Halo Infinite: Recursive zk-SNARKs from any Additive Polynomial Commitment Scheme*

Dan Boneh Justin Drake Ben Fisch Ariel Gabizon

Abstract

A polynomial commitment scheme (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. Recently, it has been shown that any efficient and succinct PCS can be used to construct a SNARK with similar security and efficiency characteristics. We define an *additive* PCS to capture a "homomorphic" property of commitments over a computational group $\mathbb{G}$ of bounded size. All existing examples of additive schemes (e.g., Bulletproofs, KZG, DARK, Dory) are also what we call *m-spanning*, meaning that commitments to the monomials of degree less than m generate $\mathbb{G}$. Our first technical result is a black-box transformation of any m-spanning additive PCS into a hiding PCS with a zero-knowledge evaluation proof. Our second technical result is that every additive succinct PCS supports efficient *proof aggregation*.

PCS proof aggregation reduces the task of proving evaluations of multiple commitments at multiple independent points to the task of proving the evaluation of a single "aggregate" commitment at a single point. We present two flavors of aggregation: private and public. In private aggregation the prover has a private witness consisting of openings of the input commitments. In public aggregation, the prover/verifier share the same inputs, which includes non-interactive evaluation proofs for each input commitment. Our public aggregation protocol applies to any additive succinct PCS. Our private aggregation protocol applies more broadly to any succinct PCS that supports an efficient *linear combination scheme*: a protocol for verifiably aggregating commitments into a new commitment to their linear combination. This includes non-additive schemes such as the post-quantum FRI-based PCS.

We apply these results to the Halo *proof carrying data* (PCD) system. Halo was originally built using the Bulletproofs inner-product argument as the underlying PCS, and was recently generalized to work with the KZG PCS. We show that Halo can be instantiated with any PCS that supports efficient PCS aggregation, private or public. Thus, our results show that efficient (zero-knowledge) SNARKs and PCD can be constructed from any succinct PCS that has an efficient linear combination scheme, even if the PCS itself is inefficient. These results yield new Halo-like PCD systems from PCS constructions beyond Bulletproofs and KZG, including DARK, FRI, and Dory. The post-quantum Halo instantiation from FRI is particularly surprising as FRI is not additive.

Contents

*This paper subsumes the results of an earlier unpublished preprint by Boneh, Drake, Fisch and Gabizon[BDFG20]

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) [KZG10], and has emerged as a key tool for building succinct and efficient non-interactive argument systems called SNARKs [BCCT12]. A succinct and efficient PCS can be used to compile an information theoretic interactive proof system known as a *Polynomial Interactive Oracle Proof* [BFS20] (PIOP), or equivalently *Algebraic Holographic Proofs* [CHM+20]), 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 [MBKM19, CHM+20, GWC19] or no trusted setup [BFS20, COS20, Set20, KPV19].

The original PCS, called the KZG PCS [KZG10], 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 [BMV19]). Another PCS, called the Bulletproofs PCS [BCC+16, BBB+18], 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 [BFS20] is based on groups of unknown order, and very recently Dory [Lee20] uses pairing-based commitments and generalized inner-product arguments [BMV19]. A post-quantum efficient and succinct PCS without trusted setup can be built using FRI [VP19, KPV19, BGKS19]. In practice, these schemes all have very different performance profiles and properties.

A proof-carrying data (PCD) system [CT10, BCCT13] 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 [Lab18, BMRS20], 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 [Val08], 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 [BCGT13].

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 SNARK constructions. We show that the PCS schemes mentioned above (KZG, Bulletproofs, DARK, FRI, Dory) 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.

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.

In Section 4 we show that every additive PCS that additionally satisfies a technical condition we call ***m*-spanning**, can be compiled into a hiding PCS with a zero-knowledge Eval protocol. m-spanning means that commitments to polynomials of degree at most m generate $\mathbb{G}$. Our compilation is generic, relying only on the additive property. Four of the aforementioned examples (Bulletproofs, Dory, KZG, and DARK) are 1-spanning. Our compiler is a generalization of the technique used to make the DARK PCS zero-knowledge [BFS20], and also has its roots in Zero-Knowledge Sumcheck [CFS17].

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

The formal statement of this result is in Theorem 4 and Theorem 7. An important example of a linearly amortizable non-additive PCS is the FRI-based PCS, as explained in Section 3.2. We thus obtain a non-trivial private aggregation scheme for the FRI-based PCS. Along the way to proving Theorem 7 we present a generic succinct proof of knowledge of the classical homomorphism pre-image problem, which has its roots in the Bulletproofs protocol.

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 5). Second, in Section 7 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 [BGH19], which was also formalized and generalized by Bünz et. al. [BCMS20].

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 [BCGT13, COS20] whenever the verification complexity of the private aggregation is smaller than the verification complexity of Eval.

This includes the post-quantum FRI scheme, where our aggregation verifier is asymptotically a factor $\Omega(\log n)$ smaller than the FRI verifier for polynomials of degree n. This may lead to an order of magnitude reduction in the recursion threshold over the prior FRI-based PCD system Fractal [COS20]. We leave concrete performance analysis for future work.

Theorem 2 (informal)**.** *PCD, IVC, and preprocessing SNARKs can be constructed from any efficient PIOP combined with any PCS that has an efficient linear combination 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 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 [CL20].

Batch evaluation for the KZG scheme and applications to pairing based zkSNARKs In Appendix B, we focus specifically on the original PCS of Kate, Zaverucha and Goldberg [KZG10]. As our results apply to all additive schemes, one can naturally instantiate the batch evaluation scheme described in Section 5 for KZG. We show that for KZG, a more efficient batch evaluation in terms of prover communication in the Eval procedure is possible, at the expense of extra verifier operations. This exploits the "multiplicative" nature of KZG that makes it possible to check, given commitments to four polynomials f, g, h, z, whether $f \cdot g \equiv h \cdot z$. This check is done using a single pairing computation over the given commitments. As an application of this batch evaluation, we reduce the proof length and prover run time of the `PLONK` zk-SNARK [GWC19], at the expense of one extra verifier pairing.

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 [Gro10, Lip12, BCCT13, GGPR13, PHGR13, BCI$^+$13, Gro16a, GM17, MBKM19, GWC19, CHM$^+$20, BBHR19, BFS20, COS20, BGH19, Set20]. In addition to the PCS constructions mentioned earlier, there is also a scheme by Bootle et. al. 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. [BDLN16, BBC$^+$18]. In Section 6 we describe a construction of a PCS from any collision-resistant homomorphism based on our succinct proof of homomorphism pre-images (HPI) that has constant size commitment, logarithmic size proofs and linear verification time.[1] Attema and Cramer [AC20] concurrently 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

[1]This can be combined with the technique of Bootle et. al. 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.

the statement being proven (i.e., SNARKs) [Val08, BCCT13, BCTV14, COS20]. The Halo protocol [BGH19, BCMS20] was the first construction of PCD from an underlying inefficient proof system (combining the Sonic PIOP [MBKM19] 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. [BCMS20] 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 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 as they show that the Halo PCD construction paradigm can be applied with any additive PCS, and more generally any PCS with an efficient linear combination scheme.

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. 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. See §A.1 for the formal definition.

2.1 Interactive protocols

A two-party interactive protocol named Label between parties A and B, where a and b are private inputs to A and B respectively and out_A and out_B are their private outputs, is denoted:

$$\mathsf{Label}\big(A(a), B(b)\big) \to (\mathsf{out}_A, \mathsf{out}_B)$$

Formally, an interactive protocol is a pair of interactive Turing machines. When party A (resp. B) *runs in time* t this formally means that the interactive Turing machine of party A (resp. B) runs for t steps.

In a **public-coin** protocol party B does not have any private inputs (i.e., b is a common public input) and all of B's messages to A are i.i.d. from a message space $\mathcal{X}$ called the challenge space. The output out_B is always common to both parties in this case. The **Fiat-Shamir (FS) transform** [FS87] is a transformation of a public-coin interactive protocol into a non-interactive algorithm. The transformed protocol consists of two parts. First:

$$\mathsf{NI\text{-}Label}(a, b) \rightarrow (\mathsf{out}_A, \mathsf{out}_B, \mathsf{tr})$$

simulates a transcript of the messages sent between parties A and B by deriving each of B's random "challenges" as the hash of the partial transcript preceding the challenge. Second:

$$\mathcal{V}_{\mathsf{Label}}(b, \mathsf{tr}, \mathsf{out}_B) \rightarrow \delta \in \{0, 1\}$$

replays the transcript tr to check its validity and consistency with the output.[2] In the new non-interactive protocol, party A runs $\mathsf{NI\text{-}ProtocolName}(a, b)$ and sends to party B only $(\mathsf{out}_B, \mathsf{tr})$, who then runs $\mathcal{V}_{\mathsf{Label}}(b, \mathsf{tr}, \mathsf{out}_B)$.

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[3] to $\mathcal{A}$. See §A.3 for the formal definition.

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 [GK96, PS96]. 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* [BCS16], also equivalent to *round-by-round soundness* [CCH+19, Hol19]. There are also special classes of constant-round protocols for which the Fiat-Shamir transform can be instantiated using correlation-intractable hash functions [KRR17, CCRR18, CCH+19], or even simpler non-cryptographic hash functions [CLMQ20]. 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 1. *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 $(\mathcal{P}_{FS}, \mathcal{V}_{FS})$ obtained from applying Fiat-Shamir using an explicit hash sampled from $\mathcal{H}$ is knowledge-sound.*

[2] It checks that each of the challenges in the simulation are the correct hash values of transcript prefixes and that (b, out_B) are the correct outputs consistent with tr. Since the protocol is public-coin we may assume that party B's output is a deterministic function of the protocol transcript and party B's input.

[3] 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.

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). See §A.3 for the formal definition of HVZK and §A.2 for relevant background on probability distributions.

2.2 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).

- $\mathsf{Eval}\big(\mathcal{P}(f, \mathsf{open}), \mathcal{V}(pp, \mathsf{C}, z, y)\big) \to (\perp, b)$ 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 $o(d \cdot \log|\mathbb{F}|)$, i.e. sublinear in the size of the committed polynomial, then the PCS is called **efficient**. If both the size of commitments and communication complexity of the Eval protocol are $o(d \cdot \log|\mathbb{F}|)$ then the scheme is called **succinct**.

A PCS could be succinct and not efficient. One example is a PCS based on the Bulletproofs system [BCC+16, BBB+18]. 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 [KZG10]) 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 (Binding PCS)**.** *A PCS is **binding** if for all PPT adversaries $\mathcal{A}$:*

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

Definition 3 (Knowledge soundness)**.** *A PCS has **knowledge soundness** if for all pp output by $\mathit{Setup}(\lambda, d)$ and $d \in \mathbb{N}$, the interactive public-coin protocol Eval is a* proof of knowledge *for the NP relation $\mathcal{R}_{\mathit{Eval}}(pp, d)$ defined as follows:*

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

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 [BFS20], 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:

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

If $\mathbb{H}$ is finite then bounded witness ZK with $B = max_{w \in \mathbb{H}} \ \mathcal{N}(w)$ is equivalent to plain ZK. Moreover, the set of opening hints returned as primary outputs of the Commit algorithm *is* a finite subset of $\mathbb{H}$ because the input domain $\mathbb{F}^{(<d)}[X]$ and the randomness of the algorithm are finite. The distinction between bounded witness ZK and plain ZK is particularly relevant for additive schemes. If the PCS only satisfies bounded witness ZK then the evaluation protocol may enable the verifier to distinguish between primary commitments and linear combinations of commitments, but would not reveal any further information about the committed $f \in \mathbb{F}^{(<d)}[X]$ other than $f(z) = y$.

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 $h = t \cdot f \bmod p$ for a integer $t \in \mathbb{Z}$ as a valid opening of C_f to the polynomial $f \bmod p$.

Lemma 1. *Let $\mathcal{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify}, \mathsf{Eval})$ denote a PCS. 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 $h = t \cdot f \bmod p$, then the new PCS is still binding.*

Proof. Suppose an adversary outputs a commitment C and two openings $(f_1, (t_1, \mathsf{open}_1))$ and $(f_2, (t_2, \mathsf{open}_2))$ 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 $t_1 h_2 \neq t_2 h_1 \bmod p$. Thus, this would contradict the binding property of the original PCS. $\square$

Note that this relaxation does not affect the "honest" opening output by Commit, which will still be accepted by Verify. It is only helpful as a proof technique so that the knowledge extractor is allowed to obtain a more general opening of C_f.

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 (non-standard) definition from Section 2.2, 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 4. *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 (Definition 11) of size at most $2^{\mathsf{poly}(\lambda)}$. An additive PCS for polynomials in $\mathbb{F}^{(<d)}[X]$ is **additively succinct** if the size of $\mathbb{G}_{\lambda,d}$ is $o(|\mathbb{F}|^d)$.*

A consequence of additive succinctness is that linear combinations of commitments remain succinct. Under the restriction that $|\mathbb{G}_{\lambda,d}|$ is $o(|\mathbb{F}|^d)$, sums of commitments to polynomials in $\mathbb{F}^{(<d)}[X]$ will have representation size sublinear in the representation size of the sums of committed polynomials, no matter the number of summands.

There are examples of schemes where the commitments are contained within a group $\mathbb{G}$ of bounded size, but the add^* operation only works for a bounded number of operations (e.g., the valid commitments are not a subgroup). Examples include DARK and lattice-based schemes [BFS20, BBC+18]. These schemes are not additive as $\mathbb{G}$ does not satisfy our definition of a commitment group, but we call a scheme with this property a *bounded additive* PCS.

Definition 5. *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 $\chi(\mathsf{open}) = f$.*

Just as we call $\mathbb{G}$ the "commitment" group 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). $\mathsf{Commit}(pp, f)$ selects a representation of f in $\mathbb{H}$ and applies the homomorphism ϕ to derive the commitment. Given $(\mathsf{C}_f, \mathsf{open}_f) \in \mathbb{G} \times \mathbb{H}$ and $(\mathsf{C}_h, \mathsf{open}_h) \in \mathbb{G} \times \mathbb{H}$, the commitment $\mathsf{C}_z := \mathsf{C}_f + \mathsf{C}_h$ and opening $\mathsf{open}_z := \mathsf{open}_f + \mathsf{open}_z$ satisfy $\mathsf{C}_z = \phi(\mathsf{open}_f + \mathsf{open}_h) = \phi(\mathsf{open}_z)$ and $\chi(\mathsf{open}_z) = \chi(\mathsf{open}_f) + \chi(\mathsf{open}_h) = f + h$.

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 $\phi : \mathbb{Z}^d \to \mathbb{G}$. This is described in Section 4. In fact, we further show how an additive PCS may be transformed into a hiding homomorphic PCS provided that commitments to the first $m < d$ monomials generate $\mathbb{G}$.

Definition 6. *A PCS is called m-spanning if for any $\lambda, d \in \mathbb{N}$ and $pp \leftarrow \mathsf{Setup}(\lambda, d)$ the commitments $(\mathsf{C}_i, \mathsf{open}_i) \leftarrow \mathsf{Commit}(pp, X^{i-1})$ for $i \in [1, m]$ generate $\mathbb{G}$, i.e. $\langle \mathsf{C}_1, ..., \mathsf{C}_m \rangle = \mathbb{G}$. A **spanning PCS** is m-spanning for some $m > 0$.*

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 single aggregate commitment and later open this at points.

Definition 7 (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:*

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

The public output is $(C^*, b) \in \mathbb{G} \times \{0, 1\}$ where $|C^*| = \mathsf{poly}(\lambda)$ independent of ℓ, and $b \in \{0, 1\}$ 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:*

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

The requirement that C^* has size $\mathsf{poly}(\lambda)$ (independent of ℓ) rules out the linear combination protocol that simply outputs a formal sum. 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. Note that the security property does not require the linear combination protocol itself to be a proof of knowledge of openings for the input commitments. Every PCS in fact has a generic size-optimal linear combination protocol. Given polynomials f and g and commitments C_f and C_g, it is always possible to compute a fresh commitment C_h to $h := f + g$, and prove the linear relation between the three commitments by running Eval three times, once on each commitment, at a common point $\rho \in \mathbb{F}$ randomly chosen by the verifier (or non-interactively generated via Fiat-Shamir). The verifier would check that $f(\rho) + g(\rho) = h(\rho)$. For a knowledge-sound Eval, this is a proof of knowledge of openings of the three commitments to f, g, and h satisfying $f + g = h$.

A linear combination protocol is interesting when more efficient than the generic (trivial) one. A linear combination scheme for an additive PCS over $\mathbb{F}_p$ simply outputs the linear combination of the input commitments over $\mathbb{G}$. This also works for bounded additive schemes over $\mathbb{F}_p$ as long as add^* can compute openings for linear combinations of ℓ commitments with integer coefficients in the range $[0, p)$.

