Halo Infinite: Proof-Carrying Data from Additive Polynomial Commitments

Dan Boneh[1], Justin Drake[2], Ben Fisch[1], and Ariel Gabizon[3]

[1] Stanford
[2] Ethereum Foundation
[3] AZTEC Protocol

Abstract. Polynomial commitment schemes (PCS) have recently been in the spotlight for their key role in building SNARKs. A PCS provides the ability to commit to a polynomial over a finite field and prove its evaluation at points. A *succinct* PCS has commitment and evaluation proof size sublinear in the degree of the polynomial. An *efficient* PCS has sublinear proof verification. Any efficient and succinct PCS can be used to construct a SNARK with similar security and efficiency characteristics (in the random oracle model).

Proof-carrying data (PCD) enables a set of parties to carry out an indefinitely long distributed computation where every step along the way is accompanied by a proof of correctness. It generalizes *incrementally verifiable computation* and can even be used to construct SNARKs. Until recently, however, the only known method for constructing PCD required expensive SNARK recursion. A system called *Halo* first demonstrated a new methodology for building PCD without SNARKs, exploiting an aggregation property of the *Bulletproofs* inner-product argument. The construction was *heuristic* because it makes non-black-box use of a concrete instantiation of the Fiat-Shamir transform. We expand upon this methodology to show that PCD can be (heuristically) built from any homomorphic polynomial commitment scheme (PCS), even if the PCS evaluation proofs are neither succinct nor efficient. In fact, the Halo methodology extends to any PCS that has an even more general property, namely the ability to aggregate linear combinations of commitments into a new succinct commitment that can later be opened to this linear combination. Our results thus imply new constructions of SNARKs and PCD that were not previously described in the literature and serve as a blueprint for future constructions as well.

1 Introduction

A polynomial commitment scheme (PCS) enables a prover to commit to a polynomial $f \in \mathbb{F}[X]$ of degree at most d. Later, given two public values $x, y \in \mathbb{F}$, the prover can convince a verifier that the committed polynomial f satisfies $y = f(x)$ and that f has degree at most d. This is done using a public coin evaluation protocol called Eval. The PCS is said to be *efficient* if the verifier runs in time $o(d \log |\mathbb{F}|)$, and is said to be *succinct* if the commitment string and the communication complexity of Eval is $o(d \log |\mathbb{F}|)$.

This important concept was first introduced by Kate, Zaverucha, and Goldberg (KZG) [40], and has emerged as a key tool for building succinct and efficient non-interactive argument systems called SNARKs [12]. A succinct and efficient PCS can be used to compile an information theoretic interactive proof system known as a *Polynomial Interactive Oracle Proof* [20] (PIOP), or equivalently *Algebraic Holographic Proofs* [28]), into a SNARK. There are many examples of *efficient* PIOPs for NP languages, where the verifier complexity is logarithmic or even constant in the size of the statement being proven. This construction paradigm led to several recent SNARK systems with improved characteristics, including very efficient pre-processing SNARKs with a universal trusted setup [45,28,32] or no trusted setup [20,30,48,41].

The original PCS, called the KZG PCS [40], is both efficient and succinct. It is based on pairings and requires a linear size reference string generated by a trusted setup (a recent improvement shrinks the size of the reference string [21]). Another PCS, called the Bulletproofs PCS [17,19], does not require pairings or a trusted setup, and is succinct, but is not efficient. Some schemes are both efficient and succinct and do not require a trusted setup: DARK [20] is based on groups of unknown order, and very recently Dory [43] uses pairing-based commitments and generalized inner-product arguments [21]. A post-quantum efficient and succinct PCS without trusted setup can be built using FRI [50,41,11]. In practice, these schemes all have very different performance profiles and properties.

A proof-carrying data (PCD) system [31,13] is a powerful primitive that is more general than a SNARK. Consider a distributed computation that runs along a path of t ordered nodes. The computation is defined by a function $F : \mathbb{F}^{\ell_1} \times \mathbb{F}^{\ell_2} \to \mathbb{F}^{\ell_1}$ in which node i takes two inputs: the output $z_{i-1} \in \mathbb{F}^{\ell_1}$ of node $(i-1)$, and a local input $\mathsf{loc}_i \in \mathbb{F}^{\ell_2}$. The node outputs $z_i = F(z_{i-1}, \mathsf{loc}_i) \in \mathbb{F}^{\ell_1}$. A PCD system enables each node to provide a proof to the next node which attests not only to the correctness of its local computation, but also to the correctness of all prior computations along the path. The work to produce/verify each local proof is proportional to the size of the local computation and is independent of the length of the path. A PCD system can be more generally applied to any distributed computation over a directed acyclic graph of nodes. An important performance metric of a PCD system is its *recursion threshold*: the minimum size complexity of F for which recursion is possible. PCD is currently being used in practice to construct a "constant-size blockchain" system [42,16], where the latest proof attests to the validity of all state transitions (i.e., transactions) in the blockchain history.

PCD systems generalize *incrementally verifiable computation* (IVC), proposed by Valiant [49], where a machine outputs a proof after each step of computation that attests to the correct history of computation steps. This can be used to construct SNARKs for *succinct bounded RAM programs*, which captures many programs in practice that have a small memory footprint relative to their running time. It is also theoretically sufficient for constructing preprocessing SNARKs for arithmetic circuits [7].

1.1 Contributions

We define several abstract properties of a PCS and show that these abstract properties are sufficient to construct powerful proof systems, including PCD and IVC. These abstract constructions give a general and unified approach to understanding recent PCD constructions. We show that the PCS schemes mentioned above satisfy some or all of our abstract properties. In some cases, instantiating our abstract proof systems with these PCS schemes leads to new proof systems that were not previously known. In fact, we could instantiate the PCS in two different ways from *any* collision-resistant linear hash function $h : \mathbb{F}^d \to \mathbb{G}$, one that optimizes for the size of proofs passed along nodes of the PCD, and the other that optimizes for prover time (i.e., the size of the recursive statement).[4]

We begin by defining an **additive** PCS as a simple refinement of a PCS, where the space of commitment strings form a computational group $\mathbb{G}$ under some binary operation add. Group elements must have representation size $\mathsf{poly}(\lambda)$ in terms of the security parameter λ of the PCS and add must run in time $\mathsf{poly}(\lambda)$. This means that it is possible to efficiently compute integer linear combinations of commitments. Moreover, a second requirement is that the prover can efficiently derive a valid opening string to open the linear combination of commitments to the same linear combination of the underlying committed polynomials. Because $\mathbb{G}$ is finite, the size of the linearly combined commitments is bounded, independent of the number of summands or sizes of the integer coefficients. A trivial way to impose a group structure on the commitment space of any PCS is to define $\mathbb{G}$ as the group of formal linear combinations of commitment strings, however, this trivial group is not bounded and therefore does not qualify the PCS as additive.

A useful property of an additive PCS is the ability to *aggregate* PCS evaluations, akin to signature aggregation. We define two flavors of **PCS aggregation schemes**: private and public. First, consider a tuple $(C, x, y) \in \mathbb{G} \times \mathbb{F}^2$, where C is a commitment to some polynomial $f \in \mathbb{F}^{(<d)}[X]$. We say that the prover has a witness for this tuple, if when the prover runs the Eval protocol with the verifier on input (C, x, y), the verifier accepts with probability one. A **(private) aggregation scheme** is an interactive protocol between a prover and a verifier where the public input known to both is ℓ tuples $(C_1, x_1, y_1), \ldots, (C_\ell, x_\ell, y_\ell) \in \mathbb{G} \times \mathbb{F}^2$, and the public output is a single tuple $(C^*, x^*, y^*) \in \mathbb{G} \times \mathbb{F}^2$. At the end of the protocol, the verifier is convinced that if the prover has a witness for (C^*, x^*, y^*), then it must also have witnesses for (C_i, x_i, y_i) for all $i \in [\ell]$. A private aggregation scheme is non-trivial if it is more efficient than running the Eval protocol on the $\ell + 1$ tuples. It is *efficient* if the verifier complexity is sublinear in the degree of the committed polynomials.

A **public aggregation scheme** enables a prover who does not know the witnesses for the ℓ input tuples to aggregate the non-interactive proofs for these tuples. This is also a two-party protocol where, for each $i \in [\ell]$, both parties receive a tuple $(C_i, x_i, y_i) \in \mathbb{G} \times \mathbb{F}^2$ and a corresponding non-interactive proof π_i.

[4] A homomorphism $h : \mathbb{Z}^d \to \mathbb{G}$ that is collision-resistant modulo p suffices, i.e. finding collisions where $\mathbf{x} \neq \mathbf{y} \bmod p$ is intractable.

The common output is a tuple $(C^*, x^*, y^*) \in \mathbb{G} \times \mathbb{F}^2$ for which the prover has a witness. The prover can subsequently produce a non-interactive proof for this output tuple. Informally, a valid proof for the output tuple demonstrates the validity of each input proof for the input tuples. As there is no information asymmetry between the two parties, the protocol is only interesting if the verifier does significantly less work than the prover.

A key theorem of this paper is that every additive PCS has an efficient private aggregation scheme. In fact, the theorem is more general. It is possible that a PCS is not additive, but there is still an efficient algorithm that takes as input a list of ℓ commitments along with ℓ integer coefficient weights, and outputs a new $\mathsf{poly}(\lambda)$-size commitment in $\mathbb{G}$ to the linear combination of the underlying committed input polynomials, along with a proof of correctness. We call this a **linear combination scheme** (LCS). The LCS is *efficient* if the verifier is sublinear in the degree of the committed polynomials. Moreover, if the LCS verifier complexity is asymptotically faster than running the Eval verifier ℓ times, then we call the PCS *linearly amortizable* because it allows for opening linear combinations of commitments with amortized efficiency gains. If the PCS is additive it suffices to compute linear combinations of commitments over $\mathbb{G}$ and no additional proof is required, hence every additive PCS is linearly amortizable. We prove that:

Theorem 1.1 (informal). *Every PCS that has an efficient linear combination scheme has an efficient private aggregation scheme. Every* succinct *additive PCS has an efficient public aggregation scheme.*

The formal statement of this result is in Theorem 4.2 and Theorem 5.2. As a concrete implication, we can take any linear collision-resistant hash function $h : \mathbb{F}^d \to \mathbb{G}$ and build a trivial PCS where the evaluation proof outputs the entire polynomial. Although this is not succinct, it is still additive and thus, as the theorem states, it has an efficient private aggregation scheme. Additionally, combining this hash function with a succinct protocol for proving pre-images of h would give a succinct additive PCS, which has an efficient public aggregation scheme. In fact, there exists a generic succinct protocol for proving pre-images of h (Section 5).

The first part of the result (private aggregation, Theorem 4.2) is based on a novel batched evaluation protocol for opening commitments to distinct polynomials at distinct points. Previously, standard batched evaluation techniques for homomorphic polynomial commitments included: (1) opening distinct commitments at the same point, and (2) opening a single commitment at multiple points. The first is accomplished by opening a random linear combination of the original commitments. The second is accomplished by interpolating a degree-n polynomial t over the n opening points such that the committed polynomial f is equal to t over the domain H of these points, and proving that $f - t$ is divisible by the zero polynomial z_H over this domain. The prover computes a commitment C_q to the quotient polynomial $q := \frac{f-t}{z_H}$ and proves that $q \cdot Z_H = (f - t)$ by opening C_q and C_f at a random challenge point. Both of these standard

batch evaluation protocols are single-round. We elegantly compose these two approaches to get a two-round protocol for batch opening *multiple* polynomials at *multiple* points. While the analysis of the standard batch evaluation protocol for a multiple commitments at a common point is based on the invertibility of a Vandermonde matrix, the analysis of our protocol relies on the invertibility of the Hadamard product of a random Vandermonde matrix with a square matrix of non-zero field elements (Lemma 4.7). The KZG instantiation of this protocol was presented in an earlier manuscript of our work [15].

Our result for public aggregation (Theorem 5.2) leverages the generic private aggregation scheme from Theorem 4.2 combined with a generic succinct proof of knowledge of the classical homomorphism pre-image problem (Section 5), which has its roots in the Bulletproofs protocol. Public aggregation is a factor $O(\log d)$ more costly (in communication size) than private aggregation.

Aggregation schemes have a number of important applications to constructing PCS-based SNARKs. First, aggregation schemes can be used for batch evaluation of polynomial commitments in order to reduce the work of the verifier (Section 4). Second, in Section 6 we discuss a fascinating and powerful application of PCS aggregation to recursive proof systems. This application generalizes a construction by Bowe, Grigg, and Hopwood called Halo [18], which was also formalized and generalized by Bünz et. al. [23].

PCD and IVC from PCS aggregation Suppose $F : \mathbb{F}^\ell \to \mathbb{F}^\ell$ and we wish to prove the correctness of t iterations of F, i.e. that $F^{(t)}(z_0) = z_t$. It turns out that given any succinct PCS with an efficient aggregation scheme, it is possible to construct an efficient non-interactive proof system for this type of statement whose proof size and verification complexity is proportional to the size and verification complexity of the PCS on polynomials of degree $|F|$, completely independent of t. As our results have shown, this includes any additive PCS and even non-additive schemes that have an efficient linear combination scheme. Most significantly, the PCS itself does not need to have efficient verification.

In fact, a PCS with an efficient aggregation scheme can be used to construct a PCD system. Not only does this mean that PCD, IVC, and preprocessing SNARKs can be constructed from any PCS with an efficient linear combination scheme, but we also expect this should lead to practical improvements over the prior proof bootstrapping techniques [7,30] whenever the verification complexity of the private aggregation is smaller than the verification complexity of Eval. We leave concrete performance analysis for future work, although follow up work [22] has already shown that the instantiation of PCD based on our private aggregation scheme using a simple Pedersen hash function achieves an order-of-magnitude reduction in the size of the recursive statement (reducing the recursion threshold accordingly).

Theorem 1.2 (informal). *PCD with proofs linear in the predicate size can be constructed from any PCS that has an efficient linear combination scheme. PCD with sublinear proofs can be constructed from any PCS with an efficient public aggregation scheme.*

In summary, our results pave the way for novel constructions of PCD, IVC, and SNARKs with new efficiency and security characteristics by directing the research effort towards PCS constructions that have the simple abstract additivity properties formalized in this paper. The constructions of PCD/IVC following this methodology do require a *heuristic* security assumption because they involve instantiating random oracles (more specifically, the Fiat-Shamir transform) with concrete hash functions. All known constructions of PCD/IVC require heuristic security (i.e., knowledge assumptions or concrete instantiations of random oracles) and there is evidence that this is inherent [29].

