbb{F}$
$O(\mathsf{M})\ \mathbb{F}$	$O(\mathsf{M})\ \mathbb{F}$	$O(\mathsf{M})\ \mathbb{F}$	$O(\mathsf{M})\ \mathbb{F}$	$O(1)\ \mathbb{F}$	$O(\mathsf{M})\ \mathbb{F}$	$\lvert\mathsf{acc}.\mathbb{w}\rvert = O(\mathsf{M})\ \mathbb{F}$

Above, M *denotes 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 $O(\log \mathsf{M})$ group elements, *much less* than the NARK's $O(\mathsf{M})$ field elements;
- the SNARK's accumulator verifier uses $O(\log \mathsf{M})$ 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]. (In fact, as our construction does not rely on any number-theoretic properties of $|\mathbb{G}|$, we could even use the $(\texttt{secp256k1}, \texttt{secq256k1})$ cycle, where `secp256k1` is the curve used in Bitcoin.) 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.

See Section 2.3 for a summary of the ideas behind Theorem 1, and the full version for technical details.

(4) Split accumulation for common predicates. We obtain split accumulation schemes with constant-size accumulation verifiers for common predicates: (i) *Hadamard products (and more generally any bilinear function) under Pedersen commitments* (see Section 2.5 for a summary and the full version for details); (ii) *polynomial evaluations under Pedersen commitments* (see Section 2.6 for a summary and the full version for technical details). Split accumulation for Hadamard products is a building block that we use to prove Theorem 1.

(5) Implementation and evaluation. We contribute a set of Rust libraries[1] 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) split accumulation for our zkNARK for R1CS; (d) split accumulation for Hadamard products under Pedersen commitments; (e) split accumulation for polynomial evaluations under Pedersen commitments; (f) atomic accumulation for polynomial commitments based on inner product arguments and pairings from [BCMS20]; (g) constraints for all the foregoing 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. Our experiments focus on determining the *recursion threshold*, which informally is the number of constraints that need to be proved at each step of the recursion. Our evaluation demonstrates that, over curves from the popular "Pasta" cycle [Hop20], the recursion threshold for split accumulation of our NARK for R1CS is as low as $52{,}000$ constraints, which is at least $8.5\times$ cheaper than the cost of IVC constructed from atomic accumulation for discrete-logarithm-based protocols [BCMS20]. In fact, the recursion threshold is even lower than that for IVC constructed from prior state-of-the-art pairing-friendly SNARKs [Gro16]. While this comes at the expense of much larger proof sizes, this overhead is attractive for notable applications (e.g., incrementally-verifiable ledgers).

[1] `https://github.com/arkworks-rs/accumulation`

See the full version for more details on our implementation and evaluation, respectively.

Remark 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 study 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 2.6 and detailed in the full version). 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 based on 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 a 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. In particular, contrary to what is claimed as a theorem in [BDFG20], it is *not* known how to build PCD from a PC scheme with an aggregation/accumulation scheme in any model without making additional heuristic assumptions. This is because obtaining a NARK from a PC scheme using known techniques requires the use of a random oracle, which we do not know how to accumulate. In contrast, we construct PCD in the standard model starting directly from an aggregation/accumulation scheme *for a NARK*, and *no additional assumptions*. Separately, the security of our accumulation scheme for a NARK in the standard model *is* an assumption, which is conjectured based on a security proof 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 our new notion of split accumulation for relation predicates, and compare 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 on a new result about split accumulation for Hadamard products, which we discuss in Section 2.5. Then, in Section 2.6, we discuss our split accumulation for a Pedersen-based polynomial commitment, which can act as a drop-in replacement for polynomial commitments used in prior SNARKs, such as those of [BGH19]. Finally, in Section 2.7 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.

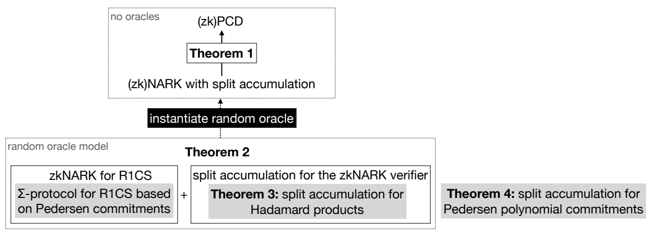
no oracles
(zk)PCD
Theorem 1
(zk)NARK with split accumulation
instantiate random oracle
random oracle model
Theorem 2
zkNARK for R1CS
Σ-protocol for R1CS based on Pedersen commitments
+
split accumulation for the zkNARK verifier
Theorem 3: split accumulation for Hadamard products
Theorem 4: split accumulation for Pedersen polynomial commitments

Fig. 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 introduce, which we call *split accumulation*.

Atomic accumulation for languages. An *accumulation scheme for a language predicate* $\Phi \colon X \to \{0,1\}$ is 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 running the predicate Φ on each input.

This is done as follows. Starting from an initial ("empty") accumulator acc_1, the prover is used to accumulate the first input q_1 to produce a new accumulator $\mathsf{acc}_2 \leftarrow \mathrm{P}(\mathsf{q}_1, \mathsf{acc}_1)$; then the prover is used again to accumulate the second input q_2 to produce a new accumulator $\mathsf{acc}_3 \leftarrow \mathrm{P}(\mathsf{q}_2, \mathsf{acc}_2)$; 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: $\mathrm{V}(\mathsf{q}_1, \mathsf{acc}_1, \mathsf{acc}_2) = 1, \mathrm{V}(\mathsf{q}_2, \mathsf{acc}_2, \mathsf{acc}_3) = 1$, 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, which additionally includes an accumulation proof used to check an accumulation step (but is not passed on).

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 every accumulator* acc *and predicate input* $\mathsf{q} \in X$*, if* $\mathrm{D}(\mathsf{acc}) = 1$ *and* $\Phi(\mathsf{q}) = 1$*, then for* $(\mathsf{acc}^\star, \mathsf{pf}^\star) \leftarrow \mathrm{P}(\mathsf{acc}, \mathsf{q})$ *it holds that* $\mathrm{V}(\mathsf{q}, \mathsf{acc}, \mathsf{acc}^\star, \mathsf{pf}^\star) = 1$ *and* $\mathrm{D}(\mathsf{acc}^\star) = 1$.
- Soundness: *For every efficiently-generated old accumulator* acc*, predicate input* $\mathsf{q} \in X$*, new accumulator* $\mathsf{acc}^\star$*, and accumulation proof* $\mathsf{pf}^\star$*, if* $\mathrm{D}(\mathsf{acc}^\star) = 1$ *and* $\mathrm{V}(\mathsf{q}, \mathsf{acc}, \mathsf{acc}^\star, \mathsf{pf}^\star) = 1$ *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 accumulators $[\mathsf{acc}_j]_{j=1}^m$ and multiple predicate inputs $[\mathsf{q}_i]_{i=1}^n$ in one step, the optional property of zero knowledge (enabled by the accumulation proof $\mathsf{pf}^\star$), the fact that $\mathrm{P}, \mathrm{V}, \mathrm{D}$ should receive keys $\mathsf{apk}, \mathsf{avk}, \mathsf{dk}$ generated by an indexer algorithm that receives the specification of Φ, and others. 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 pf 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, in this paper we consider a relaxation where this is not the case, and will enable us to obtain new instantiations that lead to new theoretical and practical results.

Split accumulation for relations. We propose a relaxed notion of accumulation: 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. Split accumulation differs from atomic accumulation in that: (a) an input to Φ consists of a short instance part qx and a (possibly) long witness part qw; (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 predicate input $\mathsf{q}_i = (\mathsf{qx}_i, \mathsf{qw}_i)$ into a prior accumulator acc_i to obtain a new accumulator and validity proof $(\mathsf{acc}_{i+1}, \mathsf{pf}_{i+1}) \leftarrow \mathrm{P}(\mathsf{q}_i, \mathsf{acc}_i)$. Different from before, however, we wish to establish that given instances $\mathsf{qx}_1, \ldots, \mathsf{qx}_T$ there exist (more precisely, a party knows) witnesses $\mathsf{qw}_1, \ldots, \mathsf{qw}_T$ such that $\Phi(\mathsf{qx}_1, \mathsf{qw}_1) \wedge \cdots \wedge \Phi(\mathsf{qx}_T, \mathsf{qw}_T) = 1$. For this it suffices to check that:

- the verifier accepts each accumulation step given only the short instance parts: $\mathrm{V}(\mathsf{qx}_1, \mathsf{acc}_1.\mathbb{x}, \mathsf{acc}_2.\mathbb{x}, \mathsf{pf}_2) = 1$, $\mathrm{V}(\mathsf{qx}_2, \mathsf{acc}_2.\mathbb{x}, \mathsf{acc}_3.\mathbb{x}, \mathsf{pf}_3) = 1$, 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 of 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.[2] Informally the security definition could be summarized as follows.