Amortization ratios The **amortization ratios** $(\epsilon(\ell), \delta(\ell))$ of a linear combination scheme characterize the amortization benefit of using $\mathsf{LinCombine}$ together with Eval to open a linear combination of commitments at a point. The trivial protocol that runs a separate Eval on each input achieves no amortization and thus has ratios $\epsilon = 1$ and $\delta = 1$. The values $\epsilon(\ell)$ is the communication complexity ratio of the $\mathsf{LinCombine}$ method versus the trivial protocol, and $\delta(\ell)$ is the verification complexity ratio. We may refer to a single ratio when $\epsilon = \delta$. When both amortization ratios are $o(1)$ we say that the PCS is **linearly amortizable**.

Verifier efficiency 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. The verifier efficiency of a linear combination scheme and its amortization ratio(s) are distinct characteristics. If the PCS is efficient then even the trivial linear combination scheme is efficient. If the PCS is inefficient, it is possible for a linear combination scheme to be inefficient and still have amortization ratios less than 1.

3.2 PCS examples and their additive properties

The table below summarizes the properties of several schemes. All the major examples of PCS constructions are linearly amortizable. The linear combination scheme (LCS) amortization ratio (column 3) is most relevant for the efficiency of batch evaluation. The complexity ratio of the LCS verifier to the Eval verifier (column 5) is most relevant for the efficiency[4] of the proof recursion

[4]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.

application discussed in Section 7. Recall that ℓ is the number of polynomial commitments being linearly combined, and d is their maximum degree.

	additive	LCS size-optimal	LCS amortization	$\lvert\mathcal{V}_{\mathsf{LinCombine}}\rvert$	$\frac{\lvert\mathcal{V}_{\mathsf{LinCombine}}\rvert}{\lvert\mathcal{V}_{\mathsf{Eval}}\rvert}$
Bulletproofs	yes	yes	$1/\ell$	$O_\lambda(\ell)$	$\ell/\Omega(d)$
Dory	yes	yes	$1/\ell$	$O_\lambda(\ell)$	$\ell/\Omega(\log d)$
KZG	yes	yes	$1/\ell$	$O_\lambda(\ell)$	$\ell/\Omega(1)$
DARK	bounded	yes	$1/\ell$	$O_\lambda(\ell)$	$\ell/\Omega(\log d)$
FRI	no	no	$\frac{1}{\ell} + \frac{1}{\Omega(\log d)}$	$O_\lambda(\ell \log d)$	$\ell/\Omega(\log d)$

See §A.6 for an overview of the schemes.

FRI: a non-additive PCS The Fast Reed-Solomon IOP of Proximity (FRI) [BBHR18] 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. See §A.7 for more background on FRI.

The FRI PCS is not additive by Definition 4, but it does have a batch evaluation protocol that achieves amortized efficiency ratio of $\frac{1}{\ell} + \frac{1}{\Omega(\log d)}$ over ℓ commitments (see Algorithm 8.2 of Aurora [BCR+19]). Similarly, the FRI PCS also has an efficient LCS. The aggregate commitment contains a fresh commitment to the linear combination of input committed polynomials. The protocol proves that this fresh committed codeword is close to the linear combination of input codewords. The amortization ratio comes from the fact that proving proximity of vectors is a factor $\log d$ more efficient than proving proximity to an RS codeword, which is only demonstrated once FRI Eval is run on the aggregate commitment. For technical reasons explained in §A.7 the aggregate commitment also needs to include a second commitment and is therefore not size-optimal.

4 Zero knowledge compiler for any additive PCS

In this section we describe a generic compiler for transforming any m-spanning additive PCS over $\mathbb{F}_p$ (Definition 6) into a hiding PCS with a zero-knowledge Eval. For the remainder of this section $\mathbb{F} := \mathbb{F}_p$, for some prime number p.

Theorem 3. *There is a generic compiler that takes any m-spanning additive PCS over $\mathbb{F}_p$ and transforms it into a statistically-hiding homomorphic PCS whose evaluation protocol satisfies bounded witness zero-knowledge. The new PCS commitment group $\mathbb{G}$ on setup parameters (λ, d) is equal to the original input PCS group for parameters $(\lambda, d+m)$. The new evaluation protocol has the following efficiency, with $\Delta := max(m, \lambda + \log|\mathbb{G}|)$:*

- *The communication overhead is $O(\Delta)$, consisting of an integer vector of size at most Δ bits and a constant number of elements in $\mathbb{G}$ and $\mathbb{F}$.*
- *The verification overhead is $O(\Delta)$ group operations in $\mathbb{G}$.*
- *The prover overhead is $O(\Delta + d)$ group operations in $\mathbb{G}$ and a constant number of field operations in $\mathbb{F}$.*

The proof of Theorem 3 is covered by Lemma 2 and Lemma 3.

4.1 Compiler I: From Additive to Homomorphic

The first step is to transform the non-hiding additive PCS into a homomorphic PCS. The steps of this transformation were explained briefly in Section 3. Let (Setup, Commit, Verify, Eval) denote the protocols of the non-hiding additive PCS. Since the PCS is non-hiding we assume (without loss of generality) that the commitment algorithm Commit is a deterministic function. The new homomorphic non-hiding PCS has the same $\mathsf{Setup}, \mathsf{Verify}$, and Eval protocols and only modifies the commitment algorithm Commit^* as follows:

$\mathsf{Commit}^*(pp, f) \to (\mathsf{C}, \mathsf{open})$: on inputs pp and $f \in \mathbb{F}^{(<d)}[X]$, finds the integer coefficient vector representation $(\hat{f}_0, ..., \hat{f}_{d-1}) \in [0, p)^d$ of f and

1. Runs $(\mathsf{C}_i, \mathsf{open}_i) \leftarrow \mathsf{Commit}(pp, X^{i-1})$ for $i \in [1, d]$
2. Sets $\mathsf{C} := \sum_{i=0}^{d-1} \hat{f}_i \cdot \mathsf{C}_i$ and $\mathsf{open} := (\hat{f}_0, ..., \hat{f}_{d-1})$.

By definition, C is also a valid commitment to the polynomial f under the original scheme. The prover uses the algorithm add^* to derive a valid opening string open' for C to f, i.e. such that $\mathsf{Verify}(pp, f, \mathsf{open}', \mathsf{C}) = 1$. For the evaluation protocol, the prover uses open' and runs the Eval protocol of the original scheme. Note that open' may *not* be the same as open, but can always be computed from open using add^*. 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 $\mathsf{open}' = \mathsf{open}$. In other words, the transformation described above has 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 $f \in \mathbb{F}^{(<d)}[X]$. If the additive PCS is m-spanning then the homomorphism $\phi_m : \mathbb{Z}^m \to \mathbb{G}$ given by $\mathbf{v} \mapsto \sum_{i=1}^{m} v_i \cdot \mathsf{C}_i$ is surjective. The new scheme is binding: given a collision $\hat{\mathbf{f}}' \neq \hat{\mathbf{f}} \bmod p$ such that $\mathsf{C} = \phi(\hat{\mathbf{f}}) = \phi(\hat{\mathbf{f}}')$, 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.

4.2 Compiler II: From Homomorphic to Hiding

The parameters of the compilation are $(m, \kappa) \in \mathbb{N}$. The input to the compiler is a non-hiding m-spanning homomorphic PCS with the following characteristics for any $pp \leftarrow \mathsf{Setup}(\lambda, d)$:

- $\mathsf{Commit}(pp, f)$ is deterministic (w.l.o.g., since it is non-hiding).
- $\mathbb{H} = \mathbb{Z}^d$ and $\chi(\mathbf{f})$ returns the unique polynomial in $\mathbb{F}^{(<d)}[X]$ with coefficient vector $\mathbf{f} \bmod p$.
- $(g_i, e_i) \leftarrow \mathsf{Commit}(pp, X^i)$ where e_i is the ith standard basis vector in $\mathbb{Z}^D$. For all $\mathbf{f} \in \mathbb{Z}^d$, $\phi(\mathbf{f}) = \sum_{i=1}^{d} f_i \cdot g_i$.

Any m-spanning additive PCS that is first passed through Compiler I has these characteristics.

Hiding: 1-spanning case As a warm-up, we first describe a simple transformation that works when $\mathbb{G}$ is cyclic, $m = \log |\mathbb{G}|/\kappa$, and $g_1, ..., g_m$ are each generators of $\mathbb{G}$. The new setup $\mathsf{Setup}^*(\lambda, d)$ will run $\mathsf{Setup}(\lambda, d + m)$, which defines $\phi : \mathbb{Z}^{d+m} \to \mathbb{G}$. The new hiding commitment to $\mathbf{f} = (f_0, ..., f_{d-1}) \in [0, p)^d$ samples a random degree m blinding polynomial $r \in \mathbb{F}^{(<m)}[X]$ and outputs a non-hiding commitment to the polynomial $f' := r + X^m f$ using the original commitment algorithm.

More precisely, it samples a random integer vector $\mathbf{r} \in [0, 2^\kappa)^m$, sets $\mathsf{open} := (\mathbf{r}, \mathbf{f}) \in \mathbb{Z}^{d+m}$, and returns $\mathsf{C} := \phi(\mathsf{open})$. The new scheme is still homomorphic with $\mathbb{H} = \mathbb{Z}^{d+m}$, the same homomorphism ϕ, and a new homomorphism $\chi^* : \mathbb{H} \to \mathbb{F}^d$ such that $\chi(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_2 \bmod p$ for any $\mathbf{x}_1 \in \mathbb{Z}^m$ and $\mathbf{x}_2 \in \mathbb{Z}^d$.

The commitment is hiding if the random variable $Z(r_1, ..., r_m) = \sum_{i=1}^m r_i \cdot g_i$ is indistinguishable from uniform $\mathbb{G}$ when $\mathbf{r}$ is sampled uniformly from $[0, 2^\kappa)^m$. By a classical theorem, when g_i are generators of $\mathbb{G}$ and $r_i \xleftarrow{\$} [0, 2^\kappa]$, the random variable $Z(r_1, ..., r_m)$ converges to uniform for $m \in O(\log |\mathbb{G}|/\kappa)$ [Bab91, Coo02, Dix08]. This is the case in all the examples of additive schemes from Section 3.2.

Hiding: m-spanning case In the more general case of an m-spanning scheme (i.e., where g_i are not necessarily generators but $\langle g_1, ..., g_m \rangle = \mathbb{G}$), it suffices to use commitments to a random polynomial basis of $\mathbb{F}^{(<m)}[X]$ to generate the blinding factor instead of commitments to the monomial basis. Let $D := d + m$ and suppose $\mathbb{G}$ has order[5] q. A matrix $\mathbf{A} \xleftarrow{\$} [0, q)^m$ is randomly sampled and included in the setup parameters or generated from a seed. With overwhelming probability $\mathbf{A} \in GL_m(\mathbb{F})$. The new commitment algorithm to $f \in \mathbb{F}^{(<d)}[X]$ with coefficient vector $\mathbf{f} \in \mathbb{Z}^d$ samples $\mathbf{r} \xleftarrow{\$} [0, 2^\kappa)^m$, sets $\mathsf{open} := (\mathbf{A} \cdot \mathbf{r}, \mathbf{f})$ and returns $\phi(\mathsf{open})$. This is interpreted as the sum of a commitment to $X^m \cdot f \in \mathbb{F}^{(<D)}[X]$ and a commitment to the blinding polynomial $\sum_{i=1}^m r_i \cdot a_i(X)$ where $a_i(X) \in \mathbb{F}^{(<m)}[X]$ has coefficients equal to the ith column of $\mathbf{A}$. To avoid the $O(m^2)$ computation $\mathbf{A} \cdot \mathbf{r}$ for each new commitment, the setup parameters may also include the preprocessed group elements $\tilde{g}_j = \sum_{i=1}^m a_{ij} \cdot g_i$ for $i \in [m]$ so that $\mathsf{C} = \sum_{i=1}^m r_i \cdot \tilde{g}_i + \sum_{i=1}^d f_i \cdot g_{i+m}$. This commitment is statistically hiding for $m \cdot \kappa \geq \lambda + \log |\mathbb{G}|$ based on the Leftover Hash Lemma [HILL99].

Evaluation We give a succinct non-ZK evaluation protocol for the transformed hiding commitment where the increase in communication over the original protocol is $O(m\kappa)$.[6] The next section gives a compilation from a hiding PCS with a non-ZK evaluation protocol into one with a ZK evaluation protocol. Given $\mathsf{open} = (\mathbf{r}, \mathbf{f})$, the prover first sends $\mathbf{r}$ to the verifier, who can then derive $\mathsf{C}' := \mathsf{C} - \sum_{i=1}^m r_i \cdot \tilde{g}_i$, which is a valid non-hiding commitment to $X^m \cdot f$ under the original PCS. The prover additionally sends a non-hiding commitment C_f to f. The verifier samples $\rho \in \mathbb{F}$ and they run the evaluation protocol of the original PCS to open C_f at ρ to $f(\rho)$ and C' to the value $\rho^m \cdot f(\rho)$. Finally, they also run Eval on C_f at z to open $f(z) = y$. The three evaluations can be batched.

Lemma 2. *The PCS returned by the hiding compiler in Figure 1 is binding, knowledge-sound, and statistically hiding for $m\kappa \geq \lambda + \log |\mathbb{G}|$.*

See §A.8.1 for the proof.

Examples The following examples of this transformation cover the common cases of homomorphic commitment schemes. In all these examples, no basis transformation is necessary (i.e., the matrix $\mathbf{A}$ is the identity matrix).

[5] If the order q of the group $\mathbb{G}$ is unknown, an upper bound on q may be derived from the element representation size, and the matrix $\mathbf{A}$ can still be sampled using integers from a sufficiently large range such that $\mathbf{A} \bmod q$ is statistically indistinguishable from random over $\mathbb{Z}_q$.

[6] To reduce the communication overhead, the prover could send $\mathsf{C}_r = \phi(\mathbf{A} \cdot \mathbf{r}, \mathbf{0}^m)$ instead of $\mathbf{r}$ and run the evaluation protocol to prove this is a commitment to a polynomial of degree at most m.

Figure 1: **Compiler III (hiding)** Parameters $m = m(\lambda)$ and $\kappa = \kappa(\lambda)$ are functions. The compiler uses a sampling algorithm $\mathbf{A} \leftarrow \mathsf{Sample}(\lambda, m)$ that returns a matrix in $\mathbb{Z}^{m\times m}$. By default $\mathbf{A}$ is sampled uniformly over $[0, q)^{m\times m}$ where $q = |\mathbb{G}|$. When q is unknown $\mathbf{A}$ may be sampled such that $\mathbf{A}$ mod q is statistically close to uniform. In special cases $\mathbf{A}$ may be the identity matrix.

$\mathsf{Setup}^*(\lambda, d) \rightarrow pp$ sets $\kappa = \kappa(\lambda)$, $m = m(\lambda)$, $D \leftarrow d + m$, $pp' \leftarrow \mathsf{Setup}(\lambda, D)$, $\mathbf{A} \leftarrow \mathsf{Sample}(\lambda, m)$ and outputs $(pp', \mathbf{A})$. For all $j \in [1, D]$, compute $(g_j, e_j) \leftarrow \mathsf{Commit}(pp', X^j)$ and for $i \in [1, m]$ compute $\tilde{g}_j \leftarrow \sum_{i=1}^{m} a_{ij} \cdot g_i$ where a_{ij} is the (i, j)th entry of $\mathbf{A}$. For $\mathbf{x} \in \mathbb{Z}^D$ let $\phi(\mathbf{x}) = \sum_{i=1}^{D} x_i \cdot g_i$.

$\mathsf{Commit}^*(pp, f) \rightarrow (\mathsf{C}, \mathsf{open})$ receives the coefficient vector $\mathbf{f}$ of $f \in \mathbb{F}^{(<d)}[X]$, samples $\mathbf{r} \xleftarrow{\$} [0, 2^\kappa)^m$, sets $\mathsf{open} := (\mathbf{r}, \mathbf{f})$ and $\mathsf{C} := \sum_{i=1}^{m} r_i \cdot \tilde{g}_i + \sum_{i=1}^{d} f_i \cdot g_{i+m}$.

$\mathsf{Verify}^*(pp, f, \mathsf{open}, \mathsf{C})$ returns 1 iff $\mathsf{open} = (\mathbf{r}, \mathbf{f}) \in \mathbb{Z}^m \times \mathbb{Z}^d$, $\mathbf{f}$ mod p is the coefficient vector of f, and $\phi((\mathbf{A} \cdot \mathbf{r}, \mathbf{f})) = \mathsf{C}$.