1.2 Related work

The construction of general purpose efficient SNARK systems is a hotly pursued topic. There are many examples of such proof systems that work for any NP relation [35,44,13,33,46,14,36,37,45,32,28,6,20,30,18,48]. In addition to the PCS constructions mentioned earlier, there is also a scheme by Bootle et. al. [17] that achieves $\sqrt{n}$ commitment size and Eval complexity based on any additively homomorphic commitment, and a similar lattice-based construction by Baum et. al. [3,2]. In Section 5 we describe a construction of a PCS from any collision-resistant homomorphism based on our succinct proof of homomorphism preimages (HPI) that has constant size commitment, logarithmic size proofs and linear verification time.[5] Attema and Cramer [1] described a generalization of Bulletproofs to proving linear forms of Pedersen committed vectors, which is a special case of our HPI protocol.

Constructions of IVC/PCD use *recursive composition*, which enables the prover to prove knowledge of a proof that the verification algorithm would accept. Until recently, constructions following this paradigm placed a complete description of the proof verifier inside the recursive statement. Thus, PCD was limited to proof systems where the verifier description is sublinear in the statement being proven (i.e., SNARKs) [49,13,10,30]. The Halo protocol [18,23] was the first construction of PCD from an underlying inefficient proof system (combining the Sonic PIOP [45] and the Bulletproofs PCS). There were two key ideas. The first was, in our terminology, a public aggregation scheme for the Bulletproofs PCS. The second was that the recursive statement can omit the inefficient portion of the proof system's verifier, i.e. the Eval verifier. The Eval proof inputs to a PCD step are aggregated along with the output Eval proofs, and the recursive statement only checks that aggregation was done correctly. This aggregates all Eval proofs into a single evaluation proof that is checked once at the end, amortizing the cost of Eval verification over the distributed computation length (i.e., recursion depth). Bünz et. al. [23] generalize this proof technique further using a primitive they call SNARK *accumulation schemes*. They also define PCS accumulation schemes, which can be combined with PIOP-based SNARKs to get a

[5] This can be combined with the technique of Bootle et. al. [17] to get a PCS with $\sqrt{n}$ commitment size, $\sqrt{n}$ verification time, and logarithmic proof size based on any collision-resistant homomorphism. We do not include the details in this work.

SNARK accumulation scheme. Our notion of public aggregation coincides with PCS accumulation. A small tweak to the definition of PCS accumulation we call *private* accumulation coincides with private aggregation and can be used to construct PCD with larger proofs (linear in the predicate size). Our results are thus perfectly complementary.

2 Preliminaries

Basic notations For an integer $n \geq 1$, we write $[n]$ to denote the set of integers $\{1, \ldots, n\}$. For any mathematical set $\mathcal{S}$ the notation $|\mathcal{S}|$ denotes the cardinality of $\mathcal{S}$. Unless specified otherwise, we use λ to denote the security parameter. We say a function $f(\lambda)$ is negligible in λ, denoted by $\mathsf{negl}(\lambda)$, if $f(\lambda) = o(1/\lambda^c)$ for all $c \in \mathbb{N}$. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We use $\mathsf{poly}(\lambda)$ to denote a quantity whose value is bounded by a fixed polynomial in λ. For a field $\mathbb{F}$, we use $\mathbb{F}^{(<d)}[X]$ for the set of polynomials in $\mathbb{F}[X]$ of degree at most d. We use $\{0,1\}^*$ to denote binary strings of arbitrary length and ε to denote the empty string. We may use the notations $\mathbb{F}_p$ and $\mathbb{Z}_p$ interchangeably to denote the unique prime field of characteristic p. For modular arithmetic, we use the notation $a \equiv b \pmod{n}$ to denote that integers $a, b \in \mathbb{Z}$ are equivalent modulo $n \in \mathbb{Z}$. The notation $a \bmod n$ denotes the unique integer $b \in [0, n)$ such that $a \equiv b \pmod{n}$.

For an abstract group, $\mathbb{G}$ denotes the set of elements in the group, and for any $g_1, g_2 \in \mathbb{G}$ the element $g_1 + g_2$ is the result of applying the binary operation to g_1 and g_2. The inverse of $g \in \mathbb{G}$ is denoted $-g$ and $g_1 - g_2 := g_1 + (-g_2)$. For any $n \in \mathbb{N}$ and $g \in \mathbb{G}$ the element $n \cdot g$ is defined as adding n copies of g. For $n \in \mathbb{Z}$, $n < 0$, then $n \cdot g$ is defined as $-(|n| \cdot g)$. The group $\mathbb{G}$ is called a *computational group* if there exist efficient algorithms for implementing the addition and inversion operations.

Proofs of knowledge An NP relation $\mathcal{R}$ is a subset of strings $x, w \in \{0,1\}^*$ such that there is a decision algorithm to decide $(x, w) \in \mathcal{R}$ that runs in time polynomial in $|x|$ and $|w|$. The language of $\mathcal{R}$, denoted $\mathcal{L}_R$, is the set $\{x \in \{0,1\}^* : \exists w \in \{0,1\}^* \; s.t. \; (x, w) \in \mathcal{R}\}$. The string w is called the *witness* and x the *instance*. An **interactive proof of knowledge** for an NP relation $\mathcal{R}$ is a special kind of two-party interactive protocol between a prover denoted $\mathcal{P}$ and a verifier denoted $\mathcal{V}$, where $\mathcal{P}$ has a private input w and both parties have a common public input x such that $(x, w) \in \mathcal{R}$. Informally, the protocol is *complete* if $\mathcal{P}(w)$ always causes $\mathcal{V}(pp, x)$ to output 1 for any $(x, w) \in \mathcal{R}$. The protocol is *knowledge sound* if there exists an extraction algorithm $\mathcal{E}$ called the *extractor* such that for every x and adversarial prover $\mathcal{A}$ that causes $\mathcal{V}(pp, x)$ to output 1 with non-negligible probability, $\mathcal{E}$ outputs w such that $(x, w) \in \mathcal{R}$ with overwhelming probability given access[6] to $\mathcal{A}$.

[6] The extractor can run $\mathcal{A}$ for any specified number of steps, inspect the internal state of $\mathcal{A}$, and even rewind $\mathcal{A}$ to a previous state.

Definition 2.1 (Interactive Proof with Efficient[7] Prover).

Let $\mathsf{Setup}(\lambda)$ *denote a non-interactive setup algorithm that outputs public parameters* pp *given a security parameter* λ*. Let* $\Pi\big(\mathcal{P}(w), \mathcal{V}(pp, x)\big)$ *denote a two-party interactive protocol between* $\mathcal{P}$ *and* $\mathcal{V}$*, where* $\mathcal{P}$ *has private input* w *and* $\mathcal{V}$ *has the common public input* (pp, x)*. Let* $\langle \mathcal{P}(w), \mathcal{V}(pp, x)\rangle$ *be a random variables that is the output of* $\mathcal{V}$*. All algorithms run in time* $\mathsf{poly}(\lambda, |pp|, |x|, |w|)$*. The pair* (Setup, Π) *is called a* proof of knowledge *for relation* $\mathcal{R}$ *if for all non-uniform adversaries* $\mathcal{A}$ *the following properties hold:*

- *Perfect Completeness.*

$$\Pr\left[\begin{array}{c} (x,w) \notin \mathcal{R} \text{ or} \\ \langle \mathcal{P}(w), \mathcal{V}(pp,x)\rangle = 1 \end{array} : \begin{array}{c} pp \leftarrow \mathsf{Setup}(\lambda) \\ (x,w) \leftarrow \mathcal{A}(pp) \end{array}\right] = 1$$

- *Knowledge soundness [4] There exists a probabilistic oracle machine* $\mathcal{E}$ *called the* extractor *such that for every adversarial interactive prover algorithm* $\mathcal{A}$ *that is only given the public inputs* (pp, x) *and every* $x \in \mathcal{L}_R$ *the following holds: if* $\langle \mathcal{A}(\cdot), \mathcal{V}(pp, x)\rangle = 1$ *with probability* $\epsilon(x) > \mathsf{negl}(\lambda)$ *then* $\mathcal{E}^{\mathcal{A}}(pp, x)$ *with oracle access to* $\mathcal{A}$ *runs in time* $\mathsf{poly}(|x|, \lambda)$ *and outputs* w *such that* $(x, w) \in R$ *with probability* $1 - \mathsf{negl}(\lambda)$*.*

Forking lemmas The following "forking lemma" is helpful for proving knowledge soundness of multi-round public coin interactive protocols over an exponentially large challenge space (i.e., where each verifier message is a uniform sample from a space $\mathcal{X}$ that has size at least 2^{λ}). It says that if the adversary succeeds with non-negligible probability $\epsilon = 1/\mathsf{poly}(\lambda)$, then there is an $O(\mathsf{poly}(\lambda))$-time algorithm for generating a tree of accepting transcripts defined as follows. For an r-round protocol, an $(n_1, ..., n_r)$**-tree of accepting transcripts** for $n_i \geq 0$ is a tree where (i) every node v of the tree corresponds to a partial transcript tr_v, (ii) every level-i node v has n_i children nodes that correspond to continuations of tr_v with distinct ith round challenges, and (iii) every leaf node corresponds to a full transcript in which the verifier accepts. More generally, the property that each pair of challenges on sibling nodes are distinct can be replaced with any property $\pi : \mathcal{X}^2 \to \{0, 1\}$ which outputs 1 on a random pair of challenges with overwhelming probability. This forking lemma generalizes a similar lemma by Bootle *et al.* [17]. We provide a proof in the full version.

Lemma 2.2 (Forking Lemma). *Let* $(\mathcal{P}, \mathcal{V})$ *be an* r*-round public-coin interactive proof system and* $\mathcal{A}$ *an adversary that runs in expected time* $t_{\mathcal{A}}$ *such that* $\langle \mathcal{A}(\cdot), \mathcal{V}(pp, x)\rangle = 1$ *with probability* ϵ *on public input* x *and public parameters* pp*. Let* $\{\pi_i\}_{i=1}^{r}$ *be a set of properties* $\pi_i : \mathcal{X}^2 \to \{0, 1\}$ *such that*

[7] A classical interactive proof does not require the prover to be efficient. However, our definition of an interactive proof with efficient prover should also not be confused with an interactive *argument*, which only requires soundness against efficient adversaries. In our definition, the prover is required to be efficient for correctness, but soundness must hold against adversaries with unbounded running time.

$\forall_i\ Pr[\pi(x_1, x_2) = 1 : x_1, x_2 \xleftarrow{\$} \mathcal{X}] > 1 - \mathsf{negl}(\lambda)$. *If* $r \in O(\log \lambda)$ *then for any constants* $n_1, ..., n_r \in \mathbb{N}$ *there exists an algorithm* $\mathcal{T}$ *that runs in time* $\mathsf{poly}(\lambda) \cdot (t_{\mathcal{A}}/\epsilon)$ *and with probability at least* $1 - \mathsf{negl}(\lambda)/\epsilon^2$ *outputs an* $(n_1, ..., n_r)$*-tree of accepting transcripts such that for* $i \in [1, r]$ *all pairs of sibling-node challenges* $x_1, x_2 \in \mathcal{X}$ *at level* i *satisfy* $\pi_i(x_1, x_2) = 1$.

Fiat-Shamir tranform The Fiat-Shamir transform preserves knowledge soundness for any constant-round public-coin interactive proof in the random oracle model, i.e. when the "hash function" is modeled as a random oracle [34,47]. The interactive protocol must have a negligible soundness error. More generally, Fiat-Shamir preserves knowledge soundness for multi-round interactive proofs that satisfy a property called *state restoration soundness* [9], also equivalent to *round-by-round soundness* [24,38]. There are also special classes of constant-round protocols for which the Fiat-Shamir transform can be instantiated using correlation-intractable hash functions [39,25,24], or even simpler non-cryptographic hash functions [26]. In general, the security of the Fiat-Shamir transform applied to a knowledge-sound interactive proof system using a concrete hash function is heuristic. There are known examples where the transform fails to preserve soundness.

Definition 2.3. *A knowledge-sound interactive proof system* $(\mathcal{P}, \mathcal{V})$ *is* ***FS compatible*** *if there exists a hash family* $\mathcal{H}$ *such that the non-interactive proof system* $(\mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$ *obtained from applying Fiat-Shamir using an explicit hash sampled from* $\mathcal{H}$ *is knowledge-sound.*

Zero Knowledge An interactive proof satisfies **honest verifier zero-knowledge** (HVZK) if there exists a simulator that does not have access to the prover's private witness yet can produce convincing transcripts between the prover and an honest verifier that are statistically indistinguishable from real transcripts. The Fiat-Shamir transform compiles public-coin proofs that have HVZK into non-interactive proofs that have statistical zero-knowledge (for possibly malicious verifiers).

2.1 Polynomial Commitment Scheme (PCS)

A **polynomial commitment scheme**, or PCS, is a triple of PPT algorithms, Setup, Commit, and Verify along with an evaluation protocol Eval, where:

- $\mathsf{Setup}(\lambda, d) \to pp$ a deterministic algorithm that outputs public parameters pp for committing to polynomials of degree d. The parameters pp include a specification of an abelian commitment group $\mathbb{G}$, as defined below.
- $\mathsf{Commit}(pp, f) \to (\mathsf{C}, \mathsf{open})$ outputs a commitment $\mathsf{C} \in \mathbb{G}$ to the polynomial $f \in \mathbb{F}^{(<d)}[X]$ and an opening "hint" $\mathsf{open} \in \{0,1\}^*$.
- $\mathsf{Verify}(pp, f, \mathsf{open}, \mathsf{C})$ checks the validity of an opening hint open for a commitment $\mathsf{C} \in \mathbb{G}$ to the polynomial $f \in \mathbb{F}^{(<d)}[X]$ and outputs 1 (accept) or 0 (reject).

– $\mathsf{Eval}\big(\mathcal{P}(f, \mathsf{open}), \mathcal{V}(pp, \mathsf{C}, z, y)\big) \to (\bot, b)$ is a public-coin interactive protocol between a prover who has the private input (f, open) for $f \in \mathbb{F}^{(<d)}[X]$ and a verifier who has the common public input pp and $(\mathsf{C}, z, y) \in \mathbb{G} \times \mathbb{F}^2$. The verifier outputs $b \in \{0, 1\}$ and the prover has no output. The purpose of the protocol is to convince the verifier that $f(z) = y$ and $\deg(f) < d$.

All the algorithms run in time polynomial in λ and d. Furthermore, a scheme is **correct** if for all polynomials $f \in \mathbb{F}^{(<d)}[X]$ and all points $z \in \mathbb{F}$, with probability 1 the verification $\mathsf{Verify}(pp, f, \mathsf{open}, \mathsf{C})$ outputs 1 and likewise $\mathcal{V}$ outputs 1 in interaction with $\mathcal{P}$ in the Eval protocol on valid inputs.