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

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] By "working definition" we mean a definition that we can provably fulfill under concrete hardness assumptions in the random oracle model, and, separately, that provably suffices for recursive composition in the plain model without random oracles.

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 a previous instance z_i, proof π_i, and accumulator acc_i; accumulates (z_i, π_i) with acc_i to obtain a new accumulator acc_{i+1} and accumulation proof pf_{i+1}; and generates a SNARK proof π_{i+1} of the following claim expressed as a circuit R (see Fig. 2, middle box): "$z_{i+1} = F(z_i)$, and there exist a SNARK proof π_i, accumulator acc_i, and accumulation proof pf_{i+1} such that the accumulation verifier accepts $((z_i, \pi_i), \mathsf{acc}_i, \mathsf{acc}_{i+1}, \mathsf{pf}_{i+1})$". 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 the instance (z_i, acc_i) and proof π_i, and running the accumulation scheme decider on the accumulator 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 SNARK 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 SNARK 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 $((z_i, \pi_i), \mathsf{acc}_i, \mathsf{acc}_{i+1}, \mathsf{pf}_{i+1})$, then π_i is a valid SNARK 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 to 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 NARK proof *instance* $\pi_i.\mathbb{x}$, an accumulator *instance* $\mathsf{acc}_i.\mathbb{x}$, and an accumulation proof pf_{i+1} such that the accumulation verifier accepts $((z_i, \pi_i.\mathbb{x}), \mathsf{acc}_i.\mathbb{x}, \mathsf{acc}_{i+1}.\mathbb{x}, \mathsf{pf}_{i+1})$", and accordingly the IVC verifier runs the NARK verifier on $((z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i)$ (in addition to running the accumulation scheme decider on the accumulator acc_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 the full version 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 "multi-instance 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 (knowledge soundness for PCD vs. IVC). The proof of security for PCD extracts a transcript *one full layer at a time*. Since a layer consists of many nodes, each with an *independently-generated* proof and accumulator, a standard "single-instance" extraction guarantee is insufficient in general. However, in the special case of IVC, every layer consists of exactly one node, and so single-instance extraction does suffice.

Remark 3 (flavors of PCD). The recent advances in PCD from accumulation achieve weaker efficiency guarantees 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

recursion circuit via succinct verification	$R\big((\mathrm{ivk}, z_{i+1}), (z_i, \pi_i)\big)$: • check that $z_{i+1} = F(z_i)$ • set SNARK instance $\mathbb{x}_i := (\mathrm{ivk}, z_i)$ • check that $\mathsf{SNARK}.\mathcal{V}(\mathrm{ivk}, \mathbb{x}_i, \pi_i) = 1$
recursion circuit via atomic accumulation	$R\big((\mathsf{avk}, z_{i+1}, \mathsf{acc}_{i+1}), (z_i, \pi_i, \mathsf{acc}_i, \mathsf{pf}_{i+1})\big)$: • check that $z_{i+1} = F(z_i)$ • set predicate input $\mathsf{q}_i := ((\mathsf{avk}, z_i, \mathsf{acc}_i), \pi_i)$ • check that $\mathsf{ACC}.\mathrm{V}(\mathsf{avk}, \mathsf{q}_i, \mathsf{acc}_i, \mathsf{acc}_{i+1}, \mathsf{pf}_{i+1}) = 1$
recursion circuit via split accumulation	$R\big((\mathsf{avk}, z_{i+1}, \mathsf{acc}_{i+1}.\mathbb{x}), (z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}, \mathsf{pf}_{i+1})\big)$: • check that $z_{i+1} = F(z_i)$ • set predicate instance $\mathsf{qx}_i := ((\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i.\mathbb{x})$ • check that $\mathsf{ACC}.\mathrm{V}(\mathsf{avk}, \mathsf{qx}_i, \mathsf{acc}_i.\mathbb{x}, \mathsf{acc}_{i+1}.\mathbb{x}, \mathsf{pf}_{i+1}) = 1$

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

algorithms is independent of the number of nodes in the PCD computation, which is how PCD is defined.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 (zero knowledge) NARK for R1CS that has a (zero knowledge) split accumulation scheme whose accumulation verifier has constant size (more precisely, performs a constant number of group scalar multiplications, field operations, and random oracle calls).

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

Definition 3 (R1CS problem). *Given a finite field $\mathbb{F}$, coefficient matrices $A, B, C \in \mathbb{F}^{\mathsf{M} \times \mathsf{N}}$, and an instance vector $x \in \mathbb{F}^{\mathsf{n}}$, is there a witness vector $w \in \mathbb{F}^{\mathsf{N}-\mathsf{n}}$ such that $Az \circ Bz = Cz$ for $z := (x, w) \in \mathbb{F}^{\mathsf{N}}$?*

We explain our construction incrementally. In Section 2.3.1 we begin by describing a NARK for R1CS that is *not* zero knowledge, and a "basic" split accumulation scheme for it that is also not zero knowledge. In Section 2.3.2 we show how to extend the NARK and its split accumulation scheme to both be zero knowledge. In Section 2.3.3 we explain why the accumulation scheme described so far is limited to the special case of 1 old accumulator and 1 predicate input (which suffices for IVC), and sketch how to obtain accumulation for m old accumulators and n predicate inputs (which is required for PCD); this motivates the problem of accumulating Hadamard products, which we subsequently address in Section 2.5.

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 constructions that enable recursion!), and view this as a significant advantage for potential deployments of these ideas in the real world.

2.3.1 Without zero knowledge

Let $\mathsf{ck} = (G_1, \ldots, G_{\mathsf{M}}) \in \mathbb{G}^{\mathsf{M}}$ be a commitment key for the Pedersen commitment scheme with message space $\mathbb{F}^{\mathsf{M}}$, and let $\mathsf{Commit}(\mathsf{ck}, a) := \sum_{i\in[\mathsf{M}]} a_i \cdot G_i$ denote its commitment function. Consider the following non-interactive argument for R1CS:

$$\begin{array}{lcl}
\mathcal{P}\big(\mathsf{ck}, (A,B,C), x, w\big) & & \mathcal{V}\big(\mathsf{ck}, (A,B,C), x\big) \\
z := (x,w) \in \mathbb{F}^{\mathsf{N}} & & z := (x,w) \\
z_A := Az \in \mathbb{F}^{\mathsf{M}} \quad C_A := \mathsf{Commit}(\mathsf{ck}, z_A) \in \mathbb{G} & \xrightarrow{\;C_A, C_B, C_C, w\;} & z_A := Az \quad C_A \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_A) \\
z_B := Bz \in \mathbb{F}^{\mathsf{M}} \quad C_B := \mathsf{Commit}(\mathsf{ck}, z_B) \in \mathbb{G} & & z_B := Bz \quad C_B \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_B) \\
z_C := Cz \in \mathbb{F}^{\mathsf{M}} \quad C_C := \mathsf{Commit}(\mathsf{ck}, z_C) \in \mathbb{G} & & z_C := Cz \quad C_C \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_C) \\
 & & \qquad\qquad C_C \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_A \circ z_B)
\end{array}$$

The NARK's security follows from the binding property of Pedersen commitments. (At this point we are not using any homomorphic properties, but we will in the accumulation scheme.) Moreover, denoting by $\mathsf{K} = \Omega(\mathsf{M})$ the number of non-zero entries in the coefficient matrices, the NARK's efficiency is as follows:

NARK prover time	NARK verifier time	NARK argument size
$O(\mathsf{M})\ \mathbb{G}$	$O(\mathsf{M})\ \mathbb{G}$	$O(1)\ \mathbb{G}$
$O(\mathsf{K})\ \mathbb{F}$	$O(\mathsf{K})\ \mathbb{F}$	$O(\mathsf{N})\ \mathbb{F}$

The NARK may superficially appear useless because it has linear argument size and is not zero knowledge. Nevertheless, we can obtain an efficient split accumulation scheme for it, as we describe next.[3]

The predicate to be accumulated is the NARK verifier with a suitable split between predicate instance and predicate witness: Φ takes as input a predicate instance $\mathsf{qx} = (x, C_A, C_B, C_C)$ and a predicate witness $\mathsf{qw} = w$, and then runs the NARK verifier with R1CS instance x and proof $\pi = (C_A, C_B, C_C, w)$.[4]

[3] We could even "re-arrange" computation between the NARK and the accumulation scheme, and simplify the NARK further to be the NP decider (the verifier receives just the witness w and checks that the R1CS condition holds). We do not do so because this does not lead to any savings in the accumulation verifier (the main efficiency metric of interest) and also because the current presentation more naturally leads to the zero knowledge variant described in Section 2.3.2. (We note that the foregoing rearrangement is a general transformation that does not preserve zero knowledge or succinctness of the given NARK.)