$\mathsf{Eval}^*\big(\mathcal{P}(f, \mathsf{open}), \mathcal{V}(pp, \mathsf{C}, z, y)\big) \rightarrow (\perp, b)$

The prover parses $\mathsf{open} = (\mathbf{r}, \mathbf{f}) \in \mathbb{Z}^m \times \mathbb{Z}^d$, computes the non-hiding commitment $(\mathsf{C}_f, \mathsf{open}_f) \leftarrow \mathsf{Commit}(pp', f)$, and sends $(\mathsf{C}_f, \mathbf{r})$ to the verifier. Both parties derive $\mathsf{C}_r := \sum_{i=1}^{m} r_i \cdot \tilde{g}_i$ and $\mathsf{C}' := \mathsf{C} - \mathsf{C}_r$. The prover also sets $\mathsf{open}' := (\mathbf{0}^m, \mathbf{f})$. The verifier samples $\rho \xleftarrow{\$} \mathbb{F}$ and sends this to the prover. The prover sends $f(\rho)$. Finally, the parties run the following interactive proofs. The verifier outputs 1 iff $\mathcal{V}$ outputs 1 in each subprotocol:

1. $\mathsf{Eval}\big(\mathcal{P}(f, \mathsf{open}_f), \mathcal{V}(pp', \mathsf{C}_f, z, y)\big)$
2. $\mathsf{Eval}\big(\mathcal{P}(f, \mathsf{open}_f), \mathcal{V}(pp', \mathsf{C}_f, \rho, f(\rho))\big)$
3. $\mathsf{Eval}\big(\mathcal{P}(X^m \cdot f, \mathsf{open}'), \mathcal{V}(pp', \mathsf{C}', \rho, \rho^m \cdot f(\rho))\big)$

All evaluation protocols can be batched using the protocol in Figure 2.

- $\mathbb{G}$ is any group of order $p = |\mathbb{F}|$. In this case the scheme is perfectly hiding for $m = 1$ and $2^\kappa = p$ because g_1 is a generator and $r_1 \cdot g_1$ is uniformly distributed in $\mathbb{G}$. This is the case for several PCS schemes including Bulletproofs and KZG [BCC+16, BBB+18, KZG10].

- In general, when $\mathbb{G}$ is a cyclic group of known order q and g_1 is a generator of $|\mathbb{G}|$ then the scheme is perfectly hiding for $m = 1$ and $2^\kappa = q$.

- If $\mathbb{G}$ is a group of unknown order, g_1 is a generator of $\mathbb{G}$, and $2^\kappa > 2^\lambda \cdot |\mathbb{G}|$, then the scheme is statistically hiding for $m = 1$. This is the case in DARK.

- If $\mathbb{G}$ is the group $\mathbb{Z}_q^n$ for $q > p$ and all g_i are sampled independently from the uniform distribution over $\mathbb{Z}_q^n$ then the scheme is statistically hiding for $m = n \log_p q$ and $2^\kappa = p$ by the leftover hash lemma. This is known as Ajtai's hash function [Ajt96, GGH96] and is the commitment function for a PCS based on the Integer-SIS problem [BBC+18].

4.3 Compiler III: From Hiding to Zero Knowledge Eval

Lastly, we describe a generic compiler that takes any homomorphic hiding PCS with an Eval that is not zero-knowledge and transforms it into a PCS with a zero-knowledge Eval^*. The idea is similar to the sigma protocol for homomorphism pre-images and is a generalization of known techniques [CFS17, BCC+16, BFS20].

Let $\mathcal{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify}, \mathsf{Eval})$ denote the input PCS. By definition there are efficiently computable homomorphisms $\phi : \mathbb{Z}^D \to \mathbb{G}$ and $\chi : \mathbb{Z}^D \to \mathbb{F}^{(<d)}[X]$ 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 $f = \chi(\mathsf{open})$. The output of the compiler is a bounded witness zero-knowledge evaluation protocol for the max norm $||\cdot||_\infty : \mathbb{Z}^D \to \mathbb{Z}$. Let κ denote a security parameter of the compiler. Given the prover witness inputs (f, open_f) and public inputs (pp, C_f, z, y) for the claim that $f(z) = y \bmod p$, the protocol $\mathsf{ZKEval}(B)$ takes an additional parameter $B \in \mathbb{Z}$ and works as follows:

1. The prover chooses a random integer vector $\boldsymbol{\alpha} \xleftarrow{\$} [-2^\kappa \cdot B, B \cdot 2^\kappa]^D$, computes $\mathsf{C}_\alpha \leftarrow \phi(\boldsymbol{\alpha})$, which is a commitment to $\alpha \leftarrow \chi(\boldsymbol{\alpha}) \in \mathbb{F}^{(<d)}[X]$. The prover sends both $y_\alpha := \alpha(z)$ and C_α to the verifier.

2. The verifier sends a random challenge $c \xleftarrow{\$} [0, 2^\lambda)$ to the prover.

3. The prover derives $\mathbf{s} := \boldsymbol{\alpha} + c \cdot \mathsf{open}_f$. The prover and verifier each derive $\mathsf{C}_s := \mathsf{C}_\alpha + c \cdot \mathsf{C}_f$, which is a commitment to the polynomial $s := \alpha + c \cdot f \in \mathbb{F}^{(<d)}[X]$, and the prover sets $\mathsf{open}_s := \mathbf{s}$. They run the Eval protocol on public inputs $(\mathsf{C}_s, z, y_\alpha + c \cdot y)$, where the prover has private input (s, open_s), to open C_s at z to the value $s(z) = y_\alpha + c \cdot y$.

The transformed protocol is still public-coin and thus can be made non-interactive via Fiat-Shamir.

Lemma 3. *For any $B \in \mathbb{R}$ such that $B \geq p$, $\mathsf{ZKEval}(B)$ is an honest-verifier statistical zero-knowledge proof for the bounded witness PCS relation $\mathcal{R}_{\mathsf{Eval}}(pp, d, B, ||\cdot||_\infty)$ when $\kappa > 2\lambda + \log D$. If Eval is knowledge-sound then $\mathsf{ZKEval}(B)$ is also knowledge-sound.*

See §A.8.2 for the proof.

5 Batch Evaluation and Private Aggregation

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

Definition 8 (Aggregation)**.** *Let $\mathcal{PCS} = (\mathit{Setup}, \mathit{Commit}, \mathit{Verify}, \mathit{Eval})$ denote a PCS with commitment group $\mathbb{G}$. An aggregation scheme for $\mathcal{PCS}$ is a public-coin interactive protocol 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 $\mathit{Verify}(pp, f_i, \mathsf{open}_i, \mathsf{C}_i) = 1$ for all $i \in [\ell]$:*

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

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

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

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 $\mathsf{Verify}(pp, f^*, \mathsf{open}^*, C^*) = 1$.

Theorem 4. *Any PCS that has a linear combination scheme LinCombine (Definition 7) also has an aggregation scheme Aggregate (Definition 8) 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 1. *Every additive PCS (Definition 4) 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.*

Figure 2: A batch evaluation protocol for multiple commitments at multiple points based on a PCS aggregation scheme.

$\underline{\mathcal{P}(\mathbf{C}, \mathbf{z}, \mathbf{y}, \vec{\mathsf{open}}, \mathbf{f})}$	$\underline{\mathcal{V}(\mathbf{C}, \mathbf{z}, \mathbf{y})}$
$((\mathsf{open}^*, f^*), (C^*, z^*, y^*, b_1)) \leftarrow \mathsf{Aggregate}\big(\mathcal{P}(\mathbf{f}, \vec{\mathsf{open}}), \mathcal{V}(\mathbf{C}, \mathbf{z}, \mathbf{y})\big)$	
	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 $b_2 = 1$

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 1, every additive PCS, and more generally any PCS with an efficient linear combination scheme, has an efficient aggregation scheme.

Corollary 2. *If a PCS has an efficient linear combination scheme then it has an efficient aggregation scheme.*

The last corollary summarizes the communication and verification efficiency of the generic batch evaluation protocol for a PCS in terms of the complexity of its evaluation protocol and the amortization ratio of its linear combination scheme.

Corollary 3. *Suppose a PCS has a size-optimal linear combination scheme with amortization ratios $\epsilon, \delta \in [0, 1]$. Let $S_{\mathsf{Eval}}(\lambda, d)$, $C_{\mathsf{Eval}}(\lambda, d)$, and $V_{\mathsf{Eval}}(\lambda, d)$ respectively denote the worst case commitment size, Eval communication, and Eval verification complexities on commitments to polynomials of degree d. The PCS has a batch evaluation protocol for ℓ input commitments to polynomials in $\mathbb{F}^{(<d)}[X]$ with:*

- *Communication complexity at most $\epsilon \cdot (\ell + 2) \cdot C_{\mathsf{Eval}}(\lambda, d) + S_{\mathsf{Eval}}(\lambda, d) + 2\log(|\mathbb{F}|)$*
- *Verifier complexity at most $\delta \cdot (\ell + 2) \cdot V_{\mathsf{Eval}}(\lambda, d)$ plus an additional $O(\ell \log \ell)$ operations in $\mathbb{F}$*

5.1 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 3. The communication is comprised of one extra commitment and one evaluation protocol, independent of the number of input polynomials k. In Theorem 5 we show that the protocol is knowledge-sound.

The verifier is given $\big(\Omega_1,\ \mathsf{C}_{f_1}\big), \ldots, \big(\Omega_k,\ \mathsf{C}_{f_k}\big)$ where $\mathsf{C}_{f_i} \leftarrow \mathsf{Commit}(pp, f_i)$ and wants to be convinced that f_i is zero on Ω_i for all $i \in [k]$. First, let $\Omega \subseteq \mathbb{F}$ be the union of $\Omega_1, \ldots, \Omega_k$, and let $z_i(X)$ be the minimal degree polynomial that is zero on the set $\Omega \setminus \Omega_i$. Notice that $z_i f_i$ is zero on all of Ω if and only if f_i is zero on Ω_i. Define

$$f(X) := \sum_{i=1}^{k} \rho^{i-1} \cdot z_i(X) \cdot f_i(X) \quad \in \mathbb{F}[X]$$

where $\rho \xleftarrow{\$} \mathbb{F}$ is chosen at random by the verifier. If f is zero on Ω then we can conclude that f_i is zero on Ω_i for all $i \in [k]$ with overwhelming probability. To see why, suppose there exists $j \in [k]$ and $\omega \in \Omega_j$ such that $f_j(\omega) \neq 0$. Since $z_j(\omega) \neq 0$ by definition, the degree $k-1$ polynomial $g(X) := \sum_{i=1}^{k} z_i(\omega) f_i(\omega) X^{i-1}$ is not identically zero because at least the coefficient of X^{j-1} is non-zero. Then the probability $f(\omega) = g(\rho) = 0$ over $\rho \xleftarrow{\$} \mathbb{F}$ is at most $k/|\mathbb{F}|$.

Second, to verify that $f(\Omega) = 0$ we use a standard quotient proof. Let $z(X)$ be the minimal degree polynomial that is zero on Ω. Then $f(\Omega) = 0$ if and only if there is a polynomial $q \in \mathbb{F}^{(<d)}[X]$ such that $f(X) = q(X)z(X)$. The prover will commit to $q(X)$ and convince the verifier that $f(r) - q(r)z(r) = 0$ for a random $r \xleftarrow{\$} \mathbb{F}$ chosen by the verifier. One difficultly is that the verifier cannot derive the commitment to f on its own, because the $z_i(X)$ are polynomials rather than field elements. However the verifier can sample a random $r \xleftarrow{\$} \mathbb{F}$ and derive the commitment to the polynomial $\hat{f}(X) := \sum_{i=1}^{k} \rho^{i-1} \cdot z_i(r) \cdot f_i(X)$, which is sufficient.

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

Figure 3: 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).

$\mathcal{P}\big((f_1, \mathsf{open}_1, \Omega_1), \ldots, (f_k, \mathsf{open}_k, \Omega_k)\big)$		$\mathcal{V}\big((\mathsf{C}_1, \Omega_1),\ \ldots,\ (\mathsf{C}_k, \Omega_k)\big)$
$\Omega := \bigcup_{i=1}^{k} \Omega_i$		$\Omega := \bigcup_{i=1}^{k} \Omega_i$
$z(X) := \prod_{\omega \in \Omega}(X - \omega)$		$z(X) := \prod_{\omega \in \Omega}(X - \omega)$
$\forall_i\ \bar{\Omega}_i := \Omega \setminus \Omega_i$		$\forall_i\ \bar{\Omega}_i := \Omega \setminus \Omega_i$
$\forall_i\ z_i(X) := \prod_{\omega \in \bar{\Omega}_i}(X - \omega)$		$\forall_i\ z_i(X) := \prod_{\omega \in \bar{\Omega}_i}(X - \omega)$
	$\xleftarrow{\quad \rho \quad}$	$\rho \xleftarrow{\$} \mathbb{F}$
$q(X) := \sum_{i=1}^{k} \rho^{i-1} z_i f_i / z$		
$(\mathsf{C}_q, \mathsf{open}_q) \leftarrow \mathsf{Commit}(pp, q)$	$\xrightarrow{\quad \mathsf{C}_q \quad}$	
	$\xleftarrow{\quad r \quad}$	$r \xleftarrow{\$} \mathbb{F}$
$g(X) := \sum_{i=1}^{k} \rho^{i-1} z_i(r) f_i(X) - z(r) q(X)$		$\forall_i$ compute $z_i(r) \in \mathbb{F}$
(*if all is valid then* $g(r) = 0$)		
		$\mathsf{C}^* := \sum_{i=1}^{k} \rho^{i-1} z_i(r) \cdot \mathsf{C}_i$
		$\mathsf{C}_g := \mathsf{C}^* - z(r) \cdot \mathsf{C}_q$
	$\xleftrightarrow{\mathsf{Eval}\big(\mathcal{P}(g, \mathsf{open}_g), \mathcal{V}(\mathsf{C}_g, r, 0)\big)}$	

Theorem 5. *If Eval is knowledge sound, then the protocol in Figure 3 is a proof of knowledge for the relation:*

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

See §A.9 for the proof.

5.2 Batch evaluation protocol

The protocol for batch evaluation is a small generalization of the zero-testing protocol in Figure 3. 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 3 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 3 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 6. *If Eval is knowledge sound, then the batch evaluation protocol based on Figure 3 is a proof of knowledge for the relation* $\mathcal{R}_{\mathit{BatchEval}}(pp, d)$.

See §A.9 for the proof.

Moreover, 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 3 so we will not repeat the details.

5.3 Aggregation scheme (proof of Theorem 4)

When the PCS has a linear combination scheme (Definition 7), then the protocol from Section 5.2 together with the linear aggregation protocol 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:

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

1. Let $\Omega_i = \{x_i\}$ for $i \in [1, k]$, and let $t_i := y_i$.
2. Run the protocol in Section 5.2 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 $\mathcal{P}$ has privately derived $g(X)$, and $\mathcal{V}$ 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. If the PCS is additive, simply set the public output to $(\mathsf{C}_g, r, 0)$ and the prover's private output to (open_g, g).

4. If the PCS is not additive, then the prover and verifier will run $\mathsf{LinCombine}$ instead:
   - 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 $\vec{\mathsf{open}}' = (\mathsf{open}_1, ..., \mathsf{open}_k, \mathsf{open}^{(1)}, \mathsf{open}_q)$
   - 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, this composed protocol is a special case of the batch evaluation protocol in Section 5.2, which by Theorem 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 there is an extractor that obtains $(\mathsf{open}', \mathsf{open}^*, h)$ such that open^* is a valid opening of C^* to $g \in \mathbb{F}^{(<d)}[X]$ and open' is also a valid opening of C_g to the same $g \in \mathbb{F}^{(<d)}[X]$. This provides the extractor from Theorem 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 $\mathbb{G}$.

6 Homomorphic PCS Public Aggregation

The aggregation scheme in Definition 8 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 9 (Public Aggregation)**.** *Let $\mathcal{PCS} = (\mathit{Setup}, \mathit{Commit}, \mathit{Verify}, \mathit{NI\text{-}Eval})$ 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 $\boldsymbol{\pi} = (\pi_1, ..., \pi_\ell)$:*

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

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 $\mathit{Verify}(pp, f^*, \mathsf{open}^*, C^*) = 1$. The soundness requirement is that the following probability is negligible:

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

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 7. *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 7 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 $\mathbb{H} = \mathbb{Z}^n$ (described in detail in Section 4). Second, in this section 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.

6.1 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 $[\![\mathbf{x}]\!]_{\mathbf{g}} = [\![\mathbf{x}_L]\!]_{\mathbf{g}_L} + [\![x_R]\!]_{\mathbf{g}_R}$.