Commitment group A **commitment group** $\mathbb{G}$ is a computational group accompanied by two PPT algorithms: if open_f and open_g are opening hints for commitments C_f and C_g to polynomials $f, g \in \mathbb{F}^{(<d)}[X]$, then $\mathsf{add}^*\big(\mathsf{open}_f, \mathsf{open}_g\big)$ outputs an opening for $\mathsf{C}_f + \mathsf{C}_g$ to the polynomial $f+g$ and $\mathsf{invert}^*(\mathsf{open}_f)$ outputs an opening for $-\mathsf{C}_f$ to the polynomial $-f$. This is a non-standard part of the PCS definition and may appear overly restrictive. However, it does not reduce the generality of a PCS. The default way to define $\mathbb{G}$ is the space of formal linear combinations of commitments to elements of $\mathbb{F}^{(<d)}[X]$. The default add^* would simply be concatenation.

Explicit specification of $\mathbb{G}$, add^*, and invert^* is convenient for defining the additivity properties of a PCS discussed in Section 3. This also serves to highlight how additivity is merely a refinement on $\mathbb{G}$. The existence of $\mathbb{G}$, add^*, and invert^* is not a distinguished property on its own.

Efficiency/Succinctness If the Eval verifier runs in time $o(d)$, i.e. sublinear in the degree of the committed polynomial, then the PCS is called **efficient**. If both the size of commitments and communication complexity of the Eval protocol are $o(d)$ then the scheme is called **succinct**.

A PCS could be succinct and not efficient. One example is a PCS based on the Bulletproofs system [17,19]. Some PCS applications may have stricter efficiency/succinctness requirements (e.g., $\mathsf{polylog}(d)$ length or run time). A non-succinct PCS is only interesting if it is hiding, and only distinguished from a regular hiding commitment scheme if it has a zero-knowledge evaluation protocol (defined below).

Non-interactive Eval An interactive PCS Eval protocol may be compiled into a non-interactive Eval proof via the Fiat-Shamir transform. We use the notation $\pi \leftarrow \mathsf{NI\text{-}Eval}\big(pp, f, \mathsf{open}, C, x, y\big)$ and $b \leftarrow \mathcal{V}_{\mathsf{Eval}}(pp, \pi, C, x, y)$. The PCS Eval may already be non-interactive (e.g., KZG [40]) in which case Fiat-Shamir is not needed.

Security properties The scheme's algorithms $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify})$ must be binding as a standard commitment scheme. Furthermore, the protocol Eval should be complete and a proof of knowledge. Informally, this means that any successful prover in the Eval protocol on common input (C, z, y) must *know* a

polynomial $f(X) \in \mathbb{F}^{(<d)}[X]$ such that $f(z) = y$ and C is a commitment to $f(X)$. The two of these properties together also imply that the scheme is *evaluation binding*, which means that no efficient adversary can output pp and two pairs (C, z, y) and (C, z, y') where $y \neq y'$, and then succeed in Eval on both pairs (C, z, y) and (C, z, y'). The requirement that Eval is a proof of knowledge is stronger than evaluation binding alone, but is necessary for the application to SNARKs.

Definition 2.4 (Binding PCS). *A PCS is* ***binding*** *if for all PPT adversaries* $\mathcal{A}$*:*

$$\Pr\left[b_0 = b_1 = 1 \wedge f_0 \neq f_1 \ : \begin{array}{l} pp \leftarrow \mathsf{Setup}(\lambda, d) \\ (f_0, \mathsf{open}_0, \mathsf{C}_0, f_1, \mathsf{open}_1, \mathsf{C}_1) \leftarrow \mathcal{A}(pp) \\ b_0 \leftarrow \mathsf{Verify}(pp, f_0, \mathsf{open}_0, \mathsf{C}_0) \\ b_1 \leftarrow \mathsf{Verify}(pp, f_1, \mathsf{open}_1, \mathsf{C}_1) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Definition 2.5 (Knowledge soundness). *A PCS has* ***knowledge soundness*** *if for all* pp *output by* $\mathsf{Setup}(\lambda, d)$ *and* $d \in \mathbb{N}$*, the interactive public-coin protocol* Eval *is a* proof of knowledge *for the NP relation* $\mathcal{R}_{\mathsf{Eval}}(pp, d)$ *defined as follows:*

$$\mathcal{R}_{\mathsf{Eval}}(pp, d) = \left\{ \langle (\mathsf{C}, z, y), (f, \mathsf{open}) \rangle : \begin{array}{l} f \in \mathbb{F}^{(<d)}[X] \ \wedge \ f(z) = y \\ \mathsf{Verify}(pp, f, \mathsf{open}, \mathsf{C}) = 1 \end{array} \right\}$$

Hiding and Zero Knowledge A PCS scheme **hiding** if it satisfies the standard definition of a hiding commitment, i.e. commitments to distinct polynomials are statistically indistinguishable. A PCS scheme is **zero-knowledge** if its Eval protocol is a public-coin HVZK interactive proof for the relation $\mathcal{R}_{\mathsf{Eval}}(pp, d)$.

Bounded witness ZK Eval The regular definition of a zero-knowledge PCS scheme requires that the Eval protocol is a zero-knowledge proof for the relation $\mathcal{R}_{\mathsf{Eval}}(pp, d)$. This means that Eval cannot leak any information at all about the prover's witness (f, open) for the commitment open, other than the public statements $f(z) = y$, $f \in \mathbb{F}^{(<d)}[X]$, and open is valid. Some schemes, such as DARK [20], do not satisfy this strongest definition of zero-knowledge, but rather satisfy a weaker zero-knowledge PCS property that is generally sufficient in practice. Let $\mathbb{H}$ be a set containing all possible opening hints and let $\mathcal{N} : \mathbb{H} \to \mathbb{R}$ be any non-negative efficiently computable function. Let $\{\mathsf{Eval}(B) : B \in \mathbb{R}\}$ denote a family of evaluation protocols that take an extra parameter $B \in \mathbb{R}$. A PCS satisfies **bounded witness zero-knowledge** for $\mathcal{N}$ if $\mathsf{Eval}(B)$ is a public-coin HVZK interactive proof for the modified relation:

$$\mathcal{R}_{\mathsf{Eval}}(pp, d, \mathcal{N}, B) = \left\{ \langle (\mathsf{C}, z, y), (f, \mathsf{open}) \rangle : \begin{array}{l} f \in \mathbb{F}^{(<d)}[X] \ \wedge \ f(z) = y \ \wedge \ \mathcal{N}(\mathsf{open}) \leq B \\ \mathsf{Verify}(pp, f, \mathsf{open}, \mathsf{C}) = 1 \end{array} \right\}$$

"Relaxed" PCS openings For any PCS scheme, the Verify function can be relaxed such that it will accept an opening of the commitment $t \cdot \mathsf{C}_f$ to the polynomial $h = t \cdot f$ for a integer $t \in \mathbb{Z}$ as a valid opening of C_f to the polynomial f.

Lemma 2.6. *Let $\mathcal{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify}, \mathsf{Eval})$ denote a PCS for polynomials over a field $\mathbb{F}$ of characteristic p. If the algorithm Verify is replaced with an algorithm Verify^* that accepts $(f, (t, \mathsf{open}), \mathsf{C})$ if and only if $t \neq 0 \bmod p$ and Verify accepts $(h, \mathsf{open}, t \cdot \mathsf{C})$ where $h = t \cdot f$, then the new PCS is still binding.*

Proof. Suppose an adversary outputs openings $(f_1, (t_1, \mathsf{open}_1))$ and $(f_2, (t_2, \mathsf{open}_2))$ to a commitment C such that Verify^* accepts both and $f_1 \neq f_2$. This implies that Verify accepts both $(h_1, \mathsf{open}_1, t_1 \cdot \mathsf{C})$ and $(h_2, \mathsf{open}_2, t_2 \cdot \mathsf{C})$ where $h_1 = t_1 \cdot f_1$ and $h_2 = t_2 \cdot f_2$. Using the add^* operation, it would be possible to compute valid openings of $t_1 t_2 \cdot \mathsf{C}$ to both $t_1 h_2 = t_1 t_2 \cdot f_2$ and $t_2 h_1 = t_1 t_2 \cdot f_1$. Since $f_1 \neq f_2$ it follows that $t_1 h_2 \neq t_2 h_1$. Thus, this would contradict the binding property of the original PCS. □

Lemma 2.7. *Given two vectors of commitments $\mathbf{C}, \mathbf{C}^* \in \mathbb{G}^n$, a system of equations $\mathbf{AC} = \mathbf{C}^*$ for an integer matrix $\mathbf{A} \in \mathbb{Z}^{n \times n}$ that is invertible over $\mathbb{F}_p$, and a vector of openings of $\mathbf{C}^*$ to a vector of polynomials $\mathbf{f}^* = (f_1^*, ..., f_n^*) \in (\mathbb{F}^{(<d)}[X])^n$, there is an efficient algorithm to derive polynomials $\mathbf{f} = (f_1, ..., f_n) \in (\mathbb{F}^{(<d)}[X])^n$, an integer $t \in \mathbb{Z}$ such that $t \neq 0 \bmod p$, and openings for each $t \cdot \mathbf{C}_i$ to the polynomial $t \cdot f_i \bmod p$ such that $\mathbf{A} \cdot \mathbf{f} \equiv \mathbf{f}^* \pmod{p}$.*

Proof. Since $det(\mathbf{A}) \neq 0$, the matrix $\mathbf{A}$ is invertible over $\mathbb{Q}$. Let $\mathbf{A}^{-1}$ denote the inverse of $\mathbf{A}$ over $\mathbb{Q}$ and let $\mathbf{I}$ denote the identity matrix over $\mathbb{Z}$. Set $\mathbf{L}$ to be the matrix obtained by clearing the denominators of $\mathbf{A}^{-1}$, i.e. $\mathbf{L} = t \cdot \mathbf{A}^{-1}$ where $t \neq 0$ is the least common multiple of all denominators of the rational entries of $\mathbf{A}^{-1}$. We have $t \cdot \mathbf{C} = \mathbf{L} \cdot \mathbf{A} \cdot \mathbf{C} = \mathbf{L} \cdot \mathbf{C}^*$. From each linear combination of $\mathbf{C}^*$, we use add^* to derive an opening of $t \cdot \mathbf{C}_i$ to a polynomial $g_i = \langle \mathbf{L}_i, \mathbf{f}^* \rangle \in \mathbb{F}[X]$. Let $\mathbf{g} = (g_1, ..., g_n)$. Finally, solve for the vector of polynomials $\mathbf{f}$ such that $\mathbf{A} \cdot \mathbf{f} = \mathbf{f}^*$ by computing $\mathbf{A}^{-1} \bmod p$. Note that $\mathbf{L} \cdot \mathbf{A} \cdot \mathbf{f} = t \cdot \mathbf{f} = \mathbf{L} \cdot \mathbf{f}^*$. Thus, $t f_i = g_i$, for which we have a valid opening of $t \cdot \mathbf{C}_i$. □

3 Additive polynomial commitments

This section defines an **additive** PCS as a simple refinement of a PCS, where the group of commitments is a computational group of bounded size. Recall that in our definition from Section 2.1, a PCS includes a specification of a family of commitment groups indexed by the parameters (λ, d). We remarked that this is without loss of generality.

Definition 3.1. *A PCS is **additive** if every abelian commitment group $\mathbb{G}_{\lambda,d}$ determined by the public parameters $pp \xleftarrow{\$} \mathsf{Setup}(\lambda, d)$ is a computational group of size at most $2^{poly(\lambda)}$. An additive PCS for polynomials in $\mathbb{F}^{(<d)}[X]$ is **additively succinct** if the size of $\mathbb{G}_{\lambda,d}$ is $o(|\mathbb{F}|^d)$.*

There may be a group $\mathbb{G}$ that satisfies the size constraints of Definition 3.1 but does not qualify as a commitment group but the add^* operation only works for a bounded number of operations. Examples include DARK and lattice-based schemes [20,2]. We call them *bounded additive*.

Definition 3.2. *A PCS over a field $\mathbb{F}$ is* ***homomorphic*** *if for any $\lambda, d \in \mathbb{N}$ the parameters $pp \leftarrow \mathsf{Setup}(\lambda, d)$ determine two computational groups $(\mathbb{G}, \mathbb{H})$ and two polynomial time computable homomorphisms $\phi : \mathbb{H} \to \mathbb{G}$ and $\chi : \mathbb{H} \to \mathbb{F}^{(<d)}[X]$ such that the algorithm $\mathsf{Verify}(pp, f, \mathsf{C}, \mathsf{open})$ returns 1 if and only if $\phi(\mathsf{open}) = \mathsf{C}$ and $\chi(\mathsf{open}) = f$.*

We call $\mathbb{H}$ the "hint" group. For a homomorphic PCS to be binding, the homomorphism $\phi : \mathbb{H} \to \mathbb{G}$ must be collision resistant over equivalence classes in $\mathbb{H}/ker(\chi)$ (i.e., finding $x_1, x_2 \in \mathbb{H}$ such that $\chi(x_1) \neq \chi(x_2)$ and $\phi(x_1) = \phi(x_2)$ must be hard).

An additive PCS gives a homomorphic PCS. Any additive PCS over a prime field $\mathbb{F} = \mathbb{F}_p$ and commitment group $\mathbb{G}$, can be efficiently transformed into a non-hiding homomorphic PCS with the same commitment group $\mathbb{G}$. The transformation maintains succinctness if the PCS is additively succinct. The new commitment algorithm will give a homomorphism $\phi : \mathbb{Z}^d \to \mathbb{G}$.

3.1 Linear combination schemes

It is possible that a PCS is not additive, yet there is still an efficient scheme to linearly combine polynomial commitments into a succinct aggregate commitment and later open this at points.