[4] For now we view the commitment key ck and coefficient matrices A, B, C as hardcoded in the accumulation predicate Φ; our definitions later handle this more precisely.

An accumulator acc is split into an accumulator instance $\mathsf{acc}.\mathbb{x} = (x, C_A, C_B, C_C, C_\circ) \in \mathbb{F}^{\mathsf{n}} \times \mathbb{G}^4$ and an accumulator witness $\mathsf{acc}.\mathbb{w} = w \in \mathbb{F}^{\mathsf{N}-\mathsf{n}}$. The accumulation decider D validates a split accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ as follows: set $z := (x, w) \in \mathbb{F}^{\mathsf{N}}$; compute the vectors $z_A := Az$, $z_B := Bz$, and $z_C := Cz$; and check that the following conditions hold:

$$C_A \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_A)\ ,\ C_B \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_B)\ ,\ C_C \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_C)\ ,\ C_\circ \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_A \circ z_B)\ .$$

Note that the accumulation decider D is similar, *but not equal*, to the NARK verifier.

We are left to describe the accumulation prover and accumulation verifier. Both have access to a random oracle ρ. For adaptive security, queries to the random oracle should include a hash τ of the coefficient matrices A, B, C and instance size n, which can be precomputed in an offline phase. (Formally, this is done via the *indexer* algorithm of the accumulation scheme, which receives the coefficient matrices and instance size, performs all one-time computations such as deriving τ, and produces an accumulator proving key apk, an accumulator verification key avk, and a decision key dk for P, V, and D respectively.)

The intuition for accumulation is to set the new accumulator to be a random linear combination of the old accumulator and predicate input, and use the accumulation proof to collect cross terms that arise from the Hadamard product (a bilinear, not linear, operation). This naturally leads to the following simple construction.

```
$\mathrm{P}^{\rho_{\mathsf{AS}}}(\mathsf{acc}, (\mathsf{qx}, \mathsf{qw}))$:
 1. $z_A := A \cdot (\mathsf{qx}.x, \mathsf{qw}.w)$, $z_B := B \cdot (\mathsf{qx}.x, \mathsf{qw}.w)$.
 2. $z'_A := A \cdot (\mathsf{acc}.\mathbb{x}.x, \mathsf{acc}.\mathbb{w}.w)$, $z'_B := B \cdot (\mathsf{acc}.\mathbb{x}.x, \mathsf{acc}.\mathbb{w}.w)$.
 3. $\mathsf{pf} := \mathsf{Commit}(\mathsf{ck}, z_A \circ z'_B + z'_A \circ z_B)$.
 4. $\beta := \rho_{\mathsf{AS}}(\tau, \mathsf{acc}.\mathbb{x}, \mathsf{qx}, \mathsf{pf})$.
 5. $\mathsf{acc}^\star.\mathbb{x}.x := \mathsf{acc}.\mathbb{x}.x + \beta \cdot \mathsf{qx}.x$.
 6. $\mathsf{acc}^\star.\mathbb{x}.C_A := \mathsf{acc}.\mathbb{x}.C_A + \beta \cdot \mathsf{qx}.C_A$.
 7. $\mathsf{acc}^\star.\mathbb{x}.C_B := \mathsf{acc}.\mathbb{x}.C_B + \beta \cdot \mathsf{qx}.C_B$.
 8. $\mathsf{acc}^\star.\mathbb{x}.C_C := \mathsf{acc}.\mathbb{x}.C_C + \beta \cdot \mathsf{qx}.C_C$.
 9. $\mathsf{acc}^\star.\mathbb{x}.C_\circ := \mathsf{acc}.\mathbb{x}.C_\circ + \beta \cdot \mathsf{pf} + \beta^2 \cdot \mathsf{qx}.C_C$.
10. $\mathsf{acc}^\star.\mathbb{w}.w := \mathsf{acc}.\mathbb{w}.w + \beta \cdot \mathsf{qw}.w$.
11. Output $(\mathsf{acc}^\star, \mathsf{pf})$.
```

```
$\mathrm{V}^{\rho_{\mathsf{AS}}}(\mathsf{acc}.\mathbb{x}, \mathsf{qx}, \mathsf{acc}^\star.\mathbb{x}, \mathsf{pf})$:
1. $\beta := \rho_{\mathsf{AS}}(\tau, \mathsf{acc}.\mathbb{x}, \mathsf{qx}, \mathsf{pf})$.
2. $\mathsf{acc}^\star.\mathbb{x}.x \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.x + \beta \cdot \mathsf{qx}.x$.
3. $\mathsf{acc}^\star.\mathbb{x}.C_A \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_A + \beta \cdot \mathsf{qx}.C_A$.
4. $\mathsf{acc}^\star.\mathbb{x}.C_B \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_B + \beta \cdot \mathsf{qx}.C_B$.
5. $\mathsf{acc}^\star.\mathbb{x}.C_C \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_C + \beta \cdot \mathsf{qx}.C_C$.
6. $\mathsf{acc}^\star.\mathbb{x}.C_\circ \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_\circ + \beta \cdot \mathsf{pf} + \beta^2 \cdot \mathsf{qx}.C_C$.
```

The efficiency of the split accumulation scheme can be summarized by the following table:

accumulation prover time	accumulation verifier time	decider time	accumulator size
$O(\mathsf{M})\ \mathbb{G}$	$4\ \mathbb{G}$ [5]	$O(\mathsf{M})\ \mathbb{G}$	$\lvert\mathsf{acc}.\mathbb{x}\rvert = 4\ \mathbb{G} + \mathsf{n}\ \mathbb{F}$
$O(\mathsf{K})\ \mathbb{F}$	$O(\mathsf{n})\ \mathbb{F}$	$O(\mathsf{K})\ \mathbb{F}$	$\lvert\mathsf{acc}.\mathbb{w}\rvert = (\mathsf{N} - \mathsf{n})\ \mathbb{F}$
1 RO	1 RO	–	–

The key efficiency feature is that the accumulation verifier only performs 1 call to the random oracle, a constant number of group scalar multiplications, and field operations. (More precisely, the verifier makes n field operations, but this does not grow with circuit size and, more fundamentally, is inevitable because the accumulation verifier must receive the R1CS instance $x \in \mathbb{F}^{\mathsf{n}}$ as input.)

2.3.2 With zero knowledge

We explain how to add zero knowledge to the approach described in the previous section.

First, we extend the NARK to additionally achieve zero knowledge. For this we construct a sigma protocol for R1CS based on Pedersen commitments, which is summarized in Figure 3; then we apply the Fiat–Shamir transformation to it to obtain a corresponding zkNARK for R1CS. Here the commitment key for the Pedersen commitment is $\mathsf{ck} := (G_1, \ldots, G_{\mathsf{M}}, H) \in \mathbb{G}^{\mathsf{M}+1}$, as we need a spare group element for the commitment randomness. The blue text in the figure represents the "diff" compared to the non-zero-knowledge version, and indeed if all such text were removed the protocol would collapse to the previous one.

Second, we extend the split accumulation scheme to accumulate the modified protocol for R1CS. Again the predicate being accumulated is the NARK verifier but now since the NARK verifier has changed so does the predicate. A zkNARK proof π now can be viewed as a pair (π_1, π_2) denoting the prover's commitment and response in the sigma protocol. Then the predicate Φ takes as input a predicate instance $\mathsf{qx} = (x, \pi_1) \in \mathbb{F}^{\mathsf{n}} \times \mathbb{G}^8$ and a predicate witness $\mathsf{qw} = \pi_2 \in \mathbb{F}^{\mathsf{N}-\mathsf{n}+4}$, and then runs the NARK verifier with R1CS instance x and proof $\pi = (\pi_1, \pi_2)$.

An accumulator acc is split into an accumulator instance $\mathsf{acc}.\mathbb{x} = (x, C_A, C_B, C_C, C_\circ) \in \mathbb{F}^{\mathsf{n}} \times \mathbb{G}^4$ (the same as before) and an accumulator witness $\mathsf{acc}.\mathbb{w} = (w, \sigma_A, \sigma_B, \sigma_C, \sigma_\circ) \in \mathbb{F}^{\mathsf{N}-\mathsf{n}+4}$. The decider is essentially the same as in Section 2.3.1, except that now the four commitments are computed using the corresponding randomness in $\mathsf{acc}.\mathbb{w}$.