Figure 4: *A succinct interactive protocol for HPI.* For simplicity n is a power of 2.

$\underline{\mathcal{P}_{\mathsf{HPI}}(n, \mathbf{x} \in \mathbb{Z}^n, y \in \mathbb{G}, \mathbf{g} \in \mathbb{G}^n)}$		$\underline{\mathcal{V}_{\mathsf{HPI}}(n, y \in \mathbb{G}, \mathbf{g} \in \mathbb{G}^n)}$
	$\xrightarrow{\textbf{If } n = 1 \textbf{ send } \mathbf{x}}$	$[\![\mathbf{x}]\!]_{\mathbf{g}} \stackrel{?}{=} y$ **If yes accept, else reject**
	Else $n' \leftarrow \lceil \frac{n}{2} \rceil$	
$\mathbf{x} = (\mathbf{x}_L, \mathbf{x}_R); \mathbf{g} = (\mathbf{g}_L, \mathbf{g}_R)$ $y_L \leftarrow [\![\mathbf{x}_L]\!]_{\mathbf{g}_R}, y_R \leftarrow [\![\mathbf{x}_R]\!]_{\mathbf{g}_L}$		
	$\xrightarrow{y_L, y_R}$	$\alpha \xleftarrow{\$} [0, 2^\lambda)$
	$\xleftarrow{\alpha}$	
$\mathbf{x}' \leftarrow \mathbf{x}_L + \alpha \mathbf{x}_R$ $y' \leftarrow [\![\mathbf{x}']\!]_{\mathbf{g}_R + \alpha \mathbf{g}_L}$		
		$y' \leftarrow y_L + \alpha^2 y_R + \alpha y$ $\mathbf{g}' \leftarrow \mathbf{g}_R + \alpha \mathbf{g}_L$
$\mathcal{P}_{\mathsf{HPI}}(n', \mathbf{x}', y', \mathbf{g}')$	$\xleftrightarrow{\text{recurse}}$	$\mathcal{V}_{\mathsf{HPI}}(n', y', \mathbf{g}')$

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 4. The verifier's public-coin challenges are sampled uniformly from the set $\mathcal{X} := [0, 2^\lambda)$.

Connection to Bulletproofs The protocol is both a simplification and a generalization of the core protocol in Bulletproofs [BCC$^+$16, BBB$^+$18]. The Bulletproofs protocol is an *argument*[7] of knowledge for an inner-product relation between Pedersen vector commitments in a group of prime order $\mathbb{G}_p$. Provided with input commitments $\mathsf{C}_1, \mathsf{C}_2, \mathsf{C}_3 \in \mathbb{G}_p$, the prover demonstrates knowledge of $\mathbf{v}_1 \in \mathbb{G}_p^n$ and $\mathbf{v}_2 \in \mathbb{G}_p^n$ such that C_i is a Pedersen commitment to $\mathbf{v}_i$ for $i \in \{1, 2\}$ and C_3 is a commitment to $\langle \mathbf{v}_1, \mathbf{v}_2 \rangle$. In our case, the protocol simplifications come from the fact that the relation $\mathcal{R}^*_{\mathsf{HPI}}$ involves a linear function of the witness vector as opposed to quadratic. On the other hand, the protocol is also more general in the sense that ϕ can be a more general homomorphism than the Pedersen commitment function over a prime order group. As it turns out, the resulting protocol is also a *proof* of knowledge as opposed to an argument. Similar observations were made in concurrent work [AC20].

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:

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

[7] Arguments of knowledge require computational assumptions to prove knowledge extraction, whereas proofs of knowledge are information theoretically knowledge sound.

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 $(y', n', \mathbf{g}')$.

Proof communication size The proof size is two $\mathbb{G}$ elements per round for $\log n$ rounds, and then additionally a single integer $x' \in \mathbb{Z}$ sent in the final round. In the case that $p\mathbb{Z} \subseteq ker(\phi)$ then only the value $x' \bmod p$ needs to be communicated. More generally, if all coordinates of the witness $\mathbf{x}$ have absolute value at most 2^λ then x' is an integer of absolute value at most $2^{\lambda(\log n+1)}$. Letting Δ denote the size (in bit-length) of the final integer sent and $S_\mathbb{G} \in O(\log |G|)$ the representation size of group elements, then the total communication size is $2 \log n \cdot S_\mathbb{G} + \Delta$.

Prover complexity Suppose that the prover's witness $\mathbf{x} \in \mathbb{Z}^n$ has bounded norm $||\mathbf{x}||_\infty \leq B$. In the ith round of the protocol, for $i \in [1, \log n]$, the prover's work is dominated by computing two linear combinations over $\mathbb{G}$ each of length $n/2^i$ with integer coefficients of size at most $2^{(i-1)\lambda} \cdot B$. In total, these linear combinations naively cost at most $2(\lambda + \log B) \cdot n$ operations in $\mathbb{G}$. Fast algorithms for linear combinations over groups (e.g., Pippenger [Pip80]) may give up to a factor $\log n$ speedup. In the case that $\mathbb{G}$ has known order q, then the coefficients do not exceed q as they can first be reduced modulo q. In this case the total number of group operations is $O(\log q \cdot n)$.

Verifier complexity The verifier's work is $O(\lambda \cdot n)$ operations in $\mathbb{G}$ overall. The main cost is deriving the group vectors $\mathbf{g}' \leftarrow \mathbf{g}_R + \alpha \mathbf{g}_L$ for each round. As an optimization, since the $\mathbf{g}$ vectors are not used explicitly by the verifier until the last round where $d' = 1$, the verifier does not need to output the intermediate values of $\mathbf{g}'$ for rounds where $d' > 1$. It may derive the final $g' \in \mathbb{G}$ as a single linear combination of n group vectors in $\mathbb{G}$ with coefficients of size $O(\lambda \log n)$-bits from $\mathbb{Z}$. The verifier additionally computes a linear combination of 3 elements in $\mathbb{G}$ with scalars at most 2λ-bits per round to derive the final round $y' \in \mathbb{G}$ and check that $x' \cdot g' = y'$.

Batch verification Extending ideas from Bulletproofs [BBB+18] and Halo [BGH19], there is a way to amortize the cost of verifying proofs in a batch. The $\mathbf{g}$ vectors are not used explicitly by the verifier until the last round. Moreover, the final $g' = [\![\mathbf{u}]\!]_\mathbf{g} = \phi(\mathbf{u})$ where each $\mathbf{u} = (u_1, ..., u_n) \in \mathbb{Z}^n$ is defined as follows. Given challenges $\{\alpha_i\}$ for an execution of the protocol, for each $i \in [d]$ let $u_i = \prod_{j=1}^{\log n} v_{ij}$ where the value $v_{ij} = \alpha_j$ if the jth bit of i is 0 and $v_{ij} = 1$ if the jth bit of i is 1. Equivalently, $\mathbf{u}$ is the coefficient vector of the degree $n - 1$ polynomial $u(X) = \prod_{i=1}^{\log n}(\alpha_i + X^{2^{i-1}})$.

Suppose the verifier receives k proofs with final round pre-images $x'_1, .., x'_k$ and targets $y'_1, ..., y'_k$. Let $\mathbf{u}_i$ be defined by the challenges of the ith proof as described above. Rather than computing $g'_i \leftarrow \phi(\mathbf{u}_i)$ for each $i \in [k]$ and checking that $y'_i = x'_i \cdot g'_i$ individually, the verifier instead samples $r_1, ..., r_k \xleftarrow{\$} [0, 2^\lambda)$, computes $\mathbf{u}^* := \sum_{i=1}^k r_i \cdot x'_i \cdot \mathbf{u}_i$, $y^* = \sum_{i=1}^k y'_i$, and checks that $y^* = \phi(\mathbf{u}^*)$. While deriving $\mathbf{u}^*$ still requires $\Omega(kn)$ integer multiplications[8], the verifier evaluates ϕ only once. This is advantageous when evaluating ϕ is more expensive than computing linear combinations of vectors in $\mathbb{Z}^n$ (e.g., when operations in $\mathbb{G}$ are slower than integer multiplications). There are also algorithms to amortize the cost of large linear combinations of group elements (such as Pippenger's "multiexponentiation" algorithms [Pip80]), which may help for speeding up the evaluation of ϕ.

[8] Computing all $\mathbf{u}_i$ naively requires $O(n \log n)$ multiplications, but due to the overlapping structure of the vector components it is possible to derive all $\mathbf{u}_i$ with $O(n)$ multiplications overall using dynamic programming.

Moreover, when the verifier knows $|\mathbb{G}| = q$ then all the scalar multiplications can be taken over $\mathbb{Z}_q$, which is more efficient.

Theorem 8. *The protocol in Figure 4 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 using the tree-finding algorithm of Lemma 5 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 Lemma 5 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 $\mathbb{F}$. By Lemma 8, there is an efficient algorithm to compute a matrix $\mathbf{P}$ such that $\mathbf{P} \cdot \mathbf{T} \cdot \mathbf{A} = \mathbf{D}$ is a diagonal matrix with entries non-zero over $\mathbb{F}$. In particular, we obtain $d \neq 0 \bmod p$ such that $d \cdot y_w = \langle \mathbf{P}_1, \mathbf{X} \cdot \mathbf{g} \rangle$. The extractor sets $\mathbf{x}_w := \langle \mathbf{P}_1, \mathbf{X} \rangle$ and $t_w := d$, which now satisfies $[\![\mathbf{x}_w]\!]_{\mathbf{g}} = \langle \mathbf{x}_w, \mathbf{g} \rangle = t_w \cdot y_w$. $\square$

Zero knowledge The protocol in Figure 4 is not zero-knowledge. There is a simple transformation that compiles any interactive proof for $\mathcal{R}^*_{\mathsf{HPI}}$ into a zero-knowledge proof while preserving knowledge-soundness. Technically, the transformed protocol constrains the max norm of the prover's witness. For $\mathbf{x} \in \mathbb{Z}^n$ define $\mathcal{N}(x) := ||\mathbf{x}||_\infty$. The transformed protocol is an HVZK interactive proof for the modified relation:

$$\mathcal{R}_{\mathsf{Bounded\text{-}HPI}}(\phi, \mathbb{G}, B) = \{(\mathbf{x} \in \mathbb{Z}^n, y \in \mathbb{G}) : \phi(\mathbf{x}) = y \ \wedge \mathcal{N}(\mathbf{x}) < B\}$$

The transformation adds one extra round and increases communication by just $O(\lambda)$ bits.

1. The prover samples random $\mathbf{r} \xleftarrow{\$} [-2^{3\lambda-1}, 2^{3\lambda-1}]^n$ and sends $h := [\![\mathbf{r}]\!]_{\mathbf{g}}$ to the verifier.
2. The verifier samples a challenge $c \xleftarrow{\$} [0, 2^\lambda)$
3. The prover and verifier run the proof of knowledge protocol for $\mathcal{R}^*_{\mathsf{HPI}}$ where the prover's witness is $\mathbf{r} + c \cdot \mathbf{x}$ and the common input is $h + c \cdot y$.

Lemma 4. *The transformed protocol is an $n \cdot 2^{-\lambda}$-statistical HVZK interactive protocol for relation $\mathcal{R}_{\textit{Bounded-HPI}}(\phi, \mathbb{G}, 2^\lambda)$, and a proof of knowledge for relation $\mathcal{R}^*_{\textit{HPI}}(\phi, \mathbb{G}, p)$.*

See §A.10 for the proof.

HPI proof aggregation It is possible to aggregate k non-interactive HPI proofs (i.e., FS transform of Figure 4) for k HPI instances into a single HPI instance and aggregate proof, without knowing the witnesses for the k initial HPI statements. Verifying the aggregate proof convinces a verifier of the k initial proofs. It is knowledge-sound in the sense that there is an extractor that gets the states of both the initial provers and the aggregation prover and can extract witnesses for the k initial statements. Computing the aggregate proof costs $O(kn)$ work. The aggregate proof incurs only $O(\log n)$ extra communication and combined with the initial proofs requires only $O(k \log n + n)$ work to verify. The amortized verification time per proof is thus $O(\log n + n/k)$.

Verification of the HPI protocol in Figure 4 is dominated by the cost of deriving the final base element $g' \in \mathbb{G}$ as an integer linear combination of the input bases $\mathbf{g} \in \mathbb{G}^n$ in order to check $x' \cdot g' = y'$ (see discussion of batch verification above). The key observation behind the aggregation protocol is that the verifier does not actually need to compute g' as long as it is given a proof of knowledge that y' is *some* linear combination of $\mathbf{g}$. The protocol is presented in Figure 5.

Figure 5: *Public aggregation for HPI with amortized verifier efficiency.* The HPI instance is defined by $y \in \mathbb{G}$ and $\mathbf{g} \in \mathbb{G}^n$. The public inputs are proof transcripts $(\pi_1, ..., \pi_k)$ where π_i consists of $r = \log n$ prover messages $\{(y_L^{(ij)}, y_R^{(ij)})\}_{j=1}^r$ and the prover's final message $x'_i \in \mathbb{Z}$. The algorithm $(y', b) \leftarrow \mathcal{V}^*_{\mathsf{HPI}}(n, y, \pi)$ denotes a modification of the non-interactive verifier $\mathcal{V}_{\mathsf{HPI}}$ which only partially verifies the transcript π. It derives the FS simulated challenges $\{\alpha_{ij}\}_{j=1}^r$ for each round, checks the correctness of the prover messages $\{(y_L^{(ij)}, y_R^{(ij)})\}_{j=1}^r$, and derives the final round y'. It does not derive the final round g' nor check that $x' \cdot g' = y'$. It returns $(y', 1)$ if these checks pass.

$\mathcal{P}_{\mathsf{agg}}(\mathbf{y} \in \mathbb{G}^k, (\pi_1, ..., \pi_k), \mathbf{g})$		$\mathcal{V}_{\mathsf{agg}}(\mathbf{y} \in \mathbb{G}^k, (\pi_1, ..., \pi_k), \mathbf{g})$
$\forall_{i \in [k]}\ u_i(X) := \prod_{j=1}^{\log n}(\alpha_{ij} + X^{2^{j-1}})$		$\forall_{i \in [k]} (y'_i, b_i) \leftarrow \mathcal{V}^*_{\mathsf{HPI}}(n, y_i, \pi_i)$
$\mathbf{u}_i \in \mathbb{Z}^n$ coeff. of $u_i(X)$		If $\exists i\ b_i = 0$ **fail**
	$\xleftarrow{\quad \tau \quad}$	$\tau \xleftarrow{\$} [0, 2^\lambda)$
$\mathbf{u}^* := \sum_{i=1}^k \tau^{i-1} \cdot x_i \cdot \mathbf{u}_i$		$y^* := \sum_{i=1}^k \tau^{i-1} \cdot y'_i$
$\mathcal{P}_{\mathsf{HPI}}(n, \mathbf{u}^*, y^*)$	$\xleftrightarrow{\text{Protocol in Figure 4}}$	$\mathcal{V}_{\mathsf{HPI}}(n, y^*, \mathbf{g})$

The aggregation protocol in Figure 5 is also compatible with the zero-knowledge HPI protocol (Section 6.1). The zero-knowledge protocol reduces the HPI statement about a pre-image of $y \in \mathbb{G}$ to an HPI statement about a pre-image of some $y + c \cdot h \in \mathbb{G}$. The aggregation verifier must check the first round of the protocol to verify the reduction is correct, but otherwise the protocol in Figure 5 is used to aggregate the reduced statements.