Definition 3.3 (Linear Combination Scheme). *A linear combination scheme for a PCS with commitment group $\mathbb{G}$ is a public-coin interactive protocol $\mathsf{LinCombine}$ defined as follows. Given any $\mathbf{f} \in \mathbb{F}^{(<d)}[X]^\ell$, $\boldsymbol{\alpha} \in \mathbb{F}^\ell$, $\mathbf{C} \in \mathbb{G}^\ell$, and a vector of openings $\mathsf{open} = (\mathsf{open}_1, ..., \mathsf{open}_\ell)$ such that $\mathsf{Verify}(pp, f_i, \mathsf{open}_i, \mathsf{C}_i) = 1$ for all $i \in [\ell]$, the protocol $\mathsf{LinCombine}$ does:*

$$\mathsf{LinCombine}\big(\mathcal{P}(\mathbf{f}, \mathbf{open}), \mathcal{V}(pp, \mathbf{C}, \boldsymbol{\alpha})\big) \to (\mathsf{open}^*, (C^*, b)).$$

The public output is $(C^*, b) \in \mathbb{G} \times \{0, 1\}$ where $b \in \{0, 1\}$ indicates success or failure. The private output is an opening open^* for C^* to the polynomial $\sum_{i=1}^{\ell} \alpha_i \cdot f_i$. As for the security, $\mathsf{LinCombine}$ composed with Eval on the output C^* is a proof of knowledge for the relation:

$$\mathcal{R}_{\mathsf{LinComb}}(pp, d) = \left\{ \left\langle (\mathbf{C}, \mathsf{C}^*, \boldsymbol{\alpha}), (f, \mathsf{open}, \mathsf{open}^*) \right\rangle : \begin{array}{l} (\mathsf{C}^*, (f, \mathsf{open}^*)) \in \mathcal{R}_{\mathsf{Eval}}(pp, d) \\ (\mathsf{C}, (f, \mathsf{open})) \in \mathcal{R}_{\mathsf{Eval}}(pp, d) \\ \mathsf{C} = \sum_i \alpha_i \cdot \mathsf{C}_i \end{array} \right\}$$

The trivial linear combination scheme simply returns the linear combination of the input commitments over the commitment group. This clearly satisfies the security definition because $\mathsf{C}^* = \mathsf{C}$ in this case. When a scheme is additively succinct then the trivial linear combination scheme is the most natural to use. The purpose of a non-trivial $\mathsf{LinCombine}$ is to return a C^* that is more succinct than C. We call the scheme **size-optimal** if the aggregate commitment size is bounded by the worst case size of commitments to polynomials of degree d.

We remark that every PCS has a relatively uninteresting generic size-optimal linear combination protocol. The prover can simply compute a fresh commitment C^* to $f = \sum_{i=1}^{\ell} \alpha_i \cdot f_i$ and run $\ell + 1$ instances of Eval on C^* and each C_i at a common random point ρ selected by the verifier. The verifier can check the linear relation between the opening value of C^* at ρ and opening values of the list of C_i at ρ. This satisfies the security definition simply because the $\mathsf{LinCombine}$ protocol itself is a proof of knowledge of an opening of C^* to f and each C_i to f_i such that $f = \sum \alpha_i \cdot f_i$. A linear combination scheme is interesting when it is more efficient than this generic one.

We say that a linear combination scheme is **efficient** if the verifier complexity in the protocol $\mathsf{LinCombine}$ is sublinear in the maximum degree of the input polynomials.

3.2 PCS examples and their additive properties

The table below summarizes the properties of several schemes. All major PCS constructions have efficient linear combination schemes, which beat the generic one. The linear combination scheme (LCS) amortization ratio (column 3) indicates the ratio of the communication/verification complexity of using the LCS to prove the evaluation of a linear combination (i.e. run Eval on the output of the LCS) versus the generic protocol of running ℓ separate instances of Eval. This ratio is most relevant for the efficiency of batch evaluation (Section 4). The complexity ratio of the LCS verifier to the Eval verifier (column 5) is most relevant for the efficiency[8] of proof recursion (i.e., IVC/PCD) discussed in Section 6. The parameter ℓ is the number of polynomial commitments being linearly combined and d is their maximum degree.

	additive	LCS amortization	$\lvert\mathcal{V}_{\mathsf{LinCombine}}\rvert$	$\frac{\lvert\mathcal{V}_{\mathsf{LinCombine}}\rvert}{\lvert\mathcal{V}_{\mathsf{Eval}}\rvert}$
Bulletproofs	yes	$1/\ell$	$O_\lambda(\ell)$	$\ell/\Omega(d)$
Dory	yes	$1/\ell$	$O_\lambda(\ell)$	$\ell/\Omega(\log d)$
KZG	yes	$1/\ell$	$O_\lambda(\ell)$	$\ell/\Omega(1)$
DARK	bounded	$1/\ell$	$O_\lambda(\ell)$	$\ell/\Omega(\log d)$

FRI: a non-additive PCS The Fast Reed-Solomon IOP of Proximity (FRI) [5] is a protocol for proving that a committed vector in $\mathbb{F}^n$ is δ-close (in relative Hamming distance) to a Reed-Solomon (RS) codeword. FRI can be used to construct a PCS that is post-quantum.

The FRI PCS is not additive by Definition 3.1, but it does have a protocol for opening a *random* linear combination that achieves amortized efficiency ratio of $\frac{1}{\ell} + \frac{1}{\Omega(\log d)}$ over ℓ commitments, which can also be extended to achieve amortized batch evaluation (e.g., Algorithm 8.2 of Aurora [8]).

[8] The asymptotic ratio for KZG hides the fact that $\mathcal{V}_{\mathsf{Eval}}$ involves a pairing operation while $\mathcal{V}_{\mathsf{LinCombine}}$ has only $\ell \cdot \lambda$ curve additions and thus is cheaper for small ℓ.

4 Batch Evaluation and Private Aggregation

For the purpose of this section $\mathbb{F} := \mathbb{F}_p$, for some prime number p. It may be possible to generalize our results to work over extension fields, but that is beyond scope.

The batch evaluation problem Let $f_1, \ldots, f_\ell \in \mathbb{F}^{(<d)}[X]$ and let C_i be a commitment to f_i for $i \in [\ell]$. The verifier has pp and $\mathsf{C}_1, \ldots, \mathsf{C}_\ell$. For each $i \in [\ell]$ the verifier also has $(z_{i,1}, y_{i,1}), \ldots, (z_{i,\ell_i}, y_{i,\ell_i}) \in \mathbb{F}^2$. The prover wants to convince the verifier that $f_i(z_{i,j}) = y_{i,j}$ for all $i \in [\ell]$ and $j \in [\ell_i]$.

An alternative formulation of the batch evaluation problem is as follows. For each $i \in [\ell]$:

- let $\Omega_i = \{z_{i,1}, \ldots, z_{i,\ell_i}\} \subseteq \mathbb{F}$, and
- let t_i be the unique degree-$(\ell_i - 1)$ polynomial that satisfies $t_i(z_{i,j}) = y_{i,j}$ for all $j \in [\ell_i]$.

The verifier has $(\mathsf{C}_i, \Omega_i, t_i)$ for $i \in [\ell]$. The batch evaluation problem is for the prover to convince the verifier that $f_i(x) = t_i(x)$ for all $i \in [\ell]$ and $x \in \Omega_i$. We will use this formulation of the problem from now on.

When all the polynomials t_i in the batch evaluation problem are identically zero (i.e., $t_i \equiv 0$ for all $i \in [\ell]$) then the problem is called *batch zero testing*.

Aggregation scheme We define PCS proof *aggregation*, akin to signature aggregation. The aggregation of tuples $(C_1, x_1, y_1), ..., (C_\ell, x_\ell, y_\ell)$ is a single tuple (C^*, x^*, y^*) such that running Eval to open $C^* \in \mathbb{G}$ at point $x^* \in \mathbb{F}$ to $y^* \in \mathbb{F}$ suffices to open each $C_i \in \mathbb{G}$ at $x_i \in \mathbb{F}$ to $y_i \in \mathbb{F}$. Aggregation enables batch evaluation, as shown in Figure 1.

Definition 4.1 (Aggregation). *Let $\mathcal{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify}, \mathsf{Eval})$ denote a PCS with commitment group $\mathbb{G}$. An aggregation scheme for $\mathcal{PCS}$ is a public-coin interactive protocol $\mathsf{Aggregate}$ with public inputs $\mathbf{C} = (C_1, ..., C_\ell) \in \mathbb{G}^\ell$, $\mathbf{x} \in \mathbb{F}^\ell$, $\mathbf{y} \in \mathbb{F}^\ell$, and private inputs $\mathbf{f} \in \mathbb{F}^{(<d)}[X]^\ell$ and $\mathsf{open} = (\mathsf{open}_1, ..., \mathsf{open}_\ell)$ such that $\mathsf{Verify}(pp, f_i, \mathsf{open}_i, \mathsf{C}_i) = 1$ for all $i \in [\ell]$:*

$$\mathsf{Aggregate}\big(\mathcal{P}(\mathbf{f}, \mathbf{open}), \mathcal{V}(\mathbf{C}, \mathbf{x}, \mathbf{y})\big) \to ((\mathsf{open}^*, f^*), (C^*, x^*, y^*, b))$$

The public output is a tuple in $\mathbb{G} \times \mathbb{F}^2 \times \{0, 1\}$ and $|C^*| = \mathsf{poly}(\lambda)$ independent of ℓ. The security requirement is that the batch evaluation protocol shown in Figure 1 is a proof of knowledge for the relation:

$$\mathcal{R}_{\mathsf{BatchEval}}(pp, d) = \big\{ \big\langle (\mathbf{C}, \mathbf{x}, \mathbf{y}), (\mathbf{f}, \mathsf{open}) \big\rangle : ((\mathsf{C}_i, x_i, y_i), (f_i, \mathsf{open}_i)) \in \mathcal{R}_{\mathsf{Eval}}(pp, d) \big\}$$

As for correctness, if the inputs to $\mathcal{P}$ satisfy $\mathcal{R}_{\mathsf{BatchEval}}(pp, d)$ then $\mathcal{V}$ outputs $b = 1$ and the private output (open^*, f^*) satisfies $\mathsf{Verify}(pp, f^*, \mathsf{open}^*, C^*) = 1$.

Fig. 1: A batch evaluation protocol for multiple commitments at multiple points based on a PCS aggregation scheme.

$\underline{\mathcal{P}(\mathbf{C}, \mathbf{z}, \mathbf{y}, \mathbf{open}, \mathbf{f})}$	$\underline{\mathcal{V}(\mathbf{C}, \mathbf{z}, \mathbf{y})}$
$((\mathsf{open}^*, f^*), (C^*, z^*, y^*, b_1)) \leftarrow \mathsf{Aggregate}\big(\mathcal{P}(\mathbf{f}, \mathbf{open}), \mathcal{V}(\mathbf{C}, \mathbf{z}, \mathbf{y})\big)$	
	Reject if $b_1 = 0$
$(\perp, b_2) \leftarrow \mathsf{Eval}\big(\mathcal{P}(f^*, \mathsf{open}^*), \mathcal{V}(pp, C^*, z^*, y^*)\big)$	
	Accept if $b_2 = 1$

Theorem 4.2. *Any PCS that has a linear combination scheme* LinCombine *(Definition 3.3) also has an aggregation scheme* Aggregate *(Definition 4.1) that on ℓ input commitments makes a single call to* LinCombine *on $\ell + 2$ commitments with λ-bit integer coefficients. Both the prover and verifier do an additional $O(\ell \log \ell)$ operations in $\mathbb{F}$, and the prover makes one call to* Commit *on a polynomial of degree $max_i\{deg(f_i)\}$. The additional communication is one $\mathbb{G}$ element and two $\mathbb{F}$ elements.*

Corollary 4.3. *Every additive PCS (Definition 3.1) has an aggregation scheme with prover complexity $O(\ell \log \ell)$ operations in $\mathbb{F}$ plus one* Commit *to a polynomial of degree $max_i\{deg(f_i)\}$, verifier complexity $O(\ell \log \ell)$ operations in $\mathbb{F}$ plus $O(\ell \cdot \lambda)$ operations in $\mathbb{G}$, and communication of one $\mathbb{G}$ element plus two $\mathbb{F}$ elements.*

We will say that an aggregation scheme is **efficient** if the verifier complexity of the protocol Aggregate is sublinear in the maximum degree of the input polynomials. By Corollary 4.3, every additive PCS, and more generally any PCS with an efficient linear combination scheme, has an efficient aggregation scheme.

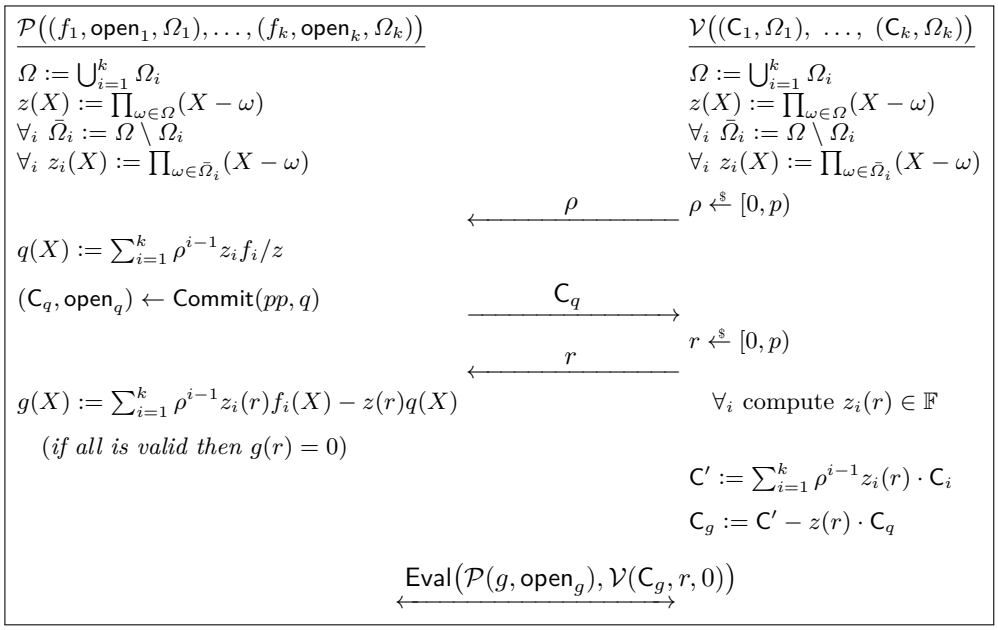
Corollary 4.4. *If a PCS has an efficient linear combination scheme then it has an efficient aggregation scheme.*

4.1 A Protocol for Batch Zero Testing

We first construct a general protocol for batch zero testing. Batch evaluation is a simple generalization. The entire protocol is shown in Figure 2. The communication is comprised of one extra commitment and one evaluation protocol, independent of the number of input polynomials k. In Theorem 4.5 we show that the protocol is knowledge-sound.

The protocol preserves zero-knowledge. The zero-knowledge simulator for this protocol samples $\tilde{\rho}, \tilde{r} \leftarrow \mathbb{F}$, computes an integer representative $\hat{z} \in [0, p)$ for $z(\tilde{r})^{-1}$, sets $\tilde{\mathsf{C}}_q := \sum_{i=1}^{k} \tilde{\rho}^{i-1} z_i(\tilde{r}) \cdot \hat{z} \cdot \mathsf{C}_i$, and sets $\tilde{\mathsf{C}}_g := \sum_{i=1}^{k} \tilde{\rho}^{i-1} z_i(\tilde{r}) \cdot \mathsf{C}_i - z(\tilde{r}) \cdot \tilde{\mathsf{C}}_q$. If there exists an opening for each C_i then there exists an opening of $\mathsf{C}_i - z(\tilde{r}) \cdot (\hat{z} \cdot \mathsf{C}_i)$ to the zero-polynomial, and thus there exists an opening of $\tilde{\mathsf{C}}_g$ to the zero-polynomial. The simulator calls the Eval simulator on public input

Fig. 2: A zero test for multiple polynomials on distinct sets:
$(\mathsf{C}_i, \mathsf{open}_i) \leftarrow \mathsf{Commit}(pp, f_i)$ and Ω_i is a non-empty subset of $\mathbb{F}$ for all $i \in [k]$. The prover computes open_g from $\rho, r, \mathsf{open}_1, \ldots, \mathsf{open}_k$ (not shown).