The accumulation prover and accumulation verifier can be extended, in a straightforward way, to support the new zkSNARK protocol; we provide these in Figure 4, with text in blue to denote the "diff" to accumulate the zero knowledge features of the NARK and with text in red to denote the features to make accumulation itself zero knowledge. There we use ρ_{NARK} to denote the oracle used for the zkNARK for R1CS, which is obtained via the Fiat–Shamir transformation applied to a sigma protocol (as mentioned above); for adaptive security, the Fiat–Shamir query includes, in addition to π_1, a hash $\tau := \rho_{\mathsf{NARK}}(A, B, C, \mathsf{n})$ of the coefficient matrices and the R1CS input $x \in \mathbb{F}^{\mathsf{n}}$ (this means that the Fiat–Shamir query equals $(\tau, \mathsf{qx}) = (\tau, x, \pi_1)$).

Note that now the accumulation prover and accumulation verifier are each making 2 calls to the random oracle, rather than 1 as before, because they have to additionally compute the sigma protocol's challenge.

2.3.3 Towards general accumulation

The accumulation schemes described in Sections 2.3.1 and 2.3.2 are limited to a special case, which we could call the "IVC setting", where accumulation involves 1 old accumulator and 1 predicate input. However, the definition of accumulation requires supporting m old accumulators $[\mathsf{acc}_j]_{j=1}^m = [(\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})]_{j=1}^m$ and n predicate inputs

[5] The verifier performs 4 group scalar multiplication by computing $\beta \cdot \mathsf{qx}.C_C$ and then $\beta \cdot \mathsf{pf} + \beta^2 \cdot \mathsf{qx}.C_C = \beta \cdot (\mathsf{pf} + \beta \cdot \mathsf{qx}.C_C)$ via another group scalar multiplication. Further it is possible to combine C_A and C_B in one commitment in both the NARK and the accumulation scheme. This reduces the group scalar multiplications in the verifier to 3, and the accumulator size to $3\ \mathbb{G} + \mathsf{n}\ \mathbb{F}$.

$\mathcal{P}(\mathsf{ck}, (A, B, C), x, w)$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathcal{V}(\mathsf{ck}, (A, B, C), x)$

$z := (x, w) \quad r \leftarrow \mathbb{F}^{\mathsf{N}-\mathsf{n}}$

$z_A := Az \quad \omega_A \leftarrow \mathbb{F} \quad C_A := \mathsf{Commit}(\mathsf{ck}, z_A; \omega_A)$

$z_B := Bz \quad \omega_B \leftarrow \mathbb{F} \quad C_B := \mathsf{Commit}(\mathsf{ck}, z_B; \omega_B)$

$z_C := Cz \quad \omega_C \leftarrow \mathbb{F} \quad C_C := \mathsf{Commit}(\mathsf{ck}, z_C; \omega_C)$

$r_A := A \cdot (0^{\mathsf{n}}, r) \quad \omega'_A \leftarrow \mathbb{F} \quad C'_A := \mathsf{Commit}(\mathsf{ck}, r_A; \omega'_A)$

$r_B := B \cdot (0^{\mathsf{n}}, r) \quad \omega'_B \leftarrow \mathbb{F} \quad C'_B := \mathsf{Commit}(\mathsf{ck}, r_B; \omega'_B)$

$r_C := C \cdot (0^{\mathsf{n}}, r) \quad \omega'_C \leftarrow \mathbb{F} \quad C'_C := \mathsf{Commit}(\mathsf{ck}, r_C; \omega'_C)$

$\omega_1 \leftarrow \mathbb{F} \quad C_1 := \mathsf{Commit}(\mathsf{ck}, z_A \circ r_B + z_B \circ r_A; \omega_1)$

$\omega_2 \leftarrow \mathbb{F} \quad C_2 := \mathsf{Commit}(\mathsf{ck}, r_A \circ r_B; \omega_2)$

$$\xrightarrow{\quad C_A, C_B, C_C,\; C'_A, C'_B, C'_C, C_1, C_2 \quad}$$

$$\xleftarrow{\quad \gamma \in \mathbb{F} \quad}$$

$s := w + \gamma r \in \mathbb{F}^{\mathsf{N}-\mathsf{n}}$

$\sigma_A := \omega_A + \gamma\omega'_A \in \mathbb{F}$

$\sigma_B := \omega_B + \gamma\omega'_B \in \mathbb{F}$

$\sigma_C := \omega_C + \gamma\omega'_C \in \mathbb{F}$

$\sigma_\circ := \omega_C + \gamma\omega_1 + \gamma^2\omega_2 \in \mathbb{F}$

$$\xrightarrow{\quad s, \sigma_A, \sigma_B, \sigma_C, \sigma_\circ \quad}$$

$s_A := A \cdot (x, s) \quad C_A + \gamma C'_A \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, s_A; \sigma_A)$

$s_B := B \cdot (x, s) \quad C_B + \gamma C'_B \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, s_B; \sigma_B)$

$s_C := C \cdot (x, s) \quad C_C + \gamma C'_C \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, s_C; \sigma_C)$

$C_C + \gamma C_1 + \gamma^2 C_2 \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, s_A \circ s_B; \sigma_\circ)$

Fig. 3: The sigma protocol for R1CS that underlies the zkNARK for R1CS.

```
$\mathrm{P}^{\rho_{\mathsf{AS}}}((\mathsf{qx}, \mathsf{qw}), \mathsf{acc})$:
 1. $z_A := A \cdot (\mathsf{qx}.x, \mathsf{qw}.s)$, $z_B := B \cdot (\mathsf{qx}.x, \mathsf{qw}.s)$.
 2. $z'_A := A \cdot (\mathsf{acc}.\mathbb{x}.x, \mathsf{acc}.\mathbb{w}.s)$, $z'_B := B \cdot (\mathsf{acc}.\mathbb{x}.x, \mathsf{acc}.\mathbb{w}.s)$.
 3. Sample $x^\star \leftarrow \mathbb{F}^{\mathsf{n}}$ and $s^\star \leftarrow \mathbb{F}^{\mathsf{N}-\mathsf{n}}$ and $\omega_2^\star \leftarrow \mathbb{F}$.
 4. $s_A^\star := A \cdot (x^\star, s^\star)$, $s_B^\star := B \cdot (x^\star, s^\star)$, $s_C^\star := C \cdot (x^\star, s^\star)$.
 5. $C_A^\star := \mathsf{Commit}(\mathsf{ck}, s_A^\star; \omega_A^\star)$ for $\omega_A^\star \leftarrow \mathbb{F}$.
 6. $C_B^\star := \mathsf{Commit}(\mathsf{ck}, s_B^\star; \omega_B^\star)$ for $\omega_B^\star \leftarrow \mathbb{F}$.
 7. $C_C^\star := \mathsf{Commit}(\mathsf{ck}, s_C^\star; \omega_C^\star)$ for $\omega_C^\star \leftarrow \mathbb{F}$.
 8. $\mathsf{pf}_1 := \mathsf{Commit}(\mathsf{ck}, z_A \circ s_B^\star + s_A^\star \circ z_B; 0)$.
 9. $\mathsf{pf}_2 := \mathsf{Commit}(\mathsf{ck}, s_A^\star \circ s_B^\star + z_A \circ z'_B + z'_A \circ z_B; \omega_2^\star)$.
10. $\mathsf{pf}_3 := \mathsf{Commit}(\mathsf{ck}, s_A^\star \circ z'_B + z'_A \circ s_B^\star; 0)$.
11. $\mathsf{pf} := (x^\star, C_A^\star, C_B^\star, C_C^\star, \mathsf{pf}_1, \mathsf{pf}_2, \mathsf{pf}_3)$.
12. $\beta := \rho_{\mathsf{AS}}(\tau, \mathsf{acc}.\mathbb{x}, \mathsf{qx}, \mathsf{pf})$.
13. Compute $\gamma := \rho_{\mathsf{NARK}}(\tau, \mathsf{qx})$.
14. $\mathsf{acc}^\star.\mathbb{x}.x := \mathsf{acc}.\mathbb{x}.x + \beta \cdot x^\star + \beta^2 \cdot \mathsf{qx}.x$.
15. $\mathsf{acc}^\star.\mathbb{x}.C_A := \mathsf{acc}.\mathbb{x}.C_A + \beta \cdot C_A^\star + \beta^2 \cdot (\mathsf{qx}.C_A + \gamma \cdot \mathsf{qx}.C'_A)$.
16. $\mathsf{acc}^\star.\mathbb{x}.C_B := \mathsf{acc}.\mathbb{x}.C_B + \beta \cdot C_B^\star + \beta^2 \cdot (\mathsf{qx}.C_B + \gamma \cdot \mathsf{qx}.C'_B)$.
17. $\mathsf{acc}^\star.\mathbb{x}.C_C := \mathsf{acc}.\mathbb{x}.C_C + \beta \cdot C_C^\star + \beta^2 \cdot (\mathsf{qx}.C_C + \gamma \cdot \mathsf{qx}.C'_C)$.
18. $\mathsf{acc}^\star.\mathbb{x}.C_\circ := \mathsf{acc}.\mathbb{x}.C_\circ + \beta \cdot \mathsf{pf}_1 + \beta^2 \cdot \mathsf{pf}_2 + \beta^3 \cdot \mathsf{pf}_3$
        $+ \beta^4 \cdot (\mathsf{qx}.C_C + \gamma \cdot C_1 + \gamma^2 \cdot C_2)$.
19. $\mathsf{acc}^\star.\mathbb{w}.s := \mathsf{acc}.\mathbb{w}.s + \beta \cdot s^\star + \beta^2 \cdot \mathsf{qw}.s$.
20. $\mathsf{acc}^\star.\mathbb{w}.\sigma_A := \mathsf{acc}.\mathbb{w}.\sigma_A + \beta \cdot \omega_A^\star + \beta^2 \cdot \mathsf{qw}.\sigma_A$.
21. $\mathsf{acc}^\star.\mathbb{w}.\sigma_B := \mathsf{acc}.\mathbb{w}.\sigma_B + \beta \cdot \omega_B^\star + \beta^2 \cdot \mathsf{qw}.\sigma_B$.
22. $\mathsf{acc}^\star.\mathbb{w}.\sigma_C := \mathsf{acc}.\mathbb{w}.\sigma_C + \beta \cdot \omega_C^\star + \beta^2 \cdot \mathsf{qw}.\sigma_C$.
23. $\mathsf{acc}^\star.\mathbb{w}.\sigma_\circ := \mathsf{acc}.\mathbb{w}.\sigma_\circ + \beta^2 \cdot \omega_2^\star + \beta^4 \cdot \mathsf{qw}.\sigma_\circ$.
24. Output $(\mathsf{acc}^\star, \mathsf{pf})$.
```

```
$\mathrm{V}^{\rho_{\mathsf{AS}}}(\mathsf{qx}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}^\star.\mathbb{x}, \mathsf{pf})$:
1. $\beta := \rho_{\mathsf{AS}}(\tau, \mathsf{acc}.\mathbb{x}, \mathsf{qx}, \mathsf{pf})$.
2. $\gamma := \rho_{\mathsf{NARK}}(\tau, \mathsf{qx})$.
3. $\mathsf{acc}^\star.\mathbb{x}.x \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.x + \beta \cdot x^\star + \beta^2 \cdot \mathsf{qx}.x$.
4. $\mathsf{acc}^\star.\mathbb{x}.C_A \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_A + \beta \cdot C_A^\star + \beta^2 \cdot (\mathsf{qx}.C_A + \gamma \cdot \mathsf{qx}.C'_A)$.
5. $\mathsf{acc}^\star.\mathbb{x}.C_B \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_B + \beta \cdot C_B^\star + \beta^2 \cdot (\mathsf{qx}.C_B + \gamma \cdot \mathsf{qx}.C'_B)$.
6. $\mathsf{acc}^\star.\mathbb{x}.C_C \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_C + \beta \cdot C_C^\star + \beta^2 \cdot (\mathsf{qx}.C_C + \gamma \cdot \mathsf{qx}.C'_C)$.
7. $\mathsf{acc}^\star.\mathbb{x}.C_\circ \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_\circ + \beta \cdot \mathsf{pf}_1 + \beta^2 \cdot \mathsf{pf}_2 + \beta^3 \cdot \mathsf{pf}_3$
       $+ \beta^4 \cdot (\mathsf{qx}.C_C + \gamma \cdot C_1 + \gamma^2 \cdot C_2)$.
```