Theorem 9. *For any $\phi : \mathbb{Z}^n \to \mathbb{G}$ given by $\phi(\mathbf{x}) = [\![\mathbf{x}]\!]_{\mathbf{g}}$, the composed protocol in which $(\mathcal{P}_{HPI}, \mathcal{V}_{HPI})$ run the protocol in Figure 4 on k instances to generate k (unchecked) transcripts and $(\mathcal{P}_{\mathsf{agg}}, \mathcal{V}_{\mathsf{agg}})$ run the protocol in Figure 5 on these transcripts, is a proof of knowledge for the relation:*

$$\mathcal{R}^*_{MultiHPI}(\phi, \mathbb{G}, p) = \left\{((\mathbf{X} \in \mathbb{Z}^{n \times k}, \mathbf{t} \in \mathbb{Z}^k), \mathbf{y} \in \mathbb{G}^k) : \forall_{i \in [k]}((t_i, \mathbf{X}_i), y_i) \in \mathcal{R}^*_{HPI}(\phi, \mathbb{G}, p)\right\}$$

Proof Sketch We only provide a sketch of this proof and leave the details to the reader. First, observe that in the analysis of Theorem 8 the extractor does not strictly need the labels (x_v, g_v) such that $x_v \cdot g_v = y_v$ at the leaves of the tree, which corresponds to the final round HPI instance for the normal HPI protocol in Figure 4. Rather, it simply uses these values as a way to derive a pre-image $\mathbf{x}_v$ of y_v such that $\phi(\mathbf{x}_v) = y_v$. In fact, the extractor succeeds assuming it has *any* labels $(t_v, \mathbf{x}_v, y_v)$ at the leaves of the tree such that $\phi(\mathbf{x}_v) = t_v \cdot y_v$. By the standard forking analysis, if $\mathcal{P}_{\mathsf{agg}}$ succeeds with non-negligible probability in the $\mathcal{P}_{\mathsf{HPI}}$ subroutine on HPI instance $y^* = \sum_{i=1}^k \tau^{i-1} y_i$, then assuming knowledge-soundness of this subroutine there is an extractor that obtains witnesses $(t_i, \mathbf{w}_i)$ for each y_i such that $\phi(\mathbf{w}_i) = t_i \cdot y_i$. (See §A.9 for an example of this extraction argument, which involves diagonalization of a Vandermonde matrix). These are fed to the extractor for $\mathcal{P}_{\mathsf{HPI}}$ instead of (x'_i, g'_i) for each $i \in [k]$, because the verifier $\mathcal{V}_{\mathsf{agg}}$ never checks that $x'_i \cdot g'_i = y'_i$. $\square$

Homomorphic PCS from collision-resistant homomorphism A collision-resistant homomorphism $h : \mathbb{Z}^d \to \mathbb{G}$ and a succinct proof of knowledge for the HPI problem can be used to construct a homomorphic PCS over $\mathbb{F}_p$. For $f \in \mathbb{F}^{(<d)}[X]$ with coefficient vector $\mathbf{f} \in \mathbb{Z}^d$, the commitment is $h(\mathbf{f})$. Let $\hat{\mathbb{G}} := \mathbb{G} \times \mathbb{F}$.. For any point $z \in \mathbb{F}$, define the homomorphism $\phi_z : \mathbb{Z}^d \to \hat{\mathbb{G}}$ such that $\phi_\alpha(\mathbf{f}) = (h(\mathbf{f}), f(z))$. The evaluation protocol opens C at z using the HPI proof of knowledge for the relation $\mathcal{R}^*_{\mathsf{HPI}}(\phi_z, \hat{\mathbb{G}}, p)$ on input $(\mathsf{C}, f(z))$ with witness $\mathbf{f}$. The HPI extractor obtains the "relaxed" opening $(t, \mathbf{x})$ such that $\phi_z(\mathbf{x}) = (t \cdot \mathsf{C}, t \cdot f(z))$ and $t \neq 0 \bmod p$.

6.2 Publicly aggregatable PCS (proof of Theorem 7)

The Halo [BGH19] protocol contains a public aggregation protocol for the Bulletproofs PCS. Inspired by this idea, we show how the HPI protocol of Figure 4 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 compiler described in Section 4.

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 8) 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 $\mathbf{g}$.

Compiler: Homomorphic to publicly aggregatable Let $\mathcal{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify}, \mathsf{Eval})$ denote the input homomorphic PCS. 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 2.2: 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:

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

1. The prover/verifier run a modification of the HPI protocol from Figure 4 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 out-

put $(x', (\mathsf{C}', y')) \in \mathbb{Z} \times \hat{\mathbb{G}}$ 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 and leave the details to the reader. The sketch is essentially the same as the proof sketch for Theorem 9. Recall that the extractor in the analysis of Theorem 8 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. The knowledge soundness of the resulting protocol is left as a conjecture:

Conjecture 1. *If Eval is FS compatible then protocol Eval^* is FS compatible.*

This conjecture is true in the random oracle model when n is a constant independent of the security parameter [GK96].

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 $\pi_i = (\pi^{(i)}_{\mathsf{HPI}}, x'_i, y'_i, \pi^{(i)}_{\mathsf{eval}})$:

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

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 5.3 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, the generic scheme from Section 5.3 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.

7 SNARKs and IVC from PCS Aggregation

Bünz et. al. [BCMS20] formally show how a concept they define called PCS accumulation schemes can be used to construct a PCD system, generalizing the Halo protocol [BGH19]. We show that a PCS public aggregation scheme satisfies the definition of a PCS accumulation scheme [BCMS20]. For a detailed and self-contained exposition of IVC/PCD for path distributed computation directly from PCS aggregation see §A.11.

PCS accumulation scheme We show that a public aggregation scheme for a PCS (Definition 9) satisfies the definition of an accumulation scheme for a non-interactive PCS from [BCMS20]. We first review the definition of an accumulation scheme. The definition has small syntactic differences from [BCMS20] due to syntactic differences in our PCS definition.

Definition 10 (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:*

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

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:

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

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, the algorithm $V(pp_{\mathsf{pc}}, \{X_i\}_{i=1}^k, \{acc_i\}_{i=1}^\ell, acc, \pi_{\mathsf{agg}})$ derives the tuples $(\boldsymbol{\pi}, \mathbf{C}, \mathbf{s}, \mathbf{t})$, parses $acc = (\mathsf{C}^*, s^*, t^*, \pi^*)$, and runs the aggregation verifier $\mathcal{V}_{\mathsf{Aggregate}}(pp_{\mathsf{pc}}, \boldsymbol{\pi}, \mathbf{C}, \mathbf{s}, \mathbf{t}, \mathsf{C}^*, s^*, t^*, \pi_{\mathsf{agg}})$.

Theorem 10. *Given any non-interactive PCS that has prover complexity $P_{\mathit{Eval}}(\lambda, d)$, verification complexity $V_{\mathit{Eval}}(\lambda, d)$, and size complexity $S_{\mathit{Eval}}(\lambda, d)$, and a non-interactive public aggregation scheme with prover complexity $P_{\mathit{agg}}(\lambda, d, \ell)$, aggregation proof size complexity $S_{\mathit{agg}}(\lambda, d, \ell)$, and verifier complexity $V_{\mathit{agg}}(\lambda, d, \ell) \in \mathit{poly}(\lambda, \ell, \log d)$ there is an efficient construction of a PCD system such that for any distributed computation with maximum in-degree ℓ and local predicate Φ of size $|\Phi| = N$:*

- *the prover runs in $O(P_{\mathit{Eval}}(\lambda, N) + P_{\mathit{agg}}(\lambda, d, \ell) + \lambda N \log N + \mathit{poly}(\lambda))$*
- *the verifier runs in $O(V_{\mathit{Eval}}(\lambda, N) + V_{\mathit{agg}}(\lambda, N, \ell))$*
- *the proof size is $O(S_{\mathit{agg}}(\lambda, d, \ell) + S_{\mathit{Eval}}(\lambda, d))$*

Theorem 10 is a consequence of the theorems in [BCMS20] combined with the observation that public aggregation gives an accumulation scheme as well generic compilers that construct SNARKs from any PCS. These compilers combine a PCS together with an information theoretic proof system known as a *Polynomial Interactive Oracle Proof* [BFS20] (or equivalently *Algebraic Holographic Proof* [CHM+19]) to produce a SNARK. Using recent Polynomial IOPs [CHM+19, GWC19], the compilation yields a SNARK with prover complexity $O(\lambda N \log N + P_{\mathsf{Eval}}(\lambda, N))$, verifier complexity $O(V_{\mathsf{Eval}}(\lambda, N))$, and size complexity $O(S_{\mathsf{Eval}}(\lambda, N))$.

For any PCS with $V_{\mathsf{Eval}}(\lambda, d) \in \mathsf{poly}(\lambda, \log d)$, it was already known how to construct a PCD with the efficiency characteristics stated. The asymptotics of Theorem 10 does not reflect how the concrete efficiency is dependent on $V_{\mathsf{agg}}(\lambda, d, \ell)$ and should result in a strict improvement on prior methods when $V_{\mathsf{agg}}(\lambda, d, \ell) \ll V_{\mathsf{Eval}}(\lambda, d)$.

Private accumulation A small tweak to Definition 10 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.

Moreover, we remark that Theorem 10 can be adapted for 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 [BCMS20] 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.

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

[AC20] Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 513–543. Springer, Heidelberg, August 2020.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, pages 99–108, 1996.

[Bab91] László Babai. Local expansion of vertex-transitive graphs and random generation in finite groups. In *23rd ACM STOC*, pages 164–174. ACM Press, May 1991.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[BBC+18] Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 669–699. Springer, Heidelberg, August 2018.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 701–732. Springer, Heidelberg, August 2019.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Heidelberg, May 2016.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013.

[BCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems: extended abstract. In Robert D. Kleinberg, editor, *ITCS 2013*, pages 401–414. ACM, January 2013.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. Cryptology ePrint Archive, Report 2020/499, 2020.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Heidelberg, October / November 2016.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 276–294. Springer, Heidelberg, August 2014.

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive, Report 2020/081, 2020. `https://eprint.iacr.org/2020/081`.

[BDLN16] Carsten Baum, Ivan Damgård, Kasper Green Larsen, and Michael Nielsen. How to prove knowledge of small secrets. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 478–498. Springer, Heidelberg, August 2016.

[BEG+91] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. In *32nd FOCS*, pages 90–99. IEEE Computer Society Press, October 1991.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Heidelberg, May 2020.

[BG93] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 390–420. Springer, Heidelberg, August 1993.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. `https://eprint.iacr.org/2019/1021`.

[BGKS19] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness. Cryptology ePrint Archive, Report 2019/336, 2019. `https://eprint.iacr.org/2019/336`.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017. `http://eprint.iacr.org/2017/1050`.

[BMRS20] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Report 2020/352, 2020. `https://eprint.iacr.org/2020/352`.

[BMV19] Benedikt Bünz, Mary Maller, and Noah Vesely. Efficient proofs for pairing-based languages. Cryptology ePrint Archive, Report 2019/1177, 2019. `https://eprint.iacr.org/2019/1177`.

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019.

[CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-Shamir and correlation intractability from strong KDM-secure encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 91–122. Springer, Heidelberg, April / May 2018.

[CFS17] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. A zero knowledge sumcheck and its applications. Cryptology ePrint Archive, Report 2017/305, 2017. `http://eprint.iacr.org/2017/305`.

[CHM+19] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. Cryptology ePrint Archive, Report 2019/1047, 2019. `https://eprint.iacr.org/2019/1047`.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020.

[CL20] Alessandro Chiesa and Siqi Liu. On the impossibility of probabilistic proofs in relativized worlds. In Thomas Vidick, editor, *ITCS 2020*, volume 151, pages 57:1–57:30. LIPIcs, January 2020.

[CLMQ20] Yilei Chen, Alex Lombardi, Fermi Ma, and Willy Quach. Does fiat-shamir require a cryptographic hash function? Cryptology ePrint Archive, Report 2020/915, 2020.

[Coo02] Gene Cooperman. Towards a practical, theoretically sound algorithm for random generation in finite groups, 2002.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 769–793. Springer, Heidelberg, May 2020.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In Andrew Chi-Chih Yao, editor, *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 310–331. Tsinghua University Press, 2010.

[Dix08] John Dixon. Generating random elements in finite groups. *The Electronic Journal of Combinatorics [electronic only]*, 15, 07 2008.

[Dra] J. Drake. *https://ethresear.ch/t/slonk-a-simple-universal-snark/6420*.

[FKL18] G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 33–62, 2018.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[Gab19] Ariel Gabizon. AuroraLight: Improved prover efficiency and SRS size in a sonic-like system. Cryptology ePrint Archive, Report 2019/601, 2019. `https://eprint.iacr.org/2019/601`.

[GGH96] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Collision-free hashing from lattice problems. *IACR Cryptology ePrint Archive*, 1996.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 9:169–192, 1996.

[GKM+18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Heidelberg, August 2018.

[GM17] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Heidelberg, August 2017.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

[Gro16a] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[Gro16b] Jens Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Report 2016/260, 2016. `http://eprint.iacr.org/2016/260`.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. `https://eprint.iacr.org/2019/953`.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[Hol19] Justin Holmgren. On round-by-round soundness and state restoration attacks. Cryptology ePrint Archive, Report 2019/1261, 2019. `https://eprint.iacr.org/2019/1261`.

[KPV19] Assimakis Kattis, Konstantin Panarin, and Alexander Vlasov. RedShift: Transparent SNARKs from list polynomial commitment IOPs. Cryptology ePrint Archive, Report 2019/1400, 2019. `https://eprint.iacr.org/2019/1400`.

[KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of Fiat-Shamir for proofs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 224–251. Springer, Heidelberg, August 2017.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

[Lab18] $O(1)$ Labs. Coda protocol, 2018. `https://codaprotocol.com/`.

[Lee20] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. Cryptology ePrint Archive, Report 2020/1274, 2020. `https://eprint.iacr.org/2020/1274`.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013.

[Pip80] Nicholas Pippenger. On the evaluation of powers and monomials. *SIAM Journal on Computing*, 9:230–250, 1980.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Heidelberg, August 2020.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg, March 2008.

[VP19] Alexander Vlasov and Konstantin Panarin. Transparent polynomial commitment scheme with polylogarithmic communication complexity. Cryptology ePrint Archive, Report 2019/1020, 2019. `https://eprint.iacr.org/2019/1020`.

A Appendices

A.1 Computational group

Definition 11. *A **computational group** is a finite group $\mathbb{G}$ whose elements are represented as bit strings of length $\mathsf{poly}(\log |\mathbb{G}|)$, where the identity element has a special string* id*, together with polynomial time algorithms* add*,* invert*, and* equal*:*

- $\mathsf{add}(g_1, g_2) \to g_3$ *takes as input the bit string representations of two group elements $g_1, g_2 \in \mathbb{G}$ and outputs the bit string representation of the element $g_1 + g_2$, or $\perp$ if either input is not a representation of a group element.*

- *$\mathsf{invert}(g) \to -g$ takes as input the bit string representation of an element $g \in \mathbb{G}$ and outputs the bit string representation of its inverse $-g \in \mathbb{G}$, or $\perp$ if the input is not a representation of a group element.*

- *$\mathsf{equal}(g_1, g_2)$ takes as input two bit strings, it outputs 1 if they are both valid representations of the same element in $\mathbb{G}$, and otherwise outputs 0.*

Our definition is not explicit about how group elements are sampled in the first place, other than the bit string id that is part of the definition of the group. Clearly, a computational group is only useful if there is a way to generate at least one initial group element other than the identity. However, this can be specified by the application. For example, our definition of polynomial commitments (Section 2.2) includes a commitment algorithm that outputs elements in a computational group.

A.2 Probability distributions

For a distribution $\mathcal{D}$, we write $x \xleftarrow{\$} \mathcal{D}$ to denote that x is sampled from $\mathcal{D}$; for a finite set S, we write $x \xleftarrow{\$} S$ to denote that x is sampled uniformly from S.

For two discrete random variables X and Y that take on values in the same set $\mathcal{U}$, we denote by $SD(X,Y)$ the **statistical distance** between X and Y, also known as the total variation distance defined as follows:

$$SD(X,Y) := max_{S \subseteq \mathcal{U}} \big| Pr[X \in S] - Pr[Y \in S] \big| = \frac{1}{2} \sum_{u \in \mathcal{U}} \big| Pr[X = u] - Pr[Y = u] \big|$$

Two random variables X and Y over $\mathcal{U}$ are **δ-close** if $SD(X,Y) \leq \delta$.

Fact 1. *If $\mathbf{X} = (X_1, ..., X_n)$ and $\mathbf{Y} = (Y_1, ..., Y_n)$ are each vectors of n independent discrete random variables, then $\mathbf{X}$ and $\mathbf{Y}$ are random variables such that $SD(\mathbf{X}, \mathbf{Y}) \leq \sum_{i=1}^{n} SD(X_i, Y_i)$.*

Fact 2. *If X is a random variable uniformly distributed over the interval $[-A, A]$ for $A \in \mathbb{Z}$ and $Y = X + z$ for a fixed $z \in \mathbb{Z}$ with bounded absolute value $|z| < B$, then $SD(X,Y) < \frac{B}{2A}$.*

A.3 Interactive proofs of knowledge

Definition 12 (Interactive Proof with Efficient[9] 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.*

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

[9] 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.

- Knowledge soundness [BG93] *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.* [BCC+16].