$(\tilde{\mathsf{C}}_g, \tilde{r}, 0)$ to get a simulated transcript $\tilde{\pi}$. It output the final simulated transcript $(\tilde{\rho}, \tilde{\mathsf{C}}_q, \tilde{r}, \tilde{\pi})$.

Theorem 4.5. *If Eval is knowledge sound, then the protocol in Figure 2 is a proof of knowledge for the relation:*

$$\mathcal{R}_{\mathit{ZTest}}(pp, d) := \left\{ \left\langle (\mathsf{C}, \boldsymbol{\Omega}), (\mathbf{f}, \mathbf{open}) \right\rangle : \begin{array}{l} \mathbf{f} = (f_1, ..., f_k)\ s.t. f_i \in \mathbb{F}^{(<d)}[X] \\ \forall i \in [k] \forall_{\omega \in \Omega_i} f_i(\omega) = 0 \\ \forall i \in [k]\, \mathit{Verify}(pp, \mathsf{C}_i, \mathsf{open}_i, f_i) = 1 \end{array} \right\}$$

The proof is included in the full version of this paper.

4.2 Batch evaluation protocol

The protocol for batch evaluation is a small generalization of the zero-testing protocol in Figure 2. Here, for $i \in [k]$, the verifier has $(\mathsf{C}_i, \Omega_i, t_i)$ where $t_i \in \mathbb{F}^{(<d)}[X]$, and needs to be convinced that $f_i(x) = t_i(x)$ for all $i \in [k]$ and all $x \in \Omega_i$. This is the same as proving that every polynomial $\hat{f}_i := f_i - t_i$ is zero on all of Ω_i. Thus, we can apply the protocol in Figure 2 to $\hat{f}_1, \ldots, \hat{f}_k$.

Naively, the verifier would need to compute a commitment to each $\hat{f}_i$, which it can do from C_i and t_i. However, we can optimize the verifier by observing that the

verifier only uses $t_i(X)$ to compute $t_i(r)$ for some random $r \in \mathbb{F}$. Hence, we can replace the verifier's computation of C' in Figure 2 by instead computing $\mathsf{C}' := \sum_{i=1}^{k} \rho^{i-1} z_i(r) \cdot (\mathsf{C}_i - t_i(r) \cdot \mathsf{C}^{(1)})$ where $\mathsf{C}^{(1)}$ is a commitment to the polynomial $f \equiv 1$. In doing so, we save the verifier the work to compute commitments to $\hat{f}_1, \ldots, \hat{f}_k$.

Theorem 4.6. *If* Eval *is knowledge sound, then the batch evaluation protocol based on Figure 2 is a proof of knowledge for the relation* $\mathcal{R}_{\mathsf{BatchEval}}(pp, d)$.

The complete proof of Theorem 4.6 is included in the full version of this paper. The proof applies the forking lemma (Lemma 2.2) to show that it is possible to generate a depth-2 tree of $2k$ protocol transcripts where:

1. There are k distinct first-round challenges $\rho_1 \neq \cdots \neq \rho_k \neq 0 \bmod p$
2. For all $i \in [k]$, two transcripts share the first-round challenge ρ_i and have distinct second-round challenges r_i and r_i' such that $z(r_i) \neq z(r_i') \neq 0$.
3. Letting $\mathbf{V} \in \mathbb{Z}^{k \times k}$ denote the Vandermonde matrix with jth row $(1, \rho_j, ..., \rho_j^{k-1})$ and letting $\mathbf{R} \in \mathbb{Z}^{k \times k}$ be the matrix with (i,j)th coordinate $z_i(r_j)$, the Hadamard product of these matrices $\mathbf{A} := \mathbf{V} \circ \mathbf{R}$ is invertible over $\mathbb{F}_p$.

The first two conditions are easy to guarantee because collisions among first round challenges occur with negligible probability, and similarly $z(r) \neq z(r') \neq 0$ with overwhelming probability over $r, r' \xleftarrow{\$} \mathbb{F}$. The third condition is guaranteed by the fact (proven in Lemma 4.7) that if every entry of $\mathbf{R}$ is non-zero over $\mathbb{F}_p$, then for $\{\rho_j\}$ sampled uniformly and independently the matrix $\mathbf{A}$ is invertible with overwhelming probability. For $\{r_j\}$ sampled uniformly and independently, $z_i(r_j) \neq 0 \bmod p$ except with probability $\frac{k}{|\mathbb{F}|}$.

For each of these transcripts, the Eval extractor is invoked to extract an opening of each $\mathsf{C}_j' = \sum_{i=1}^{k} \rho_j^{i-1} z_i(r_j) \cdot \mathsf{C}_i$ to a polynomial f_j'. This gives a system of equations that can be solved to obtain openings of the input commitments $(\mathsf{C}_1, ..., \mathsf{C}_k)$ to polynomials $(f_1, ..., f_k) = \mathbf{A}^{-1} \cdot (f_1', ..., f_k')$.

The protocol is still zero-knowledge if the PCS is hiding and Eval is zero-knowledge. The description of the simulator is nearly identical to the simulator for the protocol in Figure 2 so we will not repeat the details.

Lemma 4.7. *Let* $\mathbf{M}$ *be an* $n \times n$ *matrix over* $\mathbb{F}_p^{\times}$. *Let* $\mathbf{V}$ *be a random Vandermonde matrix over* $\mathbb{F}_p$, *sampled uniformly and independent of* $\mathbf{A}$. *Their Hadamard product* $\mathbf{V} \circ \mathbf{M}$ *is invertible with probability at least* $1 - \frac{n^2}{|\mathbb{F}|}$.

Proof. Let $\mathbf{V}(\mathbf{X})$ denote the Vandermonde matrix over formal variables $X_1, ..., X_n$. Using the Leibnitz formula, $det(\mathbf{V}(\mathbf{X}))$ is an n-variate polynomial, which is an alternating sum of $n!$ distinct monomials. The determinant of the Hadamard product, $det(\mathbf{V}(\mathbf{X}) \circ \mathbf{M})$ is also an alternating sum of $n!$ distinct monomials where the coefficient on each distinct monomial is a distinct summand of the Leibnitz formula for $det(\mathbf{M})$. All coefficients are non-zero since all entries of $\mathbf{A}$ are non-zero. Therefore, this n-variate polynomial is not identically zero. Let

$p(X_1, ..., X_n)$ denote this polynomial, which has total degree less than n^2. A random Vandermonde matrix $\mathbf{V}$ is a random assignment $\mathbf{x} = (x_1, ..., x_n)$ to the n variables $X_1, ..., X_n$ and thus $det(\mathbf{V} \circ \mathbf{M}) = p(x_1, ..., x_n)$. By the Schwartz-Zippel lemma, the probability that $p(x_1, ..., x_n) = 0$ is at most $\frac{n^2}{|\mathbb{F}|}$. $\square$

4.3 Aggregation scheme (proof of Theorem 4.2)

When the PCS has a linear combination scheme (Definition 3.3), then the protocol from Section 4.2 together with the linear aggregation protocol $\mathsf{LinCombine}$ results in an aggregation scheme for the PCS. Concretely, the protocol on public inputs $\mathbf{C} = (C_1, ..., C_k) \in \mathbb{G}^k$, $\mathbf{x} = (x_1, ..., x_k) \in \mathbb{F}^k$, and $\mathbf{y} = (y_1, ..., y_k) \in \mathbb{F}^k$ with prover private inputs $\mathbf{f} = (f_1, ..., f_k) \in \mathbb{F}^{(<d)}[X]^k$ and $\mathsf{open} = (\mathsf{open}_1, ..., \mathsf{open}_k)$ operates as follows:

$\mathsf{Aggregate}\big(\mathcal{P}(\mathbf{f}, \mathbf{open}), \mathcal{V}(\mathbf{C}, \mathbf{x}, \mathbf{y})\big) \to ((\mathsf{open}^*, f^*), (C^*, x^*, y^*, b))$

1. Let $\Omega_i = \{x_i\}$ for $i \in [1, k]$, and let $t_i := y_i$.
2. Run the protocol in Section 4.2 with public inputs $\{(\mathsf{C}_i, \Omega_i, t_i)\}_{i \in [k]}$ and prover private inputs $\{(f_i, \mathsf{open}_i)\}_{i \in [k]}$ up until the point that $\mathcal{P}$ and $\mathcal{V}$ derive C_g, the prover $\mathcal{P}$ has privately derived $g(X)$, and the verifier $\mathcal{V}$ has sent the challenge $r \in \mathbb{F}$. Note that C_g is a linear combination of the input commitments $\mathbf{C}$, the C_q sent during the protocol, and $\mathsf{C}^{(1)}$ (the commitment to 1).
3. The prover and verifier will run $\mathsf{LinCombine}$ to produce a succinct commitment C^* to the same polynomial as C_g:
   - Let $\mathbf{C}' := (\mathsf{C}_1, ..., \mathsf{C}_k, \mathsf{C}^{(1)}, \mathsf{C}_q)$
   - Let $\mathbf{f}' := (f_1, ..., f_k, 1, q)$ and let $\mathbf{open}' = (\mathsf{open}_1, ..., \mathsf{open}_k, \mathsf{open}^{(1)}, \mathsf{open}_q)$
   - For $i \in [k]$ let $\alpha_i := \rho^{i-1} \cdot z_i(r) \cdot f_i$, let $\alpha_{k+1} := -\sum_{i=1}^{k} \rho^{i-1} \cdot z_i(r) \cdot y_i$, and let $\alpha_{k+2} := -z(r)$. Let $\boldsymbol{\alpha} := (\alpha_1, ..., \alpha_{k+2})$.
   - Run the protocol $\mathsf{LinCombine}\big(\mathcal{P}(\mathbf{f}', \mathsf{open}'), \mathcal{V}(pp, \mathbf{C}', \boldsymbol{\alpha}) \to (\mathsf{open}^*, (C^*, b)\big)$.
   - The prover's private output is (open^*, g) and the verifier's public output is $(C^*, r, 0, b)$.

In the case that $(C^*, \mathsf{open}^*) = (\mathsf{C}_g, \mathsf{open}_g)$, i.e. the PCS is additive, then composing this protocol with an Eval on C_g is a special case of the batch evaluation protocol in Section 4.2, which by Theorem 4.6 is a proof of knowledge for relation $\mathcal{R}_{\mathsf{BatchEval}}(pp, d)$. More generally, by the security property of the linear combination scheme $\mathsf{LinCombine}$, composing the protocol with an Eval on $(\mathsf{C}^*, r, 0)$ is equivalent to running Eval on $(\mathsf{C}_g, r, 0)$, i.e. it is a proof of knowledge of an opening for C_g at the pair $(r, 0)$. Thus, this provides the extractor from Theorem 4.6 with the same information it needs to extract an $\mathcal{R}_{\mathsf{BatchEval}}(pp, d)$ witness.

The prover complexity in the aggregation protocol is $O(k \log k)$ operations in $\mathbb{F}$ using FFTs plus the complexity of a single call to Commit on a polynomial of degree at most d. The verifier complexity is $O(k \log k)$ operations in $\mathbb{F}$ and $O(k \cdot \lambda)$ operations in $\mathbb{G}$.

5 Homomorphic PCS Public Aggregation

The aggregation scheme in Definition 4.1 requires the aggregator, who plays the role of a prover, to know openings of all the input commitments. In a *public aggregation scheme*, the aggregator isn't required to know the openings of the input commitments but performs more work than the verifier. We define public aggregation only for a PCS with a non-interactive evaluation protocol $\mathsf{NI\text{-}Eval}$.

The verifier in the $\mathsf{Aggregate}$ protocol receives $\mathsf{NI\text{-}Eval}$ proofs π_i for each (C_i, x_i, y_i) input tuple. The prover's output is (open^*, f^*) and the verifier's output is (C^*, x^*, y^*, b). If the prover succeeds in the aggregation protocol (i.e., the verifier outputs $b = 1$) and the verifier separately verifies the membership of (C^*, x^*, y^*) in $\mathcal{R}_{\mathsf{Eval}}(pp, d)$ then it should be convinced that each input tuple is also in $\mathcal{R}_{\mathsf{Eval}}(pp, d)$ with overwhelming probability.

Definition 5.1 (Public Aggregation). *Let $\mathcal{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify}, \mathsf{NI\text{-}Eval})$ denote a PCS with commitment group $\mathbb{G}$ and a non-interactive evaluation protocol. A public aggregation scheme for $\mathcal{PCS}$ is a public-coin interactive protocol $\mathsf{Aggregate}$ that has public inputs $\mathbf{C} = (C_1, ..., C_\ell) \in \mathbb{G}^\ell$, $\mathbf{x} \in \mathbb{F}^\ell$, $\mathbf{y} \in \mathbb{F}^\ell$, and $\boldsymbol{\pi} = (\pi_1, ..., \pi_\ell)$:*

$$\mathsf{Aggregate}\big(\mathcal{P}, \mathcal{V}(pp, \boldsymbol{\pi}, \mathbf{C}, \mathbf{x}, \mathbf{y})\big) \to ((\mathsf{open}^*, f^*), (C^*, x^*, y^*, b))$$

In a correct scheme, if the inputs satisfy $\mathcal{V}_{\mathsf{Eval}}(\pi_i, C_i, x_i, y_i) = 1$ for all $i \in [\ell]$, then the outputs satisfy $b = 1$ and $\mathsf{Verify}(pp, f^*, \mathsf{open}^*, C^*) = 1$. The soundness requirement is that the following probability is negligible:

$$\Pr\left[\begin{array}{l|l} & pp \leftarrow \mathsf{Setup}(\lambda, d) \\ b \wedge \mathcal{V}_{\mathsf{Eval}}(\pi^*, C^*, x^*, y^*) = 1 & (\mathbf{C}, \mathbf{x}, \mathbf{y}, \boldsymbol{\pi}) \leftarrow \mathcal{A}(pp) \\ \exists_i \mathcal{V}_{\mathsf{Eval}}(pp, \pi_i, C_i, x_i, y_i) \neq 1 & ((\mathsf{open}^*, f^*), (C^*, x^*, y^*, b)) \leftarrow \mathsf{Aggregate}\big(\mathcal{P}, \mathcal{V}(pp, \boldsymbol{\pi}, \mathbf{C}, \mathbf{x}, \mathbf{y})\big) \\ & \pi^* \leftarrow \mathsf{NI\text{-}Eval}\big(pp, f^*, \mathsf{open}^*, C^*, x^*, y^*\big) \end{array}\right]$$

A public aggregation scheme is **efficient** if the verifier complexity of the protocol $\mathsf{Aggregate}$ is sublinear in the maximum degree of the input polynomials.