Fig. 4: Accumulation prover and accumulation verifier for the zkNARK for R1CS.

$[(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^{n}$, for any m and n. (E.g., to construct PCD we set both m and n equal to the "arity" of the compliance predicate.) How can we extend the ideas described so far to this more general case?

The zkNARK verifier performs two types of computations: linear checks and a Hadamard product check. We describe how to accumulate each of these in the general case.

- *Linear checks.* A split accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ in Section 2.3.2 included sub-accumulators for different linear checks: x, C_A, C_B, C_C in $\mathsf{acc}.\mathbb{x}$ and $w, \sigma_A, \sigma_B, \sigma_C$ in $\mathsf{acc}.\mathbb{w}$. We can keep these components and simply use more random coefficients or, as we do, further powers of the element β. For example, in the accumulation prover P a computation such as $\mathsf{acc}^{\star}.\mathbb{x}.x := \mathsf{acc}.\mathbb{x}.x + \beta \cdot \mathsf{qx}.x$ is replaced by a computation such as $\mathsf{acc}^{\star}.\mathbb{x}.x := \sum_{j=1}^{m} \beta^{j-1} \cdot \mathsf{acc}_j.\mathbb{x}.x + \sum_{i=1}^{n} \beta^{m+j-1} \cdot \mathsf{qx}_i.x$.
- *Hadamard product check.* A split accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ in Section 2.3.2 also included a sub-accumulator for the Hadamard product check: $C_\circ$ in $\mathsf{acc}.\mathbb{x}$ and $\sigma_\circ$ in $\mathsf{acc}.\mathbb{w}$. Because a Hadamard product is a *bi*linear operation, combining two Hadamard products via a random coefficient led to a quadratic polynomial whose coefficients include the two original Hadamard products and a cross term. This is indeed why we stored the cross term in the accumulation proof pf. However, if we consider the cross terms that arise from combining more than two Hadamard products (i.e., when $m + n > 2$) then the corresponding polynomials do not lend themselves to accumulation because the original Hadamard products appear together with other cross terms. To handle this issue, we introduce in Section 2.5 a new subroutine that accumulates Hadamard products via an additional round of interaction.

2.4 On proving knowledge soundness

In order to construct accumulation schemes that fulfill the type of knowledge soundness that we ultimately need for PCD (see Section 2.2), we formulate a new *expected-time forking lemma in the random oracle model*, which is informally stated below. In our setting, $(\mathfrak{q}, \mathfrak{b}, \mathsf{o}) \in L$ if $\mathsf{o} = ([\mathsf{qx}_i]_{i=1}^{n}, \mathsf{acc}, \mathsf{pf})$ is such that $\mathrm{D}(\mathsf{acc}) = 1$ and, given that $\rho(\mathfrak{q}) = \mathfrak{b}$, the accumulation verifier accepts: $\mathrm{V}^{\rho}([\mathsf{qx}_i]_{i=1}^{n}, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1$.

Lemma 1 (informal)**.** *Let L be an efficiently recognizable set. There exists an algorithm* Fork *such that for every expected polynomial time algorithm A and integer $N \in \mathbb{N}$ the following holds. With all but negligible probability over the choice of random oracle ρ, randomness r of A, and randomness of* Fork*, if $A^{\rho}(r)$ outputs a tuple $(\mathfrak{q}, \mathfrak{b}, \mathsf{o}) \in L$ with $\rho(\mathfrak{q}) = \mathfrak{b}$, then $\mathsf{Fork}^{A,\rho}(1^N, \mathfrak{q}, \mathfrak{b}, \mathsf{o}, r)$ outputs $[(\mathfrak{b}_j, \mathsf{o}_j)]_{j=1}^{N}$ such that $\mathfrak{b}_1, \ldots, \mathfrak{b}_N$ are pairwise distinct and for each $j \in [N]$ it holds that $(\mathfrak{q}, \mathfrak{b}_j, \mathsf{o}_j) \in L$.*

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 [BCC+16]). Second, we can obtain any polynomial number of accepting transcripts in expected polynomial time with only negligible loss in success probability (unlike forking lemmas for signature schemes, which typically extract two transcripts in strict polynomial time [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 in one recursive step becomes the adversary in the next. This last feature requires some care, since the running time of the adversary, and in particular the length of its random tape, may not be bounded.

Moreover, in our security proofs we at times additionally rely on an expected-time variant of the *zero-finding game lemma* from [BCMS20] to show that if a particular polynomial equation holds at a point obtained from the random oracle via a "commitment" to the equation, then it must with overwhelming probability be a polynomial identity. For more details, see the full version.

2.5 Split accumulation for Hadamard products

We construct a split accumulation scheme for a predicate Φ_{HP} that considers the Hadamard product of committed vectors. For a commitment key ck for messages in $\mathbb{F}^{\ell}$, the predicate Φ_{HP} takes as input a predicate instance $\mathsf{qx} = (C_1, C_2, C_3) \in \mathbb{G}^3$ consisting of three Pedersen commitments, a predicate witness $\mathsf{qw} = (a, b, \omega_1, \omega_2, \omega_3)$ consisting of two vectors $a, b \in \mathbb{F}^{\ell}$ and three opening randomness elements $\omega_1, \omega_2, \omega_3 \in \mathbb{F}$, and checks that $C_1 = \mathsf{CM.Commit}(\mathsf{ck}, a; \omega_1)$, $C_2 = \mathsf{CM.Commit}(\mathsf{ck}, b; \omega_2)$, and $C_3 = \mathsf{CM.Commit}(\mathsf{ck}, a \circ b; \omega_3)$. In other words, C_3 is a commitment to the Hadamard product of the vectors committed in C_1 and C_2.