Lemma 5 (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 $\forall_i$ $Pr[\pi(x_1, x_2) = 1 : x_1, x_2 \xleftarrow{\$} \mathcal{X}] > 1 - \mathsf{negl}(\lambda)$. 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 $\pi_i(x_1, x_2) = 1$.*

The forking lemma is used to prove knowledge soundness of $(\mathcal{P}, \mathcal{V})$ in combination with a deterministic extraction algorithm that outputs a witness given an $(n_1, ..., n_r)$-tree of accepting transcripts satisfying properties $\pi_1, \ldots, \pi_r$. The proof of our Lemma 5 is nearly identical to the proof in [BCC+16]. The tree-finding algorithm in [BCC+16] is an adaptive rejection sampling algorithm which samples at most $\mathsf{poly}(\lambda)/\epsilon$ challenges overall from the uniform distribution over $\mathcal{X}$, and outputs a subset of these challenges. They show that all sibling challenges are unique with overwhelming probability by a union bound over the probability of a collision between any pair of challenges sampled by the algorithm. This union bound argument can be extended to any property of challenge pairs that holds with overwhelming probability $1 - \mathsf{negl}(\lambda)$ for a randomly sampled pair.

Lemma 6 provides another helpful fact about partial transcripts in the tree returned by the algorithm of Lemma 5. If tr is a partial transcript of the protocol interaction between $\mathcal{A}$ and a verifier on the first i round challenges $(x_1, ..., x_i)$ and st is the internal state of $\mathcal{A}$ after generating tr, then "running $\mathcal{A}$ on partial transcript tr" means that the internal state of $\mathcal{A}$ is restored to st, and the protocol is continued on uniform random challenges for the remaining rounds. $\mathcal{A}$ "succeeds" on tr if it causes the verifier to accept when it is run on tr.

Lemma 6. *Fix any $\delta \in [0, 1]$. With probability at least $1 - \frac{\delta}{\epsilon} \cdot \mathsf{poly}(\lambda)$, every partial transcript tr in the transcript tree output by the algorithm $\mathcal{T}$ in Lemma 5 with adversary $\mathcal{A}$ has the property that $\mathcal{A}$ succeeds on tr with probability at least δ.*

This lemma holds because the transcripts that appear in the tree are the result of rejection sampling. In the course of the tree finding algorithm, at most $\mathsf{poly}(\lambda)/\epsilon$ partial transcripts are "tested" and the probability a given partial transcript is *not rejected* is bounded by the probability $\mathcal{A}$ succeeds on it. By a union bound, there is a probability at most $\mathsf{poly}(\lambda) \cdot \delta/\epsilon$ that the output tree contains a partial transcript that $\mathcal{A}$ succeeds on it with probability less than δ.

Definition 13 (HVZK for interactive proofs). *Let $\mathsf{View}_{\langle \mathcal{P}(x,w), \mathcal{V}(x)\rangle}$ denote the view of the verifier in an interactive protocol described in Definition 12 on common input x and prover witness input w. It is a random variable over the randomness of $\mathcal{P}$ and $\mathcal{V}$. The interactive protocol has δ-statistical honest verifier zero-knowledge (HVZK) if there exists a probabilistic polynomial time algorithm $\mathcal{S}$ such that for every $(x, w) \in \mathcal{R}$, the random variable $\mathcal{S}(x)$ is δ-close to the random variable $\mathsf{View}_{\langle \mathcal{P}(x,w), \mathcal{V}(x)\rangle}$. The protocol has perfect HVZK when $\delta = 0$.*

A.4 Hash functions

Definition 14 (Collision-resistant hashing). *A hash function family $\mathcal{H} = \{H_\lambda\}_{\lambda \in \mathbb{N}}$ is a collection of functions such that $H_\lambda : \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \mathcal{T}_\lambda$ where $|\mathcal{T}_\lambda| < |\mathcal{X}_\lambda|$. Let $\mathcal{A}$ denote an algorithm that takes inputs $k \in \mathcal{K}_\lambda$ and let $\mathsf{CR}[\mathcal{A}, \mathcal{H}](\lambda)$ denote the probability over $k \xleftarrow{\$} \mathcal{K}_\lambda$ that $\mathcal{A}$ outputs a pair $(x, y) \in \mathcal{X}_\lambda^2$ such that $H_\lambda(k, x) = H_\lambda(k, y)$ and $x \neq y$. $\mathcal{H}$ is a **collision-resistant hash function** (CRHF) family if $\mathsf{CR}[\mathcal{A}, \mathcal{H}](\lambda)$ is a negligible function of λ for all $\mathcal{A}$ with runtime polynomial in λ.*

The above definition is asymptotic. As a more concrete way to define collision-resistance, we may say that $H : \mathcal{K} \times \mathcal{X} \to \mathcal{T}$ is a "λ-bit secure CRHF" if $|\mathcal{T}| < |\mathcal{X}|$ and there is no probabilistic algorithm $\mathcal{A}$ that runs on input $k \xleftarrow{\$} \mathcal{K}$ and returns a collision $x \neq y$ such that $h(x) = h(y)$ in *expected time* 2^λ steps. The probability is over the randomness of h and internal randomness of the algorithm. This precludes attacks that always find a collision in less than 2^λ steps, or that find a collision with probability ϵ in less than $\epsilon \cdot 2^\lambda$ steps.

Whenever we informally refer to a hash function $h : \mathcal{X} \to \mathcal{T}$ as collision-resistant, it is understood that h is sampled from a family of *keyed hash functions*, i.e. $h := H_\lambda(k, \cdot)$ for $k \xleftarrow{\$} \mathcal{K}$ and a chosen security level $\lambda \in \mathbb{N}$, where either $\{H_\lambda\}_{\lambda \in \mathbb{N}}$ is a CRHF family as in Definition 14 or a concrete λ-bit secure CRHF as defined above.

Definition 15 (Universal hashing). *A family of keyed hash functions $H : \mathcal{K} \times \mathcal{X} \to \mathcal{T}$ is **2-universal** if for all $(x, y) \in \mathcal{X}^2$ such that $x \neq y$, the probability over $k \xleftarrow{\$} \mathcal{K}$ that $H(k, x) = H(k, y)$ is $\frac{1}{|\mathcal{T}|}$.*

The classical Leftover Hash Lemma [HILL99] expresses how a 2-universal hash family can be used as randomness extractors to obtain an element distributed close to uniform over $\mathcal{T}$ from any non-uniform random variable X over $\mathcal{X}$ that has more than $\log |\mathcal{T}|$ bits of min-entropy. The **min-entropy** of X is defined as $\mathbb{H}_\infty(X) := -\log\ max_{x \in \mathcal{X}} Pr[X = x]$.

Lemma 7 (Leftover Hash Lemma). *If $H : \mathcal{K} \times \mathcal{X} \to \mathcal{T}$ is a 2-universal hash family, X is a random variable over $\mathcal{X}$, $Y := H(k, X)$ is the random variable over $\mathcal{T}$ for $k \xleftarrow{\$} \mathcal{K}$, and $U_\mathcal{T}$ is the uniform distribution over $\mathcal{T}$, then $SD((Y, k), (U_\mathcal{T}, k)) \leq \frac{1}{2} \cdot \sqrt{\frac{|\mathcal{T}|}{2^{\mathbb{H}_\infty(X)}}}$.*

A.5 Module equations for PCS

The following lemmas are useful for knowledge soundness analysis (i.e., extraction) for protocols involving a PCS. Let $\mathbb{G}$ denote an abelian group We first prove an elementary linear algebraic fact.

Lemma 8. *Let $\mathbb{G}$ be a $\mathbb{Z}$-module. Let p be a prime and $\mathbb{F} = \mathbb{Z}_p$. Given two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{G}^n$ and a system of equations $\mathbf{A}\mathbf{x} = \mathbf{y}$ for a matrix $\mathbf{A} \in \mathbb{Z}^{n \times n}$ that is invertible over $\mathbb{F}$, there is an efficient algorithm to derive a diagonal integer matrix $\mathbf{D}$ with diagonal entries all non-zero modulo p and a matrix $\mathbf{L}$ such that $\mathbf{D} \cdot \mathbf{x} = \mathbf{L} \cdot \mathbf{y}$. In particular, $\mathbf{L}\mathbf{A} = \mathbf{D}$.*

Proof. Since $det(\mathbf{A}) \neq 0$, the matrix $\mathbf{A}$ is invertible over the rationals $\mathbb{Q}$. 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. $\mathbf{L} = x \cdot \mathbf{A}^{-1}$ where $x \neq 0$ is the least common multiple of all denominators of the rational entries of $\mathbf{A}^{-1}$. The matrix $\mathbf{L} \cdot \mathbf{A} = x \cdot \mathbf{A}^{-1} \cdot \mathbf{A} = x \cdot \mathbf{I}$ is a diagonal integer matrix. □

The next lemma is a direct result of this fact. Suppose that $\mathbb{G}$ is the abelian group for a PCS.

Lemma 9. *Given two vectors of commitments $\mathbf{C}, \mathbf{C}^* \in \mathbb{G}^n$, a system of equations $\mathbf{A}\mathbf{C} = \mathbf{C}^*$ 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 $\mathbf{f} = (f_1, ..., f_n) \in (\mathbb{F}^{(<d)}[X])^n$, integer vector $\mathbf{t} \in \mathbb{Z}^n$ such that $t_i \neq 0 \bmod p$, and openings for each $t_i \cdot \mathbf{C}_i$ to the polynomial $t_i \cdot f_i \bmod p$ such that $\mathbf{A} \cdot \mathbf{f} \equiv \mathbf{f}^* \pmod{p}$.*

Proof. By Lemma 8, there exists a diagonal matrix $\mathbf{T}$ with integer entries $t_1, ..., t_n \neq 0 \bmod p$ and a matrix $\mathbf{L}$ such that $\mathbf{T} \cdot \mathbf{C} = \mathbf{L} \cdot \mathbf{C}^*$ and $\mathbf{L} \cdot \mathbf{A} = \mathbf{T}$. From each linear combination of $\mathbf{C}^*$, we use add^* to derive an opening of $t_i \cdot \mathbf{C}_i$ 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 $\mathbf{L} \cdot \mathbf{A} \cdot \mathbf{f} = \mathbf{T} \cdot \mathbf{f} = \mathbf{L} \cdot \mathbf{f}^*$ where $\mathbf{T}$ is a diagonal matrix with entries $t_i \neq 0 \bmod p$. Thus, $t_i f_i = g_i$, for which we have a valid opening of $t_i \cdot \mathbf{C}_i$. □

A.6 Additive PCS examples

Bulletproofs The polynomial commitment is a Pedersen hash function over a prime order group $\mathbb{G}_p$. The setup parameters includes d randomly sampled generators $g_0, ..., g_{d-1}$. In additive group notation, the commitment to $f \in \mathbb{F}$ with coefficient vector $(f_0, ..., f_{d-1})$ is $C_f := \sum_{i=0}^{d-1} f_{i-1} \cdot g_i$. There is no special opening string. The commitment function is a group homomorphism from $\mathbb{F}^d \to \mathbb{G}_p$. The evaluation protocol is based on the inner-product argument of Bootle et. al. [BCC+16], improved upon by Bünz et. al. [BBB+18]. The PCS evaluation requires opening a linear form, and therefore is slightly simpler than the original version for inner-products (e.g., see [AC20] or our homomorphism pre-image protocol in Section 6). The communication complexity of Eval is $O(\log d)$ group elements and the verification complexity is $O(d)$ group operations. Neither communication nor verification complexity increase when applied to a linear combination of two commitments.

KZG The KZG [KZG10] polynomial commitment uses a triple of groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t)$ that have an efficiently computable non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$. The groups have the same prime order p as the field $\mathbb{F}$ over which the polynomials are defined. A trusted setup generates additional public parameters $(g, g_1, ..., g_{d-1}, h, h_1)$ where $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$ are generators, $s \in \mathbb{F}$ is sampled uniformly, $g_i = s^i \cdot g$ and $h_1 = s \cdot h$. The value of s must remain secret. Similar to the Bulletproof PCS, a commitment to a polynomial $f \in \mathbb{F}$ with coefficient vector $(f_0, ..., f_{d-1})$ is $C_f := \sum_{i=0}^{d-1} f_{i-1} \cdot g_i$. Note that $C_f = g^{f(s)}$. There is no special opening string. The commitment function is a group homomorphism from $\mathbb{F}^d \to \mathbb{G}_1$. To prove that $f(z) = y$, the prover simply outputs a commitment C_q to the quotient polynomial $q := \frac{f-y}{X-z}$. A correctly generated $C_q = g^{q(s)}$ will satisfy $e(C_q, h_1 \cdot h^{-z}) = e(g^{q(s)}, h^{s-z}) = e(g^{f(s)}, h)$. The proof is accepted by the verifier if and only if $e(C_f, h) = e(C_q, h_1 \cdot h^{-z})$. Neither communication nor verification complexity increase when applied to a linear combination of two commitments.

DARK The DARK [BFS20] polynomial commitment uses a cyclic group of unknown order $\mathbb{G}$ with a generator g. If $\mathbb{G}$ can be instantiated without trusted setup (e.g., the class group of a quadratic number field), then the scheme does not require trusted setup. To support commitments to $\mathbb{F}^{(<d)}[X]$, an integer q of size $O(\log d \cdot \log p)$ bits is fixed. A commitment to the integer coefficient vector $(f_0, ..., f_{d-1}) \in [0, p)^d$ is $C_f := \sum_{i=0}^{d-1} f_{i-1} \cdot q^{i-1} \cdot g$. Equivalently, $C_f = f(q) \cdot g$ where $f(q)$ is evaluated over $\mathbb{Z}$. This commitment function is a homomorphism from $\mathbb{Z}^d \to \mathbb{G}$. However, since the commitment is only binding as long as $q > p/2$, the scheme only supports a bounded number of homomorphic additions of commitments. The evaluation proof is a recursive protocol that has the same flavor as Bulletproofs, but additionally requires the verifier to check an integer bound on the final message sent by the prover. The extracted witness are the integer coefficients of a polynomial f^* such that $C_f = f^*(q) \cdot g$. Due to technicalities of the analysis, the extracted coefficients may have size $p^{O(\log d)}$. This is the reason why q must be substantially larger than p.

A.7 FRI

Reed-Solomon codes over $\mathbb{F}$ are parametrized by a rate parameter ρ and subset $D \subseteq \mathbb{F}$ of size n and are defined as the set $RS[D, \rho] := \{f(D) : f \in \mathbb{F}^{(<\rho n)}[X]\}$. The notation $f(D)$ denotes the vector of evaluations of f on all points in D in some canonical order. Two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}^n$ are considered δ-close if their relative Hamming distance, denoted $\Delta(\mathbf{u}, \mathbf{v})$, is at most δ. (The relative Hamming distance between vectors in $\mathbb{F}^n$ is defined as the number of components in which $\mathbf{u}$ and $\mathbf{v}$ are different divided by n). For any $\mathbf{u} \in \mathbb{F}^n$ and $\delta \in [0, 1]$, let $B_\delta(\mathbf{u})$ denote the ball of vectors that are δ-close to $\mathbf{u}$. If $\mathbf{u} \in RS[D, \rho]$, then the *unique decoding radius* of $\mathbf{u}$ is $\delta_0 := \frac{1-\rho}{2}$. This is due to the fact that the evaluations of any two distinct polynomials of degree less than ρn agree in less than ρn points of D. Equivalently, if $\mathbf{u}, \mathbf{v} \in RS[D, \rho]$ are distinct codewords then $\Delta(\mathbf{u}, \mathbf{v}) > 1 - \rho$. Thus, by the triangle inequality no $\mathbf{w}$ can simultaneously have distance less than or equal to $\frac{1-\rho}{2}$ to both $\mathbf{u}$ and $\mathbf{v}$. Given $\mathbf{w} \in B_{\delta_0}(\mathbf{u})$ for $u \in RS[D, \rho]$ with $\delta = \delta_0$, the Berlekamp-Welch algorithm can be used to recover $\mathbf{u}$ in time $O(n^3)$. More generally, for $\delta < 1 - \sqrt{\rho}$, the Guruswami-Sudan algorithm may be used to recover from $\mathbf{w} \in \mathbb{F}^n$ all $\mathbf{u} \in RS[D, \rho]$ such that $\Delta(\mathbf{w}, \mathbf{u}) \leq \delta$ in time $O(n^3)$.

Setting $d = \lceil \rho n \rceil$, the FRI protocol requires only $O(\log d)$ oracle queries to locations of a vector in order to prove with overwhelming probability that the vector is in $B_\delta(\mathbf{u})$ for some $\mathbf{u} \in RS[D, \rho]$. The value of δ affects the concrete efficiency (a smaller δ requires more queries to maintain the same error probability). Equating vectors in $\mathbb{F}^n$ with polynomials $\mathbb{F}^{(<n)}[X]$, given an oracle for $f \in \mathbb{F}^{(<n)}[X]$ the FRI protocol with δ set to the unique decoding radius δ_0 proves that f has degree at most d. The FRI protocol can also be applied to rational functions implicitly defined by several oracles. For example, if the verifier has oracle access to $f, g, h \in \mathbb{F}^{(<n)}[X]$ then FRI can be used to prove that $\frac{f+g}{h}$ has degree at most d.