Theorem 5.2. *There is a black-box compilation from any additive PCS over a prime field $\mathbb{F} = \mathbb{F}_p$ and commitment group $\mathbb{G}$ into a publicly aggregatable homomorphic PCS with the same commitment group $\mathbb{G}$. The overhead of the new Eval is:*

- *Communication: $O(\log d)$ additional elements of $\mathbb{G} \times \mathbb{F}$*
- *Prover: $O((\log p + \lambda) \cdot n)$ additional operations in $\mathbb{G}$*
- *Verifier: $O(\log d)$ additional operations in $\mathbb{G} \times \mathbb{F}$*

The public aggregation scheme complexity for ℓ commitments is:

- *Communication: One $\mathbb{G}$ element and two $\mathbb{F}$ elements.*

– *Prover:* $O(\ell \log \ell)$ *operations in* $\mathbb{F}$, $O(\log p \cdot n)$ *operations in* $\mathbb{G}$, *and* $O(\ell \cdot n)$ *multiplications of* λ*-bit integers*
– *Verifier:* $O(\ell \log \ell)$ *operations in* $\mathbb{F}$ *and* $O(\ell \cdot \lambda)$ *operations in* $\mathbb{G}$.

Theorem 5.2 is proven in two parts. First, there is a simple transformation from any additive PCS into a homomorphic PCS with the same commitment group and opening group $\mathbb{H} = \mathbb{Z}^n$. Second, we present a compiler from any homomorphic PCS with opening group $\mathbb{H} = \mathbb{Z}^n$ into a new homomorphic PCS together with a public aggregation scheme that meets the performance requirements of the theorem. A key ingredient is a protocol for *succinct proof of knowledge of homomorphism pre-image*, which we present next.

5.1 A Succinct PoK for Homomorphism Pre-image

Let $\phi : \mathbb{Z}^n \to \mathbb{G}$ be any homomorphism where $\mathbb{G}$ is an abelian computational group. We will present a succinct public-coin interactive proof of knowledge for the following relation:

$$\mathcal{R}^*_{\mathsf{HPI}}(\phi, \mathbb{G}, p) = \{((\mathbf{x} \in \mathbb{Z}^n, t \in \mathbb{Z}), y \in \mathbb{G}) : \phi(\mathbf{x}) = t \cdot y \wedge t \neq 0 \bmod p\}$$

In the special case that $p\mathbb{Z} \subseteq ker(\phi)$, e.g. when $\mathbb{G}$ has order p or is an $\mathbb{F}_p$-vector space, a proof of knowledge for this relation is equivalent to a proof of knowledge for the standard homomorphism pre-image relation. In this case, given a witness $(\mathbf{x}, t)$ for $\mathcal{R}^*_{\mathsf{HPI}}$ it is possible to efficiently compute an integer vector $\mathbf{x}'$ such that $\phi(\mathbf{x}') = y$ by computing $\hat{t} \in \mathbb{Z}$ such that $\hat{t} \equiv t^{-1} \bmod p$ and setting $\mathbf{x}' := \hat{t} \cdot x$.

Let $\{e_i\}_{i \in [n]}$ denote the standard basis of $\mathbb{Z}^n$ and define $g_i := \phi(e_i)$. The homomorphism ϕ may be rewritten as the $\mathbb{Z}$-linear map $\phi(\mathbf{x}) = \langle \mathbf{x}, \mathbf{g} \rangle = \sum_{i=1}^{n} x_i \cdot g_i$. We will use $[\![\mathbf{x}]\!]_{\mathbf{g}}$ as a shorthand notation for $\langle \mathbf{x}, \mathbf{g} \rangle$ give $\mathbf{x} \in \mathbb{Z}^n$ and $\mathbf{g} \in \mathbb{G}^n$.

Note the following two properties of $[\![\cdot]\!]$:

1. **Decomposition** If $\mathbf{x} = (\mathbf{x}_L, \mathbf{x}_R)$ for $\mathbf{x}_L \in \mathbb{Z}^{n_1}$ and $\mathbf{x}_R \in \mathbb{Z}^{n_2}$ such that $n_1 + n_2 = n$ and $\mathbf{g} = (\mathbf{g}_L, \mathbf{g}_R)$ for $\mathbf{g}_L \in \mathbb{G}^{n_1}$ and $\mathbf{g}_R \in \mathbb{G}^{n_2}$, then $[\![\mathbf{x}]\!]_{\mathbf{g}} = [\![\mathbf{x}_L]\!]_{\mathbf{g}_L} + [\![x_R]\!]_{\mathbf{g}_R}$.
2. **Bilinearity** If $\alpha, \beta \in \mathbb{Z}$, $\mathbf{x} \in \mathbb{Z}^n$,and $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$ then $\alpha[\![\mathbf{x}]\!]_{\mathbf{g}} + \beta[\![\mathbf{x}]\!]_{\mathbf{h}} = [\![\alpha\mathbf{x}]\!]_{\mathbf{g}} + [\![\beta\mathbf{x}]\!]_{\mathbf{h}} = [\![\mathbf{x}]\!]_{\alpha\mathbf{g}+\beta\mathbf{h}}$

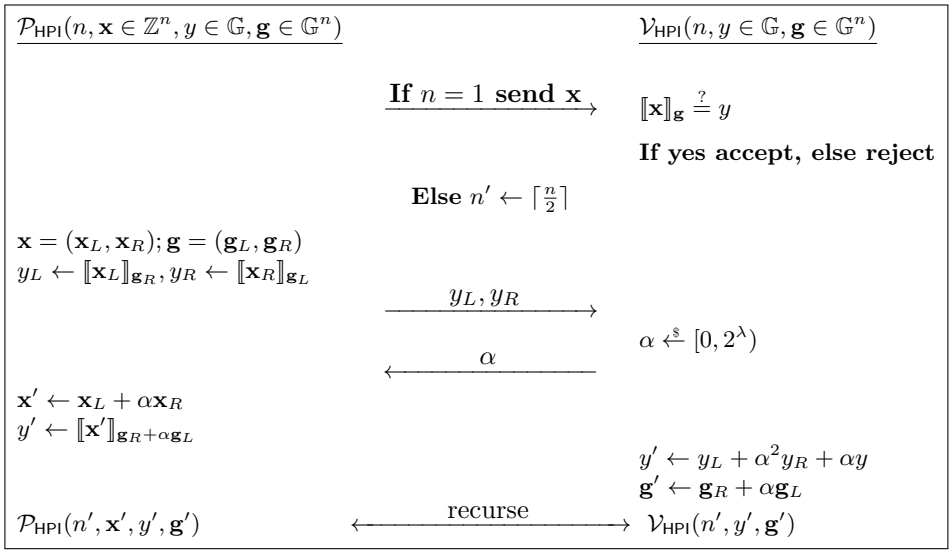
The public coin interactive proof is illustrated in Figure 3. The verifier's public-coin challenges are sampled uniformly from the set $\mathcal{X} := [0, 2^\lambda)$.

Correctness If the prover follows the protocol honestly, then $[\![\mathbf{x}]\!]_{\mathbf{g}} = [\![\mathbf{x}_L]\!]_{\mathbf{g}_L} + [\![\mathbf{x}_R]\!]_{\mathbf{g}_R}$, and:

$$\begin{aligned} y' = y_L + \alpha^2 y_R + \alpha y &= [\![\mathbf{x}_L]\!]_{\mathbf{g}_R} + [\![\alpha^2 \mathbf{x}_R]\!]_{\mathbf{g}_L} + [\![\alpha \mathbf{x}_L]\!]_{\mathbf{g}_L} + [\![\alpha \mathbf{x}_R]\!]_{\mathbf{g}_R} \\ &= [\![\mathbf{x}']\!]_{\mathbf{g}_R} + [\![\alpha \mathbf{x}']\!]_{\mathbf{g}_L} = [\![\mathbf{x}']\!]_{\mathbf{g}_R + \alpha \mathbf{g}_L} \end{aligned}$$

Thus, in each recursive round, if $\mathbf{x}$ is a valid witness for $(y, n, \mathbf{g})$ then $\mathbf{x}'$ is a valid witness for $(y', n', \mathbf{g}')$.

Fig. 3: *A succinct interactive protocol for HPI.* For simplicity n is a power of 2.



Theorem 5.3. *The protocol in Figure 3 is a proof of knowledge for the relation* $\mathcal{R}^*_{\mathsf{HPI}}(\phi, \mathbb{G}, p)$.

Proof. Our analysis will show the protocol is a proof of knowledge for the relation $\mathcal{R}^*_{\mathsf{HPI}}([\![\cdot]\!], \mathbb{G}, p)$. For simplicity we assume n is a power of 2. We define a knowledge extractor $\mathcal{E}$ that runs with an adversary $\mathcal{A}$ who succeeds for public input $(\mathbf{x}, y, \mathbf{g})$ with probability $\epsilon = 1/\mathsf{poly}(\lambda)$. $\mathcal{E}$ begins by invoking the forking lemma to generate a tree of accepting transcripts with the following characteristics:

- The tree has depth $\log n$ and branching factor 3. We will index nodes by $v \in [0, n^{\log 3})$.
- The root is labeled with the verifier's input $(y, \mathbf{g})$.
- Each non-leaf node v distinct from the root is labeled with a challenge α_v and a prover message $(y_{v,0}, y_{v,1})$.
- Each non-leaf node v has three children each labeled with three distinct verifier challenges. $\alpha_{v,1} \neq \alpha_{v,2} \neq \alpha_{v,3}$.
- Each leaf node v is labeled with a prover message $x_v \in \mathbb{Z}$.

Since the probability of collision on a pair of challenges sampled uniformly from $\mathcal{X}$ is $1/2^\lambda$, by the forking lemma (Lemma 2.2) this tree-finding algorithm runs for time polynomial in λ and succeeds excepts with negligible probability in λ.

For any non-leaf node v with parent w and message pair $(y_{v,0}, y_{v,1})$ and challenge α_v define $y_v := y_{w,0} + \alpha_v^2 \cdot y_{w,1} + \alpha_v \cdot y_w$. For any leaf node v the value of y_v is already defined by the transcript. For the root node rt define $y_{\mathsf{rt}} := y$,

where y is the input. We also define a value $\mathbf{g}_v$ for every node v as follows: if v is the root then $\mathbf{g}_v := \mathbf{g}$, else if v has a parent w then $\mathbf{g}_v := \mathbf{g}_{w,0} + \alpha_v \cdot \mathbf{g}_{w,1}$ where $\mathbf{g}_w = (\mathbf{g}_{w,0}, \mathbf{g}_{w,1})$ is the concatenation of equal length vectors $\mathbf{g}_{w,0}, \mathbf{g}_{w,1}$. If v is a node on the ith level up from the leaves then $\mathbf{g}_v \in \mathbb{G}^{2^i}$. Every component of $\mathbf{g}_v$ is a linear combination of the elements in $\mathbf{g}$ derived from challenges along a path up the tree. Thus, for each $\mathbf{g}_v$ the extractor also knows a matrix $\mathbf{U}_v \in \mathbb{Z}^{2^i \times n}$ such $\mathbf{U}_v \cdot \mathbf{g} = \mathbf{g}_v$. By construction, for every root to leaf path of nodes $v_1, ..., v_{\log n}$ the sequence of values $(\alpha_{v_i}, y_{v_i,0}, y_{v_i,1})$ form an accepting transcript between the prover and verifier where $(\mathbf{g}_{v_i}, y_{v_i})$ are the verifier's local inputs in the ith round. Moreover, the leaf node labels satisfy $x_v \cdot \mathbf{g}_v = y_v$.

We will show that given this tree, the extractor can compute $(t_v, \mathbf{x}_v) \in \mathbb{Z} \times \mathbb{Z}^n$ for each node v such that $[\![\mathbf{x}_v]\!]_{\mathbf{g}} = t_v \cdot y_v$. In particular, this means that the extractor obtains a witness $(t_{\mathsf{rt}}, \mathbf{x}_{\mathsf{rt}}) \in \mathbb{Z} \times \mathbb{Z}^n$ for $y \in \mathbb{G}$ such that $[\![\mathbf{x}_{\mathsf{rt}}]\!]_{\mathbf{g}} = t_{\mathsf{rt}} \cdot y$. This is a valid pair for the relation $\mathcal{R}^*_{\mathsf{HPI}}([\![\cdot]\!], \mathbb{Z}^n, \mathbb{G})$. The extractor begins at the leaves. Every leaf node is already labeled with $x_v \in \mathbb{Z}$ such that $x_v \cdot \mathbf{g}_v = x_v \cdot \mathbf{U}_v \cdot \mathbf{g} = y_v$ where $\mathbf{U}_v \in \mathbb{Z}^{1 \times n}$. The extractor sets $\mathbf{x}_v := x_v \cdot \mathbf{U}_v$. Next, suppose the extractor has already successfully computed an $(t_v, \mathbf{x}_v)$ pair for all children nodes of a node w. For ease of notation, temporarily let y_1, y_2, y_3 denote the y_v values for the three children and $\alpha_1, \alpha_3, \alpha_3$ denote their respective challenge labels. Similarly, let $(\mathbf{x}_i, t_i) \in \mathbb{Z}^n \times \mathbb{Z}$ for $i \in [3]$ denote the extracted labels for the children nodes. By construction, $y_i = y_w + \alpha_i^2 y_{w,0} + \alpha_i y_{w,1}$ for $i \in [3]$. Defining $\mathbf{A} \in \mathbb{Z}^{3 \times 3}$ to be the matrix with rows $(1, \alpha_i^2, \alpha_i)$, $\mathbf{T}$ the diagonal matrix with diagonal entries $t_1, t_2, t_3 \neq 0 \bmod p$, and $\mathbf{X} \in \mathbb{Z}^{3 \times n}$ the integer matrix with rows $\mathbf{x}_1$, $\mathbf{x}_2, \mathbf{x}_3$, we can summarize the relations:

$$\mathbf{A} \cdot \begin{bmatrix} y_w \\ y_{w,0} \\ y_{w,1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \qquad \mathbf{T} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} [\![\mathbf{x}_1]\!]_{\mathbf{g}} \\ [\![\mathbf{x}_2]\!]_{\mathbf{g}} \\ [\![\mathbf{x}_3]\!]_{\mathbf{g}} \end{bmatrix} = \mathbf{X} \cdot \mathbf{g}$$