Theorem 3 (informal)**.** *The Hadamard product predicate Φ_{HP} has a split accumulation scheme $\mathsf{AS}_{\mathsf{HP}}$ that is secure in the random oracle model (and assuming the hardness of the discrete logarithm problem) where verifying accumulation requires 5 group scalar multiplications and $O(1)$ field operations per claim, and results in an accumulator whose instance part is 3 group elements and witness part is $O(\ell)$ field elements. Moreover, the accumulation scheme can be made zero knowledge at a sub-constant overhead per claim.*

Below we summarize the ideas behind this result. Our construction directly extends to accumulate any bilinear function (see Remark 4).

A bivariate identity. The accumulation scheme is based on a bivariate polynomial identity, and is the result of turning a public-coin two-round reduction into a non-interactive scheme by using the random oracle. Given n pairs of vectors $[(a_i, b_i)]_{i=1}^{n}$, consider the following two polynomials with coefficients in $\mathbb{F}^{\ell}$:

$$a(X,Y) := \textstyle\sum_{i=1}^{n} X^{i-1} Y^{i-1} a_i \quad \text{and} \quad b(X) := \textstyle\sum_{i=1}^{n} X^{n-i} b_i \ .$$

The Hadamard product of the two polynomials can be written as

$$a(X,Y) \circ b(X) = \textstyle\sum_{i=1}^{2n-1} X^{i-1} t_i(Y) \quad \text{where} \quad t_n(Y) = \textstyle\sum_{i=1}^{n} Y^{i-1} a_i \circ b_i \ .$$

The expression of the coefficient polynomials $\{t_i(Y)\}_{i \neq n}$ is not important; instead, the important aspect here is that a coefficient polynomial, namely $t_n(Y)$, includes the Hadamard products of all n pairs of vectors as different coefficients. This identity is the starting point of the accumulation scheme, which informally evaluates this expression at random points to reduce the n Hadamard products to 1 Hadamard product. Similar ideas are used to reduce several Hadamard products to a single inner product in [BCC+16; BBB+18].

Batching Hadamard products. We describe a public-coin two-round reduction from n Hadamard product claims to 1 Hadamard product claim. The verifier receives n predicate instances $[\mathsf{qx}_i]_{i=1}^n = [(C_{1,i}, C_{2,i}, C_{3,i})]_{i=1}^n$ each consisting of three Pedersen commitments, and the prover receives corresponding predicate witnesses $[\mathsf{qw}_i]_{i=1}^n = [(a_i, b_i, \omega_{1,i}, \omega_{2,i}, \omega_{3,i})]_{i=1}^n$ containing the corresponding openings.

- The verifier sends a first challenge $\mu \in \mathbb{F}$.
- The prover computes the product polynomial $a(X, \mu) \circ b(X) = \sum_{i=1}^{2n-1} X^{i-1} t_i(\mu) \in \mathbb{F}^\ell[X]$; for each $i \in [2n-1] \backslash \{n\}$, computes the commitment $C_{t,i} := \mathsf{CM.Commit}(\mathsf{ck}, t_i; 0) \in \mathbb{G}$; and sends to the verifier an accumulation proof $\mathsf{pf} := [C_{t,i}, C_{t,n+i}]_{i=1}^{n-1}$.
- The verifier sends a second challenge $\nu \in \mathbb{F}$.
- The verifier computes and outputs a new predicate instance $\mathsf{qx} = (C_1, C_2, C_3)$:

$$\begin{aligned}
C_1 &= \textstyle\sum_{i=1}^n \nu^{i-1} \mu^{i-1} C_{1,i} \ , \\
C_2 &= \textstyle\sum_{i=1}^n \nu^{n-i} C_{2,i} \ , \\
C_3 &= \textstyle\sum_{i=1}^{n-1} \nu^{i-1} C_{t,i} + \nu^{n-1} \sum_{i=1}^n \mu^{i-1} C_{3,i} + \sum_{i=1}^{n-1} \nu^{n+i-1} C_{t,n+i} \ .
\end{aligned}$$

- The prover computes and outputs a corresponding predicate witness $\mathsf{qw} = (a, b, \omega_1, \omega_2, \omega_3)$:

$$\begin{aligned}
a &:= \textstyle\sum_{i=1}^n \nu^{i-1} \mu^{i-1} a_i & \omega_1 &:= \textstyle\sum_{i=1}^n \nu^{i-1} \mu^{i-1} \omega_{1,i} \ , \\
b &:= \textstyle\sum_{i=1}^n \nu^{n-i} b_i & \omega_2 &:= \textstyle\sum_{i=1}^n \nu^{n-i} \omega_{2,i} \ , \\
& & \omega_3 &:= \nu^{n-1} \textstyle\sum_{i=1}^n \mu^{i-1} \omega_{3,i} \ .
\end{aligned}$$

Observe that the new predicate instance $\mathsf{qx} = (C_1, C_2, C_3)$ consists of commitments to $a(\nu, \mu), b(\nu), a(\nu, \mu) \circ b(\nu)$ respectively, and the predicate witness $\mathsf{qw} = (a, b, \omega_1, \omega_2, \omega_3)$ consists of corresponding opening information. The properties of low-degree polynomials imply that if any of the n claims is incorrect (there is $i \in [n]$ such that $\Phi_{\mathsf{HP}}(\mathsf{qx}_i, \mathsf{qw}_i) = 0$) then, with high probability, so is the output claim ($\Phi_{\mathsf{HP}}(\mathsf{qx}, \mathsf{qw}) = 0$).

Split accumulation. The batching protocol described above yields a split accumulation scheme for Φ_{HP} in the random oracle model. An accumulator acc has the same form as a predicate input $(\mathsf{qx}, \mathsf{qw})$: $\mathsf{acc}.\mathbb{x}$ has the same form as a predicate instance qx, and $\mathsf{acc}.\mathbb{w}$ has the same form as a predicate witness qw. The accumulation decider D simply equals Φ_{HP} (this is well-defined due to the prior sentence). The accumulation prover and accumulation verifier are as follows.

- 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 produce an accumulation proof pf as well as an accumulator $\mathsf{acc} = (\mathsf{qx}, \mathsf{qw})$ whose instance part is computed like the verifier of the reduction and witness part is computed like the prover of the reduction.
- The accumulation verifier V re-derives the challenges using the random oracle, and checks that qx was correctly derived from $[\mathsf{qx}_i]_{i=1}^n$ (also via the help of the accumulation proof pf).

The construction described above is not zero knowledge. One way to achieve zero knowledge is for the accumulation prover to sample a random predicate input that satisfies the predicate, accumulate it, and include it as part of the accumulation proof pf. In our construction we opt for a more efficient solution, leveraging the fact that we are not actually interested in accumulating the random predicate input.

Efficiency. The efficiency claimed in Theorem 3 is evident from the construction. The (short) instance part of an accumulator consists of 3 group elements, while the (long) witness part of an accumulator consists of $O(\ell)$ field elements. The accumulator verifier V performs 2 random oracle calls, 5 group scalar multiplication, and $O(1)$ field operations per accumulated claim.

Security. Given an adversary that produces Hadamard product claims $[\mathsf{qx}_i]_{i=1}^n = [(C_{1,i}, C_{2,i}, C_{3,i})]_{i=1}^n$, a single Hadamard product claim $\mathsf{qx} = (C_1, C_2, C_3)$ and corresponding witness $\mathsf{qw} = (a, b, \omega_1, \omega_2, \omega_3)$, and an accumulation proof pf that makes the accumulation verifier accept, we need to extract witnesses $[\mathsf{qw}_i]_{i=1}^n = [(a_i, b_i, \omega_{1,i}, \omega_{2,i}, \omega_{3,i})]_{i=1}^n$ for the instances $[\mathsf{qx}_i]_{i=1}^n$. Our security proof works in the random oracle model, assuming hardness of the discrete logarithm problem.