FRI can be used as the evaluation protocol for a polynomial commitment scheme [VP19, KPV19, BGKS19]. The commitment C_f to $f \in \mathbb{F}^{(<d)}[X]$ is a vector commitment to the codeword $f(D) \in RS[D, \rho]$. An opening is simply an opening of the vector commitment. To open an evaluation of

C_f to $f(z) = y$ for $z \notin D$, the prover runs FRI for the codewords $f(D)$ and $q(D)$ for $q := \frac{f-y}{X-z}$, simulating oracle access to $f(D)$ and $q(D)$ by opening locations of the vector commitment to $f(D)$. In fact, it suffices to run FRI on a random linear combination of $f(D)$ and $q(D)$. For $z \in D$ the prover runs FRI just on $f(D)$ and opens the appropriate location of the commitment to $f(D)$.

FRI batch evaluation FRI can be applied directly to a linear combination codeword commitments (i.e., virtual codeword commitment) with additive as opposed to multiplicative overhead (see Protocol 8.2 of Aurora [BCR+19]). FRI involves $2 \log d$ codewords in addition to the input codeword and κ queries to each codeword, where κ is a statistical security parameter. The total number of queries is $2\kappa \log d + \kappa$, however only κ of these queries are made to the input codeword. Thus, the number of queries in FRI applied to a formal linear combination of ℓ committed codewords of equal rate is only $\kappa \cdot (\ell - 1)$ larger than applying FRI to a single codeword. As a result, the communication/verification complexity of FRI Eval on a linear combination of ℓ equal rate commitments is less than a factor $1 + \frac{\ell}{2 \log d}$ larger than on a single commitment. Moreover, to simultaneously demonstrate proximity of multiple committed vectors to RS codewords it suffices to run FRI on a random linear combination of the commitments (see Section 8 of Aurora [BCR+19]). This means that ℓ commitments may be opened at ℓ distinct points with complexity proportional to a single Eval.

FRI linear combination scheme On input commitments $\mathsf{C}_1, ..., \mathsf{C}_\ell$ to RS codewords $f_i(\mathcal{D})$ for $i \in [\ell]$, the $\mathsf{LinCombine}$ prover computes a fresh FRI PCS commitment C_f to the linearly combined polynomial $f := \sum_i \alpha_i f_i$ (i.e., a vector commitment to the codeword $f(\mathcal{D})$). It also runs the first round of FRI on the virtual commitment $\mathsf{C} := \mathsf{C}_f - \sum_{i=1}^{\ell} \alpha_i \cdot \mathsf{C}_i$ (i.e., formal linear combination) to show that:

1. C is a commitment to the zero polynomial
2. C_i is a valid polynomial commitment (i.e., close to an RS codeword) for all $i \in [\ell]$

There are various methods to do this. For example, to show that C is a commitment to the zero polynomial it suffices to run the FRI Eval protocol to open an evaluation of C at a random point to the value zero. To show that all C_i are valid commitments for $i \in [\ell]$ it suffices to run the FRI proximity proof on a random linear combination of C_i. Moreover, these protocols can be batched into a single FRI protocol on virtual codewords derived from C and $\{\mathsf{C}_i\}_{i \in [\ell]}$. The prover's first round message in this protocol contains a new commitment $\mathsf{C}_{f'}$ to a codeword for a degree $d/2$ polynomial f' and also openings for queries to C and $\mathsf{C}_{f'}$. The verifier checks the first round openings. The remaining FRI protocol rounds (omitted from the aggregation proof) would constitute an RS proximity proof for $\mathsf{C}_{f'}$.

The aggregate commitment will be the pair $(\mathsf{C}_f, \mathsf{C}_{f'})$. Technically, to conform to the PCS syntax, we augment $\mathbb{G}$ to allow for pairs of commitments $(\mathsf{C}_f, \mathsf{C}_{f'})$ and Eval at a point z uses the batched FRI protocol to simultaneously open $f(z)$ and prove RS proximity of $\mathsf{C}_{f'}$. Thus, running Eval on the aggregation output simultaneously opens an evaluation of f and completes the proof that C is a valid commitment to the zero polynomial and that each C_i was a valid commitment for $i \in [\ell]$. In total, this shows that C_f is a commitment to $f := \sum_i \alpha_i f_i$ where C_i is a commitment to f_i for $i \in [\ell]$ and also opens $f(z)$.

This linear combination scheme has an amortization ratio of $\frac{1}{\ell} + \frac{1}{\Omega(\log d)}$. The verifier complexity is dominated by running the first round of FRI on the virtual commitment C, and is thus a factor $\Omega(\log d)$ smaller than the verifier complexity of Eval.

A.8 Hiding/ZK PCS Compiler

A.8.1 Proof of Lemma 2

We recall the statement of the lemma:

> The PCS returned by the compiler in Figure 1 is binding, knowledge-sound, and statistically hiding for $m\kappa \geq \lambda + \log|\mathbb{G}|$.

Binding If Verify^* accepts distinct openings $\mathsf{open}_1 = (\mathbf{r}_1, \mathbf{f}_1)$ and $\mathsf{open}_2 = (\mathbf{r}_2, \mathbf{f}_2)$ of C to distinct $f_1 \neq f_2$ then $\mathbf{f}_1 \not\equiv \mathbf{f}_2 \bmod p$ yet $\phi((\mathbf{A} \cdot \mathbf{r}_1, \mathbf{f}_1)) = \phi(\mathbf{A} \cdot \mathbf{r}_2, \mathbf{f}_2))$. This contradicts the binding property of the input scheme.

Knowledge soundness An adversary that succeeds with non-negligible probability in the evaluation protocol succeeds with non-negligible probability in the three Eval subprotocols. Hence, the extractor can run the Eval extractors to obtain openings (f, open_f) of C_f, (f', open') of $\mathsf{C}' = \mathsf{C} - \mathsf{C}_r$ such that $f(z) = y$, $f'(\rho) = \rho^m \cdot f(\rho)$, $deg(f) < D$, and $deg(f') < D$. Moreover, there exists at least $D + m$ distinct $\rho_1, ..., \rho_{D+m}$ such that the adversary would succeed in all three Eval subprotocols with non-negligible probability on each of these evaluation points. By evaluation binding of the PCS, this implies that $f'(\rho_i) = \rho_i^m \cdot f(\rho_i)$ for each $i \in [1, D+m]$. Since $deg(f' - X^m \cdot f) < D + m$ this implies $f' = X^m \cdot f$ and $deg(f) < d$. Finally, using the additive property of the PCS the extractor can derive from $\mathbf{r}$ and open' an opening of C to the polynomial with coefficient vector $(\mathbf{r}, \mathbf{f})$.

Hiding Let $q = |\mathbb{G}|$. Let $\phi_m : \mathbb{Z}^m \to \mathbb{G}$ be defined as $\phi_m(\mathbf{x}) = \sum_{i=1}^m x_i \cdot g_i$. It suffices prove that $\mathsf{C}_r = \phi(\mathbf{A} \cdot \mathbf{r})$ for $\mathbf{r} \xleftarrow{\$} [0, 2^\kappa)^m$ has negligible distance to uniform over $\mathbb{G}$. The min-entropy of $\mathbf{r}$ is $m\kappa$. Due to the fact that the PCS is m-spanning ϕ_m is surjective. We will show that $\phi_m(\mathbf{A} \cdot \mathbf{r})$ is a good randomness extractor using the Leftover Hash Lemma.

Define the keyed hash family $\mathcal{H} : \mathbb{Z}_q^{m \times m} \times \mathbb{Z}^m \to \mathbb{G}$ such that $\mathcal{H}(\mathbf{A}, \mathbf{x}) = \phi_m(\mathbf{A} \cdot \mathbf{x})$. For uniform $\mathbf{y} \in \mathbb{Z}_q^m$ the element $\phi_m(\mathbf{y})$ is uniformly distributed over $\mathbb{G}$. Moreover, for $\mathbf{A}$ sampled uniformly and $\mathbf{x} \neq 0$ the vector $\mathbf{y} = \mathbf{A} \cdot \mathbf{x} \bmod q$ is uniformly distributed over $\mathbb{Z}_q^m$. Thus, $\mathcal{H}$ is 2-universal (Definition 15) because for any $\mathbf{x} \neq \mathbf{y}$ and uniformly distributed $\mathbf{A}$ the element $\mathcal{H}(\mathbf{A}, \mathbf{x}) - \mathcal{H}(\mathbf{A}, \mathbf{y})$ is uniformly distributed in $\mathbb{G}$ and the probability of a collision is $1/|\mathbb{G}|$. Let $U_\mathbb{G}$ denote an independent random variable from the uniform distribution over $\mathbb{G}$. Since $H_\infty(\mathbf{r}) \geq m\kappa$ and $\mathbf{A}$ is *independently* uniform over $\mathbb{G}$, by the Leftover Hash Lemma (Lemma 7):

$$SD((\mathcal{H}(\mathbf{A}, \mathbf{r}), \mathbf{A}), (U_\mathbb{G}, \mathbf{A})) \leq \frac{1}{2}\sqrt{2^{\log q - m\kappa}}$$

Thus, for $m\kappa \geq \lambda + \log q$, the distribution of C_r has negligible distance at most $2^{-\lambda}$ from uniform over $\mathbb{G}$.

A.8.2 Proof of Lemma 3 (zk compiler)

We recall the statement of the lemma:

> For any $B \in \mathbb{R}$ such that $B \geq p$, $\mathsf{ZKEval}(B)$ is an honest-verifier statistical zero-knowledge proof for the bounded witness PCS relation $\mathcal{R}_{\mathsf{Eval}}(pp, d, B, ||\cdot||_\infty)$ when $\kappa > 2\lambda + \log D$. If Eval is knowledge-sound then $\mathsf{ZKEval}(B)$ is also knowledge-sound.

The relation $\mathcal{R}_{\mathsf{Eval}}(pp, d, B, ||\cdot||_\infty)$, defined in Section 3, contains all commitment/witness pairs $(\mathsf{C}, \mathsf{open}) \in \mathcal{R}_{\mathsf{Eval}}(pp, d)$ such that $||\mathsf{open}||_\infty \leq B$, where $||\cdot||_\infty : \mathbb{R}^D \to \mathbb{R}$ is the standard infinity norm. This is well defined for the PCS returned by the compiler because $\mathsf{open} \in \mathbb{Z}^D$.

Let $|\mathbb{G}| = q$ (not necessarily known to the simulator). The simulator samples $c^* \xleftarrow{\$} [0, 2^\lambda)$ and $\mathbf{t} \xleftarrow{\$} [-2^\kappa \cdot B, 2^\kappa \cdot B]^D$. It sets $y_\beta := \beta(z)$ where $\beta = \chi(\boldsymbol{\beta})$ and defines $\mathbf{f}^* \in [0, p)^D$ such that $\chi(\mathbf{f}^*) = y$ is the constant polynomial $y \in [0, p)$. It sets $\mathbf{s}^* := \boldsymbol{\beta} + c^* \cdot \mathbf{f}^*$. It sets $\mathsf{open}^*_s := \mathbf{s}^*$ and $\mathsf{C}^*_s := \phi(\mathbf{s}^*)$. It also sets $\mathsf{C}_\beta := \mathsf{C}^*_s - c^* \cdot \mathsf{C}_f$. It generates a transcript π^* for Eval with prover input (s^*, open^*_s) and verifier input $(\mathsf{C}^*_s, z, y_\beta + c^* \cdot y)$. It outputs the full simulated transcript $(\mathsf{C}_\beta, y_\beta, c^*, \mathsf{C}^*_s, \pi^*)$.

We will argue that the tuples $\mathsf{real} = (\mathsf{C}_\alpha, y_\alpha, c, \mathbf{s}, s)$ and $\mathsf{sim} = (\mathsf{C}_\beta, y_\beta, c^*, \mathbf{s}^*, s^*)$ have statistical distance at most $D \cdot 2^{-\kappa+\lambda}$. This implies that the conditional distributions of $\pi|\mathsf{real}$ and $\pi^*|\mathsf{sim}$ also have statistical distance bounded by $D \cdot 2^{-\kappa+\lambda}$ because these tuples fully determine the inputs to the Eval subprotocol. The values c^* and c are sampled identically. The vectors $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$ are identically distributed, hence y_β and y_α are also identically distributed. Assuming $||\mathsf{open}_f||_\infty \leq B$, the random variables $\mathbf{s}^* = \boldsymbol{\beta} + c^* \cdot \mathbf{f}^*$ and $\mathbf{s} = \boldsymbol{\alpha} + c \cdot \mathsf{open}_f$ each have statistical distance at most $D \cdot 2^{-\kappa+\lambda-1}$ from uniform over $[-2^\kappa \cdot B, 2^\kappa \cdot B]^d$ (Fact 1 and Fact 2) and therefore at most distance $D \cdot 2^{-\kappa+\lambda}$ from each other by the triangle inequality. Furthermore, $\chi(\boldsymbol{\alpha} + c \cdot \mathbf{f})(z) - y_\alpha = c \cdot y$ is identically distributed to $\chi(\mathbf{s}^*)(z) - y_\beta = c^* \cdot y$. The values $\mathsf{C}_\alpha = \phi(\mathbf{s}) - c \cdot \mathsf{C}_f$ and $\mathsf{C}^*_\alpha = \phi(\mathbf{s}^*) - c^* \cdot \mathsf{C}_f$ are identically determined by the other components of the tuple. Similarly, $s = \chi(\mathbf{s})$ and $s^* = \chi(\mathbf{s}^*)$ are identically determined.

As for soundness, the extractor $\mathcal{E}$ invokes tree-finding algorithm (Lemma 5) to get two accepting transcripts that share the same first message $\mathsf{C}_\alpha, \alpha(z)$ but have distinct challenges c, c', which define $\mathsf{C}_s, \mathsf{C}'_s$. $\mathcal{E}$ then invokes $\mathcal{E}_{\mathsf{Eval}}$ for $(\mathsf{C}_s, z, \alpha(z))$ and $(\mathsf{C}'_s, z, \alpha(z))$ respectively. By Lemma 6, with high probability the transcript has the property that the adversary succeeds in Eval on both (C_α, c) and (C_α, c') with non-negligible probability. In this case, $\mathcal{E}_{\mathsf{Eval}}$ outputs openings of C_s to $s \in \mathbb{F}^{(<d)}[X]$ such that $s(z) = \alpha(z) + c \cdot y$ and of C'_s to $s' \in \mathbb{F}^{(<d)}[X]$ such that $s'(z) = \alpha'(z) + c' \cdot y$ with overwhelming probability. If it does not succeed this step is repeated up to λ times, and the probability none succeeds is negligible.

Let $\mathbf{A} \in \mathbb{Z}^{2\times 2}$ be the matrix with rows $(1, c)$ and $(1, c')$, which is invertible over $\mathbb{F}_p$ because $c \neq c'$. Since $\mathsf{C}_\alpha + c \cdot \mathsf{C}_f = \mathsf{C}_s$ and $\mathsf{C}_\alpha + c' \cdot \mathsf{C}_f = \mathsf{C}'_s$, by Lemma 9 the extractor can efficiently compute openings of C_α to α and C_f to f such that $\langle \mathbf{A}, (\alpha, f) \rangle = (s, s')^\intercal$. This implies $\alpha(z) + c \cdot f(z) = s(z) = \alpha(z) + c \cdot y \bmod p$, i.e. $f(z) = y \bmod p$.

A.9 Batch evaluation protocol

A.9.1 Proof of Theorem 5

> If Eval is knowledge sound, then the protocol in Figure 3 is a proof of knowledge for the relation:

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

Proof. Let $\mathcal{A}$ be an adversary that succeeds in the protocol with non-negligible probability. We define a knowledge extractor $\mathcal{E}$ which will call the knowledge extractor $\mathcal{E}_{\mathsf{Eval}}$ for Eval as a black box.

Step 1: $\mathcal{E}$ runs with an adversary $\mathcal{A}$ and begins by using the tree-finding algorithm of Lemma 5 to generate a tree of $2k$ accepting transcripts that has the following properties:

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. Let $\mathbf{V} \in \mathbb{Z}^{k \times k}$ denote the Vandermonde matrix with jth row $(1, \rho_j, ..., \rho_j^{k-1})$ and let $\mathbf{R} \in \mathbb{Z}^{k \times k}$ 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 $\mathbb{F}_p$.

We will first show that the property $z(r) \neq z(r') \neq 0$ holds with overwhelming probability over $r, r' \xleftarrow{\$} \mathbb{F}$. Let $|\Omega| = m > 0$. Since $z \neq 0$ for non-empty Ω and $deg(z) \leq m$, by the fundamental theorem of algebra the probability that $z(r) = 0$ is at most $\frac{m}{|\mathbb{F}|}$. Similarly, define the non-zero polynomial $z'(X) := z(X) - z(r)$, then r' is a root of $z'(X)$ with probability at most $\frac{m}{|\mathbb{F}|}$. By a union bound $z(r) \neq z(r') \neq 0$ with probability at least $1 - \frac{3m}{|\mathbb{F}|}$.