$\mathbf{T}$ is invertible over $\mathbb{F}$. Since $\mathbf{A}$ is Vandermonde it is also invertible over $\mathbb{F}$. Therefore $\mathbf{T} \cdot \mathbf{A}$ is invertible over both $\mathbb{F}$ and $\mathbb{Q}$. Setting d to be the least common multiple of the denominators of all entries in $(\mathbf{T} \cdot \mathbf{A})^{-1}$ over $\mathbb{Q}$, there exists an integer matrix $\mathbf{P}$ such that $\mathbf{P} \cdot \mathbf{T} \cdot \mathbf{A} = d \cdot \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. In particular, we obtain $d \cdot y_w = \langle \mathbf{P}_1, \mathbf{X} \cdot \mathbf{g} \rangle$. The extractor sets $\mathbf{x}_w := \langle \mathbf{P}_1, \mathbf{X} \rangle$ and $t_w := d$, which now satisfies $[\![\mathbf{x}_w]\!]_{\mathbf{g}} = \langle \mathbf{x}_w, \mathbf{g} \rangle = t_w \cdot y_w$. □

5.2 Publicly aggregatable PCS (proof of Theorem 5.2)

The Halo [18] protocol contains a public aggregation protocol for the Bulletproofs PCS. Inspired by this idea, we show how the HPI protocol of Figure 3 can be used to compile any homomorphic PCS with opening group $\mathbb{H} = \mathbb{Z}^n$ and commitment group $\mathbb{G}$ into a publicly aggregatable homomorphic PCS with the same commitment group $\mathbb{G}$. Compared with the commitment size and Eval complexity of the original PCS, the commitment size of the transformed PCS is the same, the new Eval communication has an extra $O(\log d)$ elements of $\mathbb{G}$, and the verification overhead is $O(\log d)$ operations in $\mathbb{G}$. Running the public

aggregation protocol on k commitments and evaluation points together with an Eval on the aggregate commitment achieves an amortized verification complexity of $O(\log k + \lambda + \frac{V_{\mathsf{Eval}}(\lambda,d)}{k})$ where $V_{\mathsf{Eval}}(\lambda, d)$ is the Eval verifier complexity. Any additive/homomorphic scheme can first be compiled into a homomorphic PCS with opening group $\mathbb{Z}^n$, using the simple compiler described next.

Compiler 1: From Additive to Homomorphic Given a non-hiding[9] additive PCS (Setup, Commit, Verify, Eval) the new homomorphic non-hiding PCS uses the same $\mathsf{Setup}, \mathsf{Verify}$, and Eval protocols, but commits to polynomials using the pre-computed "basis" commitments $(\mathsf{C}_i, \mathsf{open}_i) \leftarrow \mathsf{Commit}(pp, X^{i-1})$ for $i \in [1, d]$. The commitment to $f \in \mathbb{F}^{(<d)}[X]$ with coefficient vector representation $\mathbf{f} = (\hat{f}_0, ..., \hat{f}_{d-1}) \in [0, p)^d$ is the group element $\mathsf{C} := \sum_{i=0}^{d-1} \hat{f}_i \cdot \mathsf{C}_i$. The opening string open for C is the coefficient vector $\mathbf{f}$.

By definition, C is a valid commitment to the polynomial f under the original scheme with opening string open' derived from the "basis" openings open_i using add^* and the coefficients $\mathbf{f}$. The evaluation protocol runs the original Eval using open'. For some schemes (e.g., KZG and Bulletproofs) that are already homomorphic, the linear combination C would be identical to a fresh commitment to f and thus $\mathsf{open}' = \mathsf{open}$. In other words, the transformation described above would have no effect.

The transformed scheme is a homomorphic PCS because $\mathsf{C} = \phi(\mathsf{open})$ where $\phi : \mathbb{Z}^d \to \mathbb{G}$ is the homomorphism that maps $\mathbf{v} \in \mathbb{Z}^d$ to $\sum_{i=1}^{d} v_i \cdot \mathsf{C}_i$ and $\chi(\mathsf{open}) = \mathsf{open} \bmod p$ is the unique coefficient vector of $f \in \mathbb{F}^{(<d)}[X]$. The new scheme is also binding: given a collision $\mathbf{f}' \neq \mathbf{f} \bmod p$ such that $\mathsf{C} = \phi(\mathbf{f}) = \phi(\mathbf{f}')$, the algorithm add^* could be used to derive openings of C to either f or f' from the open_i values, which contradicts the binding property of Commit.

Compiler 2: Homomorphic to publicly aggregatable Denote the input homomorphic PCS by $\mathcal{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify}, \mathsf{Eval})$. The output of the compiler will be a scheme denoted $\mathcal{PCS}^* = (\mathsf{Setup}^*, \mathsf{Commit}^*, \mathsf{Verify}^*, \mathsf{Eval}^*)$ that will support public aggregation. Let $\mathbb{H} = \mathbb{Z}^n$ for some $n > d$. By definition, there are efficiently computable homomorphisms $\phi : \mathbb{Z}^n \to \mathbb{G}$ and $\chi : \mathbb{Z}^n \to \mathbb{F}^{(<d)}[X]$ such that the output $(\mathsf{C}, \mathsf{open}) \leftarrow \mathsf{Commit}(pp, f)$ for any $f \in \mathbb{F}^{(<d)}[X]$ satisfies $\mathsf{C} = \phi(\mathsf{open})$ and $f = \chi(\mathsf{open})$.

For any $\mathbf{v} \in \mathbb{Z}^n$ let $f_{\mathbf{v}} := \chi(\mathbf{v})$. Let $\hat{\mathbb{G}} := \mathbb{G} \times \mathbb{F}$. For a point $x \in \mathbb{F}$, define the homomorphism $\phi_x : \mathbb{Z}^n \to \hat{\mathbb{G}}$ as $\phi_x(\mathbf{v}) := (\phi(\mathbf{v}), f_{\mathbf{v}}(x))$. The new PCS algorithms $(\mathsf{Setup}^*, \mathsf{Commit}^*)$ are identical to $(\mathsf{Setup}, \mathsf{Commit})$. The algorithm Verify^* is the standard "relaxation" of Verify from Section 2.1: it accepts tuples $(f, (t, \mathsf{open}))$ such that $\phi(\mathsf{open}) = t \cdot \mathsf{C}$ and $\chi(\mathsf{open}) = t \cdot f$ where $t \neq 0$ is an integer. The protocol Eval^* is transformed as follows:

$\mathsf{Eval}^*\big(\mathcal{P}(f, \mathsf{open}), \mathcal{V}(\mathsf{C}, x, y)\big)$:

[9] Since the PCS is non-hiding we may assume, without loss of generality, that the commitment algorithm Commit is a deterministic function.

1. The prover/verifier run a modification of the HPI protocol from Figure 3 with $\mathcal{P}_{\mathsf{HPI}}(n, \mathsf{open}, (\mathsf{C}, y))$ and $\mathcal{V}_{\mathsf{HPI}}(n, (\mathsf{C}, y))$ for the homomorphism $\phi_x : \mathbb{Z}^n \to \hat{\mathbb{G}}$. The verifier stores the output $(x', (\mathsf{C}', y')) \in \mathbb{Z} \times \hat{\mathbb{G}}$ and performs all verification steps *except* for deriving $g' \in \hat{\mathbb{G}}$ or checking $x' \cdot g' = (\mathsf{C}', y')$. The prover derives the coefficient vector $\mathbf{u}$ of the polynomial $u(X) = \prod_{i=1}^{\log n}(\alpha_i + X^{2^{i-1}})$ defined by the verifier challenges, which satisfies $\phi_x(\mathbf{u}) = g'$ and $\phi_x(x' \cdot \mathbf{u}) = x' \cdot g' = (\mathsf{C}', y')$.
2. Run $\mathsf{Eval}\big(\mathcal{P}(f_{x' \cdot \mathbf{u}}, x' \cdot \mathbf{u}), \mathcal{V}(\mathsf{C}', x, y')\big)$, where C' is interpreted as a polynomial commitment to $f_{x' \cdot \mathbf{u}}$ with opening $x' \cdot \mathbf{u}$.

We provide only a sketch of the knowledge soundness analysis. Recall that the extractor in the analysis of Theorem 5.3 succeeds assuming it has *any* labels $(t_v, \mathbf{x}_v, y_v)$ at the leaves of the tree such that $[\![\mathbf{x}_v]\!]_{\mathbf{g}} = t_v \cdot y_v$, i.e. $\phi_s(\mathbf{x}_v) = t_v \cdot y_v$ in this case. The knowledge extractor for Eval^* begins by running the usual extractor for $\mathcal{P}_{\mathsf{HPI}}$, but calls the extractor for Eval to obtain a ϕ_x homomorphism pre-image of (C', y'). This is passed to the extractor for $\mathcal{P}_{\mathsf{HPI}}$, which in turn outputs a witness $(t, \mathbf{v}) \in \mathbb{Z} \times \mathbb{Z}^n$ such that $((\mathbf{v}, t), (\mathsf{C}, y)) \in \mathcal{R}^*_{\mathsf{HPI}}(\phi_x, \mathbb{Z}^n, \hat{\mathbb{G}})$, i.e. $\phi_x(\mathbf{v}) = (t \cdot \mathsf{C}, t \cdot y)$ and $t \neq 0$. Thus, $\phi(\mathbf{v}) = t \cdot \mathsf{C}$ and $f_{\mathbf{v}}(x) = t \cdot y$, so Verify^* accepts $(t^{-1} f_{\mathbf{v}}, (t, \mathbf{v}))$ and $t^{-1} f_{\mathbf{v}}(x) = y$, i.e. $(t^{-1} f_{\mathbf{v}}, (t, \mathbf{v}))$ is an $\mathcal{R}_{\mathsf{Eval}}$ witness for (C, x, y).

The compiled PCS has the same commitment size since the commitment algorithm is unchanged. The overhead in the Eval^* communication is $O(\log d)$ elements of $\hat{\mathbb{G}} = \mathbb{G} \times \mathbb{F}$ and the overhead in verification is $O(\log d)$ operations in $\hat{\mathbb{G}}$ (from Step 1). The prover overhead is $O((\lambda + \log B) \cdot n)$ operations in $\hat{\mathbb{G}}$ assuming $||\mathsf{open}||_\infty < B$ (in Step 1) and $O(n)$ integer multiplications to derive $\mathbf{u}$ (also from Step 1). In the case that $|\mathbb{G}| = p$ the integer multiplications become field multiplication modulo p.

If the input PCS Eval protocol is zero-knowledge and the prover/verifier run the zero-knowledge variation of the HPI protocol between $\mathcal{P}_{\mathsf{init}}$ and $\mathcal{V}_{\mathsf{init}}$ then Eval^* is also zero-knowledge. If Eval is already non-interactive (or public-coin and FS compatible) then Eval^* is still public-coin and can be made non-interactive by applying the Fiat-Shamir transform. We conjecture that the transformed protocol is sound, which is true in the random oracle model for constant n [34].:

Conjecture 5.4. If Eval is FS compatible then protocol Eval^* is FS compatible.

Comparison to Halo aggregation The Halo aggregation protocol for the Bulletproofs PCS uses the fact that the expensive part of verification is deriving $g' = \phi(\mathbf{u})$ and $u(X)$ can be evaluated in time $O(\log d)$. The aggregator proves correctness of g' (interpreted as a commitment to u) by running the Bulletproofs Eval to open it to $u(s)$ at a random point s chosen by the verifier. Multiple instances can be batched using private Eval aggregation. This works only because $\mathbf{u} \in \mathbb{Z}_p$ and $\phi : \mathbb{Z}_p^n \to \mathbb{G}$ is collision-resistant. In a more general homomorphic PCS with $\mathbf{u} \in \mathbb{Z}^n$, ϕ might only be collision-resistant over $\mathbb{Z}^n / ker(\chi)$ and it may be possible to open g' to $u(X)$ even when $\phi(\mathbf{u}) \neq g'$. The key observation that allows us to generalize the aggregation protocol for any PCS is our novel

analysis of the HPI protocol (Theorem 5.3) which shows that the verifier does not need to compute g'; it only needs a proof of knowledge that y' is *some* linear combination of $\mathbf{g}$.

Public aggregation scheme Each non-interactive proof returned by $\mathsf{NI\text{-}Eval}^*$ has the form $(\pi_{\mathsf{HPI}}, x', y', \pi_{\mathsf{eval}})$ where π_{HPI} is the transcript from the first step, $(x', y') = (x', (\mathsf{C}', t')) \in \mathbb{Z} \times (\mathbb{G} \times \mathbb{F})$ is the verifier's intermediate output in the first step, and π_{Eval} is the non-interactive Eval proof from the second step for the commitment C' to the polynomial $f_{x' \cdot \mathbf{u}}$. The vector $x' \cdot \mathbf{u}$ can be computed from the transcript π_{HPI}.

The public aggregation scheme $\mathsf{Aggregate}$ takes public inputs $\mathbf{C} = (\mathsf{C}_1, ..., \mathsf{C}_k) \in \mathbb{G}^k$, $\mathbf{s} \in \mathbb{F}^k$, $\mathbf{t} \in \mathbb{F}^k$, and a vector of $\mathsf{NI\text{-}Eval}^*$ proofs $\boldsymbol{\pi} = (\pi_1, ..., \pi_k)$ where $\pi_i = (\pi^{(i)}_{\mathsf{HPI}}, x'_i, y'_i, \pi^{(i)}_{\mathsf{eval}})$:

$$\mathsf{Aggregate}\big(\mathcal{P}, \mathcal{V}(pp, \boldsymbol{\pi}, \mathbf{C}, \mathbf{s}, \mathbf{t})\big) \rightarrow ((\mathsf{open}^*, f^*), (C^*, s^*, t^*, b))$$

The verifier does *not* check $\pi^{(i)}_{\mathsf{Eval}}$ for each $i \in [k]$, and therefore is not yet convinced that $\phi_{s_i}(x'_i \cdot \mathbf{u}_i) = y'_i$. Instead, the aggregation prover/verifier run the private aggregation protocol from Section 4.3 where the prover has private inputs $\{f_{x' \cdot \mathbf{u}_i}\}_{i=1}^k$ and opening strings $\{x' \cdot \mathbf{u}_i\}_{i=1}^k$ for each commitment C'_i such that $f_{x' \cdot \mathbf{u}_i}(s_i) = t'_i$. The output of this private aggregation protocol determine the prover's outputs (open^*, f^*) and the verifier's outputs (C^*, s^*, t^*, b).