In the proof we apply our expected-time forking lemma *twice* (see Section 2.4 for a discussion of this lemma and the full version for details including a corollary that summarizes its double invocation). This lets us construct a two-level tree of transcripts with branching factor n on the first challenge μ and branching factor $2n - 1$ on the second challenge ν. Given such a transcript tree, the extractor works as follows:

1. Using the transcripts corresponding to challenges $\{(\mu_1, \nu_{1,k})\}_{k \in [n]}$ we extract ℓ-element vectors $[a_i]_{i=1}^n, [b_i]_{i=1}^n$ and field elements $[\omega_{1,i}]_{i=1}^n, [\omega_{2,i}]_{i=1}^n$ such that $[a_i]_{i=1}^n$ and $[b_i]_{i=1}^n$ are committed in $[C_{1,i}]_{i=1}^n$ and $[C_{2,i}]_{i=1}^n$ under randomness $[\omega_{1,i}]_{i=1}^n$ and $[\omega_{2,i}]_{i=1}^n$, respectively.
2. Define $a(X, Y) := \sum_{i=1}^n X^{i-1} Y^{i-1} a_i \in \mathbb{F}^\ell[X, Y]$ and $b(X) := \sum_{i=1}^n X^{n-i} b_i \in \mathbb{F}^\ell[X]$, using the vectors extracted above; then let $t_i(Y)$ be the coefficient of X^{i-1} in $a(X, Y) \circ b(X)$. For each $j \in [n]$, using the transcripts corresponding to challenges $\{(\mu_j, \nu_{j,k})\}_{k \in [2n-1]}$, we extract field elements $[\tau_i^{(j)}]_{i=1}^{2n-1}$ such that $t_n(\mu_j)$ is committed in $\sum_{i=1}^{n-1} \mu_j^{i-1} C_{3,i}$ under randomness $\tau_n^{(j)}$ and $[t_i(\mu_j), t_{n+i}(\mu_j)]_{i=1}^{n-1}$ are committed in $\mathsf{pf}^{(j)} := [C_{t,i}^{(j)}, C_{t,n+i}^{(j)}]_{i=1}^{n-1}$ under randomness $[\tau_i^{(j)}, \tau_{n+i}^{(j)}]_{i=1}^{n-1}$ respectively.
3. Compute the solution $[\omega_{3,i}]_{i=1}^n$ to the linear system $\{\tau_n^{(j)} = \sum_{i=1}^{n-1} \mu_j^{i-1} \omega_{3,i}\}_{j \in [n]}$. Together with the relation $\{t_n(\mu_j) = \sum_{i=1}^{n-1} \mu_j^{i-1} a_i \circ b_i\}_{j \in [n]}$, we deduce that $C_{3,i}$ is a commitment to $a_i \circ b_i$ under randomness $\omega_{3,i}$ for all $i \in [n]$.
4. For each $i \in [n]$, output $\mathsf{qw}_i := (a_i, b_i, \omega_{1,i}, \omega_{2,i}, \omega_{3,i})$.

Remark 4 (extension to any bilinear operation). The ideas described above extend, in a straightforward way, to accumulating *any bilinear operation* of committed vectors. Let $f \colon \mathbb{F}^\ell \times \mathbb{F}^\ell \to \mathbb{F}^m$ be a bilinear operation, i.e., such that: (a) $f(a + a', b) = f(a, b) + f(a', b)$; (b) $f(a, b + b') = f(a, b) + f(a, b')$; (c) $\alpha \cdot f(a, b) = f(\alpha a, b) = f(a, \alpha b)$. Let Φ_f be the predicate that takes as input a predicate instance $\mathsf{qx} = (C_1, C_2, C_3) \in \mathbb{G}^3$ consisting of three Pedersen commitments, a predicate witness $\mathsf{qw} = (a, b, \omega_1, \omega_2, \omega_3)$ consisting of two vectors $a, b \in \mathbb{F}^\ell$ and three opening randomness elements $\omega_1, \omega_2, \omega_3 \in \mathbb{F}$, and checks that $C_1 = \mathsf{CM.Commit}(\mathsf{ck}_\ell, a; \omega_1)$, $C_2 = \mathsf{CM.Commit}(\mathsf{ck}_\ell, b; \omega_2)$, and $C_3 = \mathsf{CM.Commit}(\mathsf{ck}_m, f(a, b); \omega_3)$. The Hadamard product $\circ \colon \mathbb{F}^\ell \times \mathbb{F}^\ell \to \mathbb{F}^\ell$ is a bilinear operation, as is the scalar product $\langle\cdot,\cdot\rangle\colon \mathbb{F}^{\ell}\times\mathbb{F}^{\ell}\to\mathbb{F}$. Our accumulation scheme for Hadamard products works the same way, mutatis mutandis, for a general bilinear map f.

2.6 Split accumulation for Pedersen polynomial commitments

We construct an efficient split accumulation scheme $\mathsf{AS}_{\mathsf{PC}}$ for a predicate Φ_{PC} that checks a polynomial evaluation claim for a "trivial" polynomial commitment scheme $\mathsf{PC}_{\mathsf{Ped}}$ based on Pedersen commitments (see Fig. 5). In more detail, for a Pedersen commitment key ck for messages in $\mathbb{F}^{d+1}$, the predicate Φ_{PC} takes as input a predicate instance $\mathsf{qx} = (C, z, v) \in \mathbb{G}\times\mathbb{F}\times\mathbb{F}$ and a predicate witness $\mathsf{qw} = p \in \mathbb{F}^{\leq d}[X]$, and checks that $C = \mathsf{CM.Commit}(\mathsf{ck}, p)$, $p(z) = v$, and $\deg(p) \leq d$. In other words, the predicate Φ_{PC} checks that the polynomial p of degree at most d committed in C evaluates to v at z.

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

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

Fig. 5: $\mathsf{PC}_{\mathsf{Ped}}$ is a trivial polynomial commitment scheme based on the Pedersen commitment scheme CM.

Theorem 4 (informal)**.** *The* **(Pedersen) polynomial commitment predicate** Φ_{PC} *has a split accumulation scheme* $\mathsf{AS}_{\mathsf{PC}}$ *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.)*

One can use $\mathsf{AS}_{\mathsf{PC}}$ to obtain a split accumulation scheme for a different NARK; see Remark 5 for details.

In Table 1 we compare the efficiency of our split accumulation scheme $\mathsf{AS}_{\mathsf{PC}}$ for the predicate Φ_{PC} with the efficiency of the atomic accumulation scheme $\mathsf{AS}_{\mathsf{IPA}}$ [BCMS20] for the equivalent predicate defined by the check algorithm of the (succinct) PC scheme $\mathsf{PC}_{\mathsf{IPA}}$ based on the inner-product argument on cyclic groups [BCC+16; BBB+18; WTS+18]. The takeaway is that the accumulation verifier for $\mathsf{AS}_{\mathsf{PC}}$ is significantly cheaper than the accumulation verifier for $\mathsf{AS}_{\mathsf{IPA}}$.

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 †	$O(\log d)$ $\mathbb{G}$ $O(d)$ $\mathbb{F}$ $\left[+O(d)\ \mathbb{G} \text{ per accumulation}\right]$	$O(\log d)$ $\mathbb{G}$ $O(\log d)$ $\mathbb{F}$ $O(\log d)$ RO	$O(d)$ $\mathbb{G}$ $O(d)$ $\mathbb{F}$	1 $\mathbb{G}$ $O(\log d)$ $\mathbb{F}$	0
$\mathsf{AS}_{\mathsf{PC}}$ [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 $\mathsf{AS}_{\mathsf{PC}}$ for $\mathsf{PC}_{\mathsf{Ped}}$ 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 [BMM+19] or in the algebraic group model [GT20].)

Technical details are in the full version; in the rest of this section we sketch the ideas behind Theorem 4.

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) and (C_2, z, v_2) 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') 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) and (C_2, z_2, v_2) 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^*:

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

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

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

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 Φ_{PC} in the random oracle model. 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 proof pf consists of the messages to the verifier in the reduction, which includes the commitments to the witness polynomials W_i and the evaluations y_i, y'_i at z^* of p_i, w_i (that is, $\mathsf{pf} := [(W_i, y_i, y'_i)]_{i=1}^{n}$).
- 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 accumulator 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 4 (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). The (short) instance part of an accumulator consists of 1 group element and 2 field elements, while the (long) witness part of an accumulator consists of $O(d)$ 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.

Security. Given an adversary that produces evaluation claims $[\mathsf{qx}_i]_{i=1}^{n} = [(C_i, z_i, v_i)]_{i=1}^{n}$, a single claim $\mathsf{qx} = (C, z, v)$ and polynomial $\mathsf{qw} = s(X)$ with $s(z^*) = v$ to which C is a commitment, and accumulation proof pf that makes the accumulation verifier accept, we need to extract polynomials $[\mathsf{qw}_i]_{i=1}^{n} = [p_i(X)]_{i=1}^{n}$ with $p_i(z_i) = v_i$ to which C_i is a commitment. Our security proof (in the full version) works in the random oracle model, assuming hardness of the discrete logarithm problem.

In the proof, we apply our expected-time forking lemma (see Section 2.4) to obtain $2n$ polynomials $[s^{(j)}]_{j=1}^{2n}$ 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 $s^{(j)}(X) = \sum_{i=1}^{n} \alpha_j^i p_i(X) + \sum_{i=1}^{n} \alpha_j^{n+i} w_i(X)$, where $w_i(X)$ is the witness corresponding to $p_i(X)$; hence we can recover the $p_i(X), w_i(X)$ by solving a linear system (given by the Vandermonde matrix in the challenges $[\alpha_j]_{j=1}^{2n}$). We then use an expected-time variant of the zero-finding game lemma from [BCMS20] (see the full version) to show that if a particular polynomial equation on $p_i(X), w_i(X)$ 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, with high probability, each extracted polynomial p_i satisfies the corresponding evaluation claim (C_i, z_i, v_i).

Remark 5 (from $\mathsf{PC}_{\mathsf{Ped}}$ *to an accumulatable NARK).* If one replaced the (succinct) polynomial commitment scheme that underlies the preprocessing zkSNARK in [CHM+20] with the aforementioned (non-succinct) trivial Pedersen polynomial commitment scheme then (after some adjustments and using our Theorem 4) one would obtain a zkNARK 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 prover would additionally use a quasilinear number of field operations due to FFTs. Finally, in addition to poorer asymptotics, this approach would lead to a concretely more expensive accumulation verifier and overall a more complex protocol.

Nevertheless, one *can* design a concretely efficient zkNARK for R1CS based on the Pedersen PC scheme and our accumulation scheme for it. This naturally leads to an alternative construction to the one in Section 2.3 (which is instead based on accumulation of Hadamard products), and would lead to a slightly more expensive prover (which now would use FFTs) and a slightly cheaper accumulation verifier (a smaller number of group scalar multiplications). We leave this as an exercise for the interested reader.

2.7 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 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.[6] This route offers excellent verifier time (a few milliseconds regardless of the computation at a PCD node), but requires

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

a private-coin 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*. Such a framework would enable a number of desirable features: (a) ease of replacing older building blocks with new ones; (b) ease of prototyping different PCD constructions for different applications (which may have different needs), thereby enabling practitioners to make informed choices about which PCD construction is best for them; (c) simpler and more efficient auditing of complex cryptographic systems with many intermixed layers. (Realizing even a single PCD construction is a substantial implementation task.); and (d) separation of "application" logic from the underlying recursion via a common PCD interface. Together, these features would enable further industrial deployment of PCD, as well as making future research and comparisons simpler.

Implementation . The above 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 (the latter is integrated with PCD from succinct verification under a unified PCD interface).

- *Framework for accumulation.* We designed a modular framework for (atomic and split) accumulation schemes, and use it to implement, under a common interface, several accumulation schemes: (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 $\mathsf{AS}_{\mathsf{PC}}$ in this paper for the PC scheme $\mathsf{PC}_{\mathsf{Ped}}$; (d) the split accumulation scheme $\mathsf{AS}_{\mathsf{HP}}$ in this paper for the Hadamard product predicate Φ_{HP}; (e) the split accumulation scheme for our NARK for R1CS. 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 of 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.[7] Our implementation is modular: it takes as ingredients an implementation of any NARK,

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

an implementation of any accumulation scheme for that NARK, and constraints for the accumulation verifier, and produces a concrete PCD construction. This allows us, for example, to obtain a PCD instantiation based on our NARK for R1CS and its split accumulation scheme.

Evaluation for DL setting . 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 private-coin setup and more expensive curve cycles.)

We evaluate our implementation to answer two questions: (a) how efficient is recursion with split accumulation for our simple zkNARK for R1CS? (b) what is the constraint cost of split accumulation for $\mathsf{PC}_{\mathsf{Ped}}$ compared to atomic accumulation for $\mathsf{PC}_{\mathsf{IPA}}$? All our experiments are performed over the 255-bit Pallas curve in the Pasta cycle of curves [Hop20], which is used by real-world deployments.

– *Split accumulation for R1CS.* Our evaluation demonstrates that the cost of recursion for IVC with our split accumulation scheme for the simple NARK for R1CS is low, both with zero knowledge ($\sim 99 \times 10^3$ constraints) and without ($\sim 52 \times 10^3$ constraints). In fact, this cost is even lower than the cost of IVC based on highly efficient pairing-based circuit-specific SNARKs. Furthermore, like in the pairing-based case, this cost does not grow with the size of computation being checked. This is much better than prior constructions of IVC based on atomic accumulation for $\mathsf{PC}_{\mathsf{IPA}}$ in the DL setting, as we will see next.
– *Comparison of accumulation for PC schemes.* Several (S)NARKs are built from PC schemes, and the primary cost of recursion for these is determined by the cost of accumulation for the PC scheme. In light of this we compare the costs of two accumulation schemes:
  • the atomic accumulation scheme for the PC scheme $\mathsf{PC}_{\mathsf{IPA}}$ [BCMS20];
  • the new split accumulation scheme for $\mathsf{PC}_{\mathsf{Ped}}$.

  Our evaluation demonstrates that the constraint cost of the $\mathsf{AS}_{\mathsf{PC}}$ accumulation verifier is 8 to 20 times cheaper than that of the $\mathsf{AS}_{\mathsf{IPA}}$ accumulation verifier. In Figure 6 we report the asymptotic cost of $|\mathrm{V}|$ (the constraint cost of V) in $\mathsf{AS}_{\mathsf{IPA}}$, $\mathsf{AS}_{\mathsf{PC}}$, and $\mathsf{AS}_{\mathsf{R1CS}}$.[8]

[8] This comparison is meaningful because the cost of accumulating polynomial commitments provides a lower bound on the cost accumulating SNARKs that rely on these PC schemes.

We note that the cost of all the aforementioned accumulation schemes 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 the constraint cost of elliptic curve scalar multiplications should benefit all our schemes in a similar way.

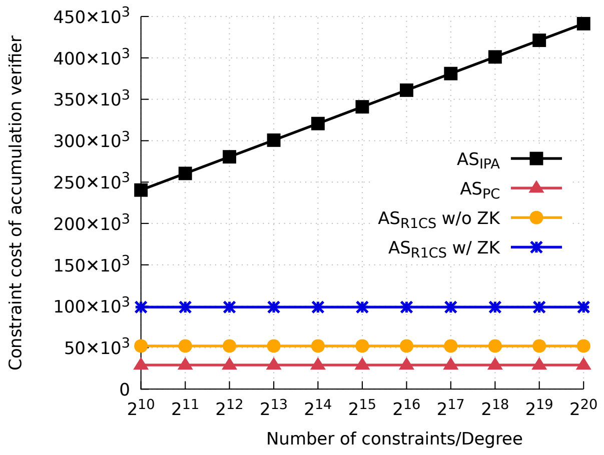

Fig. 6: Comparison of the constraint cost of the accumulation verifier V in $\mathsf{AS}_{\mathsf{IPA}}$, $\mathsf{AS}_{\mathsf{PC}}$, and $\mathsf{AS}_{\mathsf{R1CS}}$ when varying the number of constraints (for $\mathsf{AS}_{\mathsf{R1CS}}$) or the degree of the accumulated polynomial (for $\mathsf{AS}_{\mathsf{IPA}}$ and $\mathsf{AS}_{\mathsf{PC}}$) from 2^{10} to 2^{20}. Note that the cost of accumulating $\mathsf{PC}_{\mathsf{IPA}}$ and $\mathsf{PC}_{\mathsf{Ped}}$ is a lower bound on the cost of accumulating any SNARK built atop those, and this enables comparing against the cost of $\mathsf{AS}_{\mathsf{R1CS}}$.

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

[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.

[BCL+20] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner. “Proof-Carrying Data without Succinct Arguments”. In: *IACR Cryptol. ePrint Arch.* (2020). URL: `https://eprint.iacr.org/2020/1618`.

[BCMS20] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. “Proof-Carrying Data from Accumulation Schemes”. In: TCC ’20.

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

[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.

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

[BMM+19] B. Bünz, M. Maller, P. Mishra, N. Tyagi, and P. Vesely. “Proofs for Inner Pairing Products and Applications”. Cryptology ePrint Archive, Report 2019/1177.

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

[BN06] M. Bellare and G. Neven. “Multi-signatures in the plain public-Key model and a general forking lemma”. In: CCS ’06.

[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.

[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.

[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”. Cryptology ePrint Archive, Report 2013/513.

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

[Gro16] J. Groth. “On the Size of Pairing-Based Non-interactive Arguments”. In: EUROCRYPT ’16.

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

[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/`.

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

[Mina] O(1) Labs. “Mina Cryptocurrency”. https://minaprotocol.com/.

[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.

[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.