By the soundness definition of the private aggregation scheme, if the prover can succeed in the Eval protocol on public inputs (C^*, s^*, t^*) with non-negligible probability then there exists a polynomial time knowledge extractor that obtains an $\mathcal{R}_{\mathsf{Eval}}$ witness for each $(\mathsf{C}'_i, s_i, t'_i)$, which includes a ϕ_{s_i} pre-image of $y'_i = (\mathsf{C}'_i, t'_i)$. These witnesses are then used to extract $\mathcal{R}_{\mathsf{Eval}}$ witnesses for each (C_i, s_i, t_i) as described above in the knowledge-soundness analysis for Eval^*.

The public aggregation scheme verification and communication inherits the same complexity as the private aggregation protocol. From Theorem 4.2, the generic scheme from Section 4.3 has verifier complexity $O(k \log k)$ operations in $\mathbb{F}$ plus $O(k \cdot \lambda)$ operations in $\mathbb{G}$ and communication of one $\mathbb{G}$ element plus two $\mathbb{F}$ elements. The prover complexity of the private aggregation subprotocol is $O(k \log k)$ operations in $\mathbb{F}$ plus one Commit to a polynomial of degree at most d. In addition, the prover must derive each integer vector $\mathbf{u}_i$, which requires $O(k \cdot n)$ integer multiplications. In the case that $|\mathbb{G}| = p$ the integer multiplications become field multiplication modulo p.

6 SNARKs and IVC from PCS Aggregation

Bünz et. al. [23] formally show how a concept they define called PCS accumulation schemes can be used to construct a PCD system, generalizing the Halo protocol [18]. We show that a PCS public aggregation scheme satisfies the definition of a PCS accumulation scheme [23]. Our full version contains a detailed and self-contained exposition of IVC/PCD for path distributed computation directly from PCS aggregation.

A PCS accumulation scheme enables PCD from plain-model "predicate-efficient" SNARKs, defined as a SNARK with a polylogarithmic verifier that is given an oracle for checking PCS Eval proofs. The PCD transformation does not work if the SNARK involves calls to a random oracle, as it would require concretely instantiating the random oracle. Unfortunately, we only know how to construct "predicate-efficient" SNARKs in the random oracle model (e.g., [27,32]). Hence, this result gives a *heuristic* construction of PCD from PCS accumulation.

PCS accumulation scheme We show that a public aggregation scheme for a PCS (Definition 5.1) satisfies the definition of an accumulation scheme for a non-interactive PCS from [23]. We first review the definition of an accumulation scheme. The definition has small syntactic differences from [23] due to syntactic differences in our PCS definition.

Definition 6.1 (PCS accumulation). *Let $\mathcal{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify}, \mathsf{Eval})$ denote a PCS with a non-interactive Eval protocol given by a prover algorithm $\mathcal{P}_{\mathsf{Eval}}$ and verifier algorithm $\mathcal{V}_{\mathsf{Eval}}$. An accumulation scheme for $\mathcal{PCS}$ has algorithms (G, I, P, V, D) with the syntax:*

$$\begin{array}{l} G(\lambda) \to pp_{\mathsf{ac}} \\ I(pp_{\mathsf{ac}}, pp_{\mathsf{pc}}) \to (apk, avk, dk) \\ D(dk, acc) \to b_D \end{array} \qquad \begin{array}{l} P(apk, [\{X_i\}_{i=1}^k, \{acc_i\}_{i=1}^\ell) \to (acc, \pi_V) \\ V(avk, \{X_i\}_{i=1}^k, \{acc_i\}_{i=1}^\ell, acc, \pi_V) \to b_V \end{array}$$

The scheme is complete if for any pp_{pc} and $(apk, avk, dk) \leftarrow I(pp_{\mathsf{ac}}, pp_{\mathsf{pc}})$ and inputs $(\{X_i\}_{i=1}^k, [acc_i]_{i=1}^\ell)$ that satisfy $\mathcal{V}_{\mathsf{Eval}}(pp_{\mathsf{pc}}, X_i) = 1$ for $i \in [k]$ and $D(dk, acc_i) = 1$ for all $i \in [\ell]$, the accumulation scheme prover $P(apk, \{X_i\}_{i=1}^k, \{acc_i\}_{i=1}^\ell)$ outputs (acc, π_V) such that $D(dk, acc) = 1$ and $V(avk, \{X_i\}_{i=1}^k, \{acc_i\}_{i=1}^\ell, acc, \pi_V) = 1$. For soundness, the following probability is negligible in λ:

$$\Pr\left[\begin{array}{ll} V(avk, \{X_i\}_{i=1}^k, \{acc_i\}_{i=1}^\ell, acc, \pi_V) = 1 & pp_{\mathsf{pc}} \leftarrow \mathsf{Setup}(\lambda, d), pp_{\mathsf{ac}} \leftarrow G(\lambda) \\ D(dk, acc) = 1 & : \ (apk, avk, dk) \leftarrow I(pp_{\mathsf{ac}}, pp_{\mathsf{pc}}) \\ \exists_{i\in[k]} \mathcal{V}_{\mathsf{Eval}}(pp_{\mathsf{pc}}, X_i) \neq 1 \vee \exists_{i\in[\ell]} D(dk, acc_i) \neq 1 & \{X_i\}_{i=1}^k, \{acc_i\}_{i=1}^\ell, acc, \pi_V \leftarrow \mathcal{A}(pp_{\mathsf{ac}}, pp_{\mathsf{pc}}) \end{array}\right]$$

The fact that a non-interactive public aggregation scheme gives an accumulation scheme is an immediate consequence of the definitions. The algorithms G and I are trivial, setting all parameters to pp_{pc}. Each $acc = (\mathsf{C}, x, y, \pi)$ is an Eval tuple. The prover $P(pp_{\mathsf{pc}}, \{X_i\}_{i=1}^k, \{acc_i\}_{i=1}^\ell)$ first sets $\mathbf{C} \in \mathbb{G}^{k+\ell}$ so that $\mathsf{C}_i = X_i$ for $i \in [k]$ and $\mathsf{C}_i = acc_{i-k}$ for $i > k$, sets $\boldsymbol{\pi}$ so that the ith and $(i+k)$th components are the Eval proofs in X_i and acc_i respectively, and sets $(\mathbf{s}, \mathbf{t}) \in \mathbb{F}^{k+\ell} \times \mathbb{F}^{k+\ell}$ so that $(s_i, t_i) = (x_i, y_i)$ from X_i for $i \in [k]$ and from acc_i for $i > k$. It runs $\mathsf{Aggregate}(pp_{\mathsf{pc}}, \boldsymbol{\pi}, \mathbf{C}, \mathbf{s}, \mathbf{t})$ to get $(\mathsf{open}^*, f^*, \mathsf{C}^*, s^*, t^*, \pi_{\mathsf{agg}})$ and $\mathsf{Eval}(\mathsf{open}^*, f^*, \mathsf{C}^*, s^*, t^*)$ to get π^*. It returns $\pi_V := \pi_{\mathsf{agg}}$ and $acc := (\mathsf{C}^*, s^*, t^*, \pi^*)$. $D(pp_{\mathsf{pc}}, acc)$ calls the Eval verifier. Finally, $V(pp_{\mathsf{pc}}, \{X_i\}_{i=1}^k, \{acc_i\}_{i=1}^\ell, acc, \pi_{\mathsf{agg}})$ derives the tuples $(\boldsymbol{\pi}, \mathbf{C}, \mathbf{s}, \mathbf{t})$, parses $acc = (\mathsf{C}^*, s^*, t^*, \pi^*)$, and runs the aggregation verifier $\mathcal{V}_{\mathsf{Aggregate}}(pp_{\mathsf{pc}}, \boldsymbol{\pi}, \mathbf{C}, \mathbf{s}, \mathbf{t}, \mathsf{C}^*, s^*, t^*, \pi_{\mathsf{agg}})$.

Private accumulation A small tweak to Definition 6.1 would make it compatible with private aggregation. The accumulation prover is additionally given as inputs a vector of private states $\{st_i\}_{i=1}^{k+\ell}$ and outputs (st, acc, π_V). The other algorithms and the security definition are unchanged. Constructing this from a private aggregation scheme, the state st will contain the prover's private outputs (open^*, f^*) and each st_i contains an (open_i, f_i) pair.

The PCD compiler of [23] can be adapted to work with private aggregation schemes as well. This only affects the proof size which has size $O(N)$ because it includes the "private" states (openings for polynomials of degree N). Intuitively, the construction of PCD from [23] is not materially affected by using *private* accumulation because each prover node in the DAG distributed computation simply passes its private state to its target nodes as "advice". The advice does not impact the size of the recursive statement, which is only dependent on the size of the accumulation verifier. This variation of the compiler was formally proven in follow-up work [22]. To do so they formally define a "split-accumulation" scheme, which coincides with our informal tweak.

Acknowledgments

This work was funded by NSF, DARPA, a grant from ONR, and the Simons Foundation. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

1. Attema, T., Cramer, R.: Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. CRYPTO 2020, Part III. pp. 513–543 (Aug 2020)
2. Baum, C., Bootle, J., Cerulli, A., del Pino, R., Groth, J., Lyubashevsky, V.: Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. CRYPTO 2018, Part II. pp. 669–699 (Aug 2018)
3. Baum, C., Damgård, I., Larsen, K.G., Nielsen, M.: How to prove knowledge of small secrets. CRYPTO 2016, Part III. pp. 478–498 (Aug 2016)
4. Bellare, M., Goldreich, O.: On defining proofs of knowledge. CRYPTO'92. pp. 390–420 (Aug 1993)
5. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. ICALP 2018. pp. 14:1–14:17 (Jul 2018)
6. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. CRYPTO 2019, Part III. pp. 701–732 (Aug 2019)
7. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.: Fast reductions from RAMs to delegatable succinct constraint satisfaction problems: extended abstract. ITCS 2013. pp. 401–414 (Jan 2013)
8. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. EUROCRYPT 2019, Part I. pp. 103–128 (May 2019)

9. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. TCC 2016-B, Part II. pp. 31–60 (Oct / Nov 2016)
10. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. CRYPTO 2014, Part II. pp. 276–294 (Aug 2014)
11. Ben-Sasson, E., Goldberg, L., Kopparty, S., Saraf, S.: DEEP-FRI: Sampling outside the box improves soundness. Cryptology ePrint Archive, Report 2019/336 (2019)
12. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. ITCS 2012. pp. 326–349 (Jan 2012)
13. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. 45th ACM STOC. pp. 111–120 (Jun 2013)
14. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct non-interactive arguments via linear interactive proofs. TCC 2013. pp. 315–333 (Mar 2013)
15. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive, Report 2020/081 (2020)
16. Bonneau, J., Meckler, I., Rao, V., Shapiro, E.: Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Report 2020/352 (2020)
17. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. EUROCRYPT 2016, Part II. pp. 327–357 (May 2016)
18. Bowe, S., Grigg, J., Hopwood, D.: Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021 (2019)
19. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. 2018 IEEE Symposium on Security and Privacy. pp. 315–334 (May 2018)
20. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. EUROCRYPT 2020, Part I. pp. 677–706 (May 2020)
21. Bünz, B., Maller, M., Vesely, N.: Efficient proofs for pairing-based languages. Cryptology ePrint Archive, Report 2019/1177 (2019)
22. Bünz, B., Chiesa, A., Lin, W., Mishra, P., Spooner, N.: Proof-carrying data without succinct arguments. Cryptology ePrint Archive, Report 2020/1618 (2020)
23. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Proof-carrying data from accumulation schemes. Cryptology ePrint Archive, Report 2020/499 (2020)
24. Canetti, R., Chen, Y., Holmgren, J., Lombardi, A., Rothblum, G.N., Rothblum, R.D., Wichs, D.: Fiat-Shamir: from practice to theory. 51st ACM STOC. pp. 1082–1090 (Jun 2019)
25. Canetti, R., Chen, Y., Reyzin, L., Rothblum, R.D.: Fiat-Shamir and correlation intractability from strong KDM-secure encryption. EUROCRYPT 2018, Part I. pp. 91–122 (Apr / May 2018)
26. Chen, Y., Lombardi, A., Ma, F., Quach, W.: Does fiat-shamir require a cryptographic hash function? Cryptology ePrint Archive, Report 2020/915 (2020)
27. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. Cryptology ePrint Archive, Report 2019/1047 (2019)
28. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. EUROCRYPT 2020, Part I. pp. 738–768 (May 2020)

29. Chiesa, A., Liu, S.: On the impossibility of probabilistic proofs in relativized worlds. ITCS 2020. pp. 57:1–57:30 (Jan 2020)
30. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. EUROCRYPT 2020, Part I. pp. 769–793 (May 2020)
31. Chiesa, A., Tromer, E.: Proof-carrying data and hearsay arguments from signature cards. Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings. pp. 310–331 (2010), http://conference.iiis.tsinghua.edu.cn/ICS2010/content/papers/25.html
32. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019)
33. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. EUROCRYPT 2013. pp. 626–645 (May 2013)
34. Goldreich, O., Krawczyk, H.: On the composition of zero-knowledge proof systems. SIAM Journal on Computing **9**, 169–192 (1996)
35. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. ASIACRYPT 2010. pp. 321–340 (Dec 2010)
36. Groth, J.: On the size of pairing-based non-interactive arguments. EUROCRYPT 2016, Part II. pp. 305–326 (May 2016)
37. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. CRYPTO 2017, Part II. pp. 581–612 (Aug 2017)
38. Holmgren, J.: On round-by-round soundness and state restoration attacks. Cryptology ePrint Archive, Report 2019/1261 (2019)
39. Kalai, Y.T., Rothblum, G.N., Rothblum, R.D.: From obfuscation to the security of Fiat-Shamir for proofs. CRYPTO 2017, Part II. pp. 224–251 (Aug 2017)
40. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. ASIACRYPT 2010. pp. 177–194 (Dec 2010)
41. Kattis, A., Panarin, K., Vlasov, A.: RedShift: Transparent SNARKs from list polynomial commitment IOPs. Cryptology ePrint Archive, Report 2019/1400 (2019)
42. Labs, O.: Coda protocol (2018)
43. Lee, J.: Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. Cryptology ePrint Archive, Report 2020/1274 (2020)
44. Lipmaa, H.: Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. TCC 2012. pp. 169–189 (Mar 2012)
45. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. ACM CCS 2019. pp. 2111–2128 (Nov 2019)
46. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. 2013 IEEE Symposium on Security and Privacy. pp. 238–252 (May 2013)
47. Pointcheval, D., Stern, J.: Security proofs for signature schemes. EUROCRYPT'96. pp. 387–398 (May 1996)
48. Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. CRYPTO 2020, Part III. pp. 704–737 (Aug 2020)
49. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. TCC 2008. pp. 1–18 (Mar 2008)
50. Vlasov, A., Panarin, K.: Transparent polynomial commitment scheme with polylogarithmic communication complexity. Cryptology ePrint Archive, Report 2019/1020 (2019)