Next, we will show that the third property holds with overwhelming probability. By Lemma 10, 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 except with probability $\frac{k^2}{|\mathbb{F}|}$. Moreover, for $\{r_j\}$ sampled uniformly and independently, $z_i(r_j) \neq 0 \bmod p$ except with probability $\frac{k}{|\mathbb{F}|}$. Thus, $\mathbf{A}$ is invertible for random $\{\rho_j, r_j\}$ with overwhelming probability.

Having established these facts, by Lemma 5 there is an algorithm that runs in time $\mathsf{poly}(\lambda)/\epsilon$ and with overwhelming probability generates a transcript tree satisfying the three properties above.

Step 2: For $j \in [k]$ let $\mathsf{C}_j^* := \sum_{i=1}^{k} \rho_j^{i-1} z_i(r_j) \cdot \mathsf{C}_i$ and let C_{q_j} denote the commitment sent by the prover in the transcript starting with ρ_j. The next step is to show that there exists a deterministic extraction algorithm that is given the transcript tree from **Step 1** and succeeds with non-negligible probability to extract:

- Valid openings of $\mathsf{C}_1^*, ..., \mathsf{C}_k^*$ to a vector of polynomials $\mathbf{f}^* = (f_1^*, ..., f_k^*) \in \mathbb{F}^{(<d)}[X]^k$
- Valid openings of $\mathsf{C}_{q_1}, ..., \mathsf{C}_{q_k}$ to polynomials $\mathbf{q} = (q_1, ..., q_k) \in \mathbb{F}[X]^k$ such that $f_i^*(r_i) = q_i(r_i) z(r_i)$ for all $i \in [k]$

The success probability of the deterministic extractor will be over the randomness of the transcript tree output. By repeating the transcript tree generation $\mathsf{poly}(\lambda)$ times we can amplify the probability of success to $1 - \mathsf{negl}(\lambda)$.

We will describe the algorithm to extract f_1^* and q_1 such that $f_1^*(r_1) = q_1(r_1) \cdot z(r_1)$; the algorithm will be symmetric for all other $j \in [k]$. Let $\mathsf{C}_g := \mathsf{C}_1^* - z(r_1) \cdot \mathsf{C}_{q_1}$ and $\mathsf{C}_{g'} := \mathsf{C}_1^* - z(r_1') \cdot \mathsf{C}_{q_1}$. Set any non-negligible $\delta < \epsilon/\mathsf{poly}(\lambda)$. By Lemma 6, with probability at least $1 - \frac{\delta}{\epsilon}\mathsf{poly}(\lambda)$, if $\mathcal{A}$ is rerun on partial transcripts in the tree it succeeds with probability at least δ. In particular, this means there is an Eval adversary $\mathcal{A}_{\mathsf{Eval}}$ that uses $\mathcal{A}$ to succeed in the evaluation protocol on public inputs $(\mathsf{C}_g, r_1, 0)$ and $(\mathsf{C}_g, r_1', 0)$ with probability at least δ. Therefore, with probability $1 - \frac{\delta}{\epsilon}\mathsf{poly}(\lambda)$, there exists $\mathcal{E}_{\mathsf{Eval}}$ that runs for time $\mathsf{poly}(\lambda)/\delta$ and with overwhelming probability succeeds in extracting valid openings of C_g and $\mathsf{C}_{g'}$ to polynomials $g(X)$ and $g'(X)$ such that $g(r_1) = g'(r_1') = 0$.

Assuming these steps have succeeded, let $\mathbf{M}$ be the 2×2 integer matrix with columns $(1, -z(r_1))$ and $(1, -z(r_1'))$ so that $(\mathsf{C}_g, \mathsf{C}_{g'}) \cdot \mathbf{M} = (\mathsf{C}_1^*, \mathsf{C}_{q_1})$. Applying Lemma 9, the extractor obtains polynomials $f_1^*, q_1 \in \mathbb{F}^{(<d)}[X]$ such that $f_1^* - z(r_1)q_1 = g \bmod p$ and $f_1^* - z(r_1')q_1 = g' \bmod p$, integers $t, t' \neq 0$, and openings of $t \cdot \mathsf{C}_1^*$ to $t \cdot f_1^* \bmod p$ and of $t' \cdot \mathsf{C}_{q_1}$ to $t' \cdot q \bmod p$. These are valid openings of C_1^* to f_1^* and C_{q_1} to q_1. Moreover, $f_1^*(r_1) = z(r_1)q(r_1) \bmod p$ and $f(r_1') = z(r_1')q(r_1') \bmod p$.

Step 3: Finally, we show there is an algorithm that takes the information extracted in **Step 2** and with overwhelming probability outputs valid openings of $\mathsf{C}_1, ..., \mathsf{C}_k$ to a list of polynomials $f_1, ..., f_k \in \mathbb{F}^{(<d)}[X]$ such that z divides $z_i \cdot f_i$ for each $i \in [k]$. This implies that $f_i(\Omega_i) = 0$ for each $i \in [k]$.

Let $\mathbf{C}^* = (\mathsf{C}_1^*, ..., \mathsf{C}_k^*)$ and $\mathbf{C} = (\mathsf{C}_1, ..., \mathsf{C}_k)$. We have that $\mathbf{A} \cdot \mathbf{C} = \mathbf{C}^*$ for $\mathbf{A} = \mathbf{V} \circ \mathbf{R}$. If $\mathbf{A}$ is invertible, then by Lemma 9 there is an efficient algorithm to compute valid openings of the components of $\mathbf{C}$ to a vector of polynomials $\mathbf{f} = (f_1, ..., f_k) \in (\mathbb{F}^{(<d)}[X])^k$ such that $\mathbf{A}\mathbf{f} = \mathbf{f}^*$. This implies that $\sum_{i=1}^k \rho_j^{i-1} z_i(r_j) f_i(r_j) = f_j^*(r_j) = q_j(r_j) \cdot z(r_j)$ for each $j \in [k]$. We can now argue that if $\sum_{i=1}^k \rho_j^{i-1} z_i \cdot f_i \neq q_i \cdot z$ then this contradicts the binding property of the PCS.

Let $h_j := \sum_{i=1}^k \rho_j^{i-1} z_i \cdot f_i - q_j \cdot z$. If $h \neq 0$ for $j \in [k]$, then based on Lemma 6 and Lemma 5, there is an efficient algorithm to generate a fresh transcript tree with the same fixed challenges ρ_j but fresh subtrees with new challenges $\{r_j'\}$ with the property that $h_j(r_j') \neq 0$. Repeating the extraction process above, with non-negligible probability this algorithm succeeds in computing openings of each C_{q_j} to a polynomial q_j' and an opening of each C_i to a polynomial f_i' such that $\sum_{i=1}^k \rho_j^{i-1} z_i(r_j') f_i'(r_j') = q_j'(r_j) \cdot z(r_j')$. Yet, since $h_j(r_j') \neq 0$ for all j, this implies that either $f_i' \neq f_i$ for some $i \in [k]$, or $q_j' \neq q_j$ for some $j \in [k]$. This would contradict the binding property of the PCS.

We conclude that with overwhelming probability $h_j = 0$ for all $j \in [k]$. Setting $\mathbf{z} := (z_1, ..., z_k)$ so that $\mathbf{z} \circ \mathbf{f} = (z_1 \cdot f_1, ..., z_k \cdot f_k)$, we have $\mathbf{V} \cdot (\mathbf{z} \circ \mathbf{f}) = z \cdot \mathbf{q} \bmod p$. Since $\mathbf{V}$ is invertible, this shows that $\mathbf{z} \circ \mathbf{f} = z \cdot \mathbf{V}^{-1}\mathbf{q}$. Therefore, every $z_i \cdot f_i$ is a multiple of z. $\square$

Lemma 10. *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 $\frac{n^2}{|\mathbb{F}|}$. $\square$

A.9.2 Proof of Theorem 6

> If Eval is knowledge sound, then the batch evaluation protocol in Figure 6 is a proof of knowledge for the relation $\mathcal{R}_{\mathsf{BatchEval}}(pp, d)$.

Proof. The extractor requires only a one line change to the extractor in the analysis of Theorem 5. Once the extractor obtains an opening for C_j^* to $f_j^* \in \mathbb{F}^{(<d)}[X]$ for each $j \in [k]$ such that

- $\mathsf{C}_j^* = \sum_{i=1}^{k} \rho_j^{i-1} z_i(r_j) \cdot (\mathsf{C}_i - t_i(r_j)) \cdot \mathsf{C}_1$
- $f_j^*(r_j) = q_j(r_j) \cdot z(r_j)$

it derives the commitments $\tilde{\mathsf{C}}_j := \mathsf{C}_j^* + \sum_{i=1}^{k} \rho_j^{i-1} z_i(r_j) \cdot t_i(r_j) \cdot \mathsf{C}_1$ and an opening of each $\tilde{\mathsf{C}}_j$ to the polynomial $\tilde{f} := f^* + \sum_{i=1}^{k} \rho_j^{i-1} z_i(r_j) \cdot t_i(r_j)$. Since $\tilde{\mathsf{C}}_j = \sum_{i=1}^{k} \rho_j^{i-1} z_i(r_j) \cdot \mathsf{C}_i$, the extractor can proceed in exactly the same way replacing each C_j^* with $\tilde{\mathsf{C}}_j$. The extractor obtains $f_1, ..., f_k$ such that $\sum_{i=1}^{k-1} \rho_j^{i-1} z_i(r_j) f_i(r_j) = \tilde{f}_j(r_j)$. Since $\tilde{f}_j = q_j(r_j) \cdot z(r_j) + \sum_{i=1}^{k} \rho_j^{i-1} z_i(r_j) \cdot t_i(r_j)$, the remainder of the analysis shows that $\sum_{i=1}^{k} \rho_j^{i-1} z_i \cdot (f_i - t_i) = q_i \cdot z$ with overwhelming probability. Finally, this implies that each $z_i \cdot (f_i - t_i)$ is a multiple of z, by inverting the Vandermonde matrix defined by the challenges ρ_j. In conclusion, $f_i(\Omega_i) = t_i(\Omega_i)$ for all $i \in [k]$. □

A.10 Zero knowledge HPI protocol

Lemma 4 stated:

> The transformed protocol is an $n \cdot 2^{-\lambda}$-statistical HVZK interactive protocol for relation $\mathcal{R}_{\mathsf{Bounded\text{-}HPI}}(\phi, \mathbb{G}, 2^\lambda)$, and a proof of knowledge for relation $\mathcal{R}^*_{\mathsf{HPI}}(\phi, \mathbb{G}, p)$.

Proof. The simulator samples an element $\tilde{z}$ of $\mathbb{G}$ by sampling a uniform random vector $\mathbf{z} \xleftarrow{\$} [0, 2^{2\lambda})^n$ and setting $\tilde{z} \leftarrow [\![\mathbf{z}]\!]_{\mathbf{g}}$. It samples $\tilde{c} \xleftarrow{\$} [0, 2^\lambda)$. It sets $\tilde{h} \leftarrow \tilde{z} - \tilde{c} \cdot y$. It generates a simulated transcript $\tilde{\pi}$ of the honest protocol for $\mathcal{R}^*_{\mathsf{HPI}}$ playing the roles of both prover/verifier on witness $\mathbf{z}$ and verifier input $\tilde{z}$. It outputs the simulated transcript $(\tilde{h}, \tilde{c}, \tilde{z}, \tilde{\pi})$.

The challenges $\tilde{c}$ and c are identically distributed and sampled independently from all other components of the transcripts. If the prover's witness $\mathbf{x}$ is in the bounded set $(-2^\lambda, 2^\lambda)^n$ then by Fact 1 and Fact 2 the statistical distance between $\mathbf{z}$ and $\mathbf{r} + c \cdot \mathbf{x}$ is at most $n \cdot 2^{-\lambda}$. The distributions of the transcripts π and $\tilde{\pi}$ are fully determined by witnesses $\mathbf{r} + c \cdot \mathbf{x}$ and $\mathbf{z}$ respectively, and thus $(\mathbf{z}, \tilde{\pi})$ and $(\mathbf{r} + c \cdot \mathbf{x}, \pi)$ have distance at most $n \cdot 2^{-\lambda}$. Finally, since $\tilde{z} = \phi(\mathbf{z})$, $h + c \cdot y = \phi(\mathbf{r} + c \cdot \mathbf{x})$, and $\tilde{h} = \tilde{z} - \tilde{c} \cdot y$, it follows that $(\tilde{h}, \tilde{c}, \tilde{z}, \tilde{\pi})$ and $(h, c, h + c \cdot y, \pi)$ have distance at most $n \cdot 2^{-\lambda}$.

As for soundness, the extractor $\mathcal{E}$ invokes the tree-finding algorithm (Lemma 5) to get two accepting transcripts that share the same first message h but have distinct challenges c, c', which define $z = h + c \cdot y$ and $z' = h + c' \cdot y$. By Lemma 6, with high probability the transcript has the property that the adversary succeeds in the subroutine for $\mathcal{R}^*_{\mathsf{HPI}}$ on both partial transcripts with non-negligible probability. In this case, the extractor $\mathcal{E}'$ for the $\mathcal{R}^*_{\mathsf{HPI}}$ subprotocol outputs witnesses $(t, s) \in \mathbb{Z} \times \mathbb{G}_1$ such that $\phi(s) = t \cdot z$ and $(t', s') \in \mathbb{Z} \times \mathbb{G}_2$ such that $\phi(s') = t' \cdot z'$, where $t \neq 0 \bmod p$ and $t' \neq 0 \bmod p$. If it does not succeed this step is repeated up to λ times. The probability none succeed is negligible in λ.

Let $\mathbf{T} \in \mathbb{Z}^{2\times 2}$ be the diagonal matrix with entries t and t'. Let $\mathbf{A} \in \mathbb{Z}^{2\times 2}$ be the matrix with rows $(1, c)$ and $(1, c')$ so that $\langle \mathbf{TA}, (h, y) \rangle = \langle \mathbf{T}, (z, z') \rangle$. Since $det(\mathbf{T}) = t \cdot t' \neq 0 \bmod p$ and

Figure 6: A protocol for simultaneously proving equality of multiple committed polynomials with multiple public polynomials on distinct sets: $\mathsf{C}_i = \mathsf{Commit}(pp, f_i)$, Ω_i is a non-empty subset of $\mathbb{F}$, and $t_i \in \mathbb{F}^{(<d)}[X]$ for all $i \in [k]$. The protocol shows that $f_i(\Omega_i) = t_i(\Omega_i)$. The pair $(\mathsf{C}^{(1)}, \mathsf{open}^{(1)}) \leftarrow \mathsf{Commit}(pp, 1)$ is a deterministic commitment to the constant polynomial $f \equiv 1$ that can be publicly derived. The prover's derivation of the opening string open for C_f from $o_1, ..., o_k, \mathsf{open}_q$ and $\mathsf{open}^{(1)}$ using add^* is not shown.

$\mathcal{P}(\{f_i, o_i, t_i, \Omega_i\}_{i\in[k]})$		$\mathcal{V}(\{\mathsf{C}_i, t_i, \Omega_i\}_{i\in[k]})$
$\Omega := \bigcup_{i=1}^k \Omega_i$ $z := \prod_{\omega\in\Omega}(X-\omega)$ $\forall_i\ \bar{\Omega}_i := \Omega \setminus \Omega_i$ $\forall_i\ z_i := \prod_{\omega\in\bar{\Omega}_i}(X-\omega)$		$\Omega := \bigcup_{i=1}^k \Omega_i$ $z := \prod_{\omega\in\Omega}(X-\omega)$ $\forall_i\ \bar{\Omega}_i := \Omega \setminus \Omega_i$ $\forall_i\ z_i := \prod_{\omega\in\bar{\Omega}_i}(X-\omega)$
	$\xleftarrow{\ \rho\ }$	$\rho \xleftarrow{\$} \mathbb{F}$
$q(X) := \sum_{i=1}^k \rho^{i-1} z_i (f_i - t_i)/z$ $(\mathsf{C}_q, \mathsf{open}_q) \leftarrow \mathsf{Commit}(pp, q)$	$\xrightarrow{\ \mathsf{C}_q\ }$	
	$\xleftarrow{\ r\ }$	$r \xleftarrow{\$} \mathbb{F}$
$g_i := z_i(r) \cdot (f_i - t_i(r))$ $g := \sum_{i=1}^k \rho^{i-1} g_i - z(r) \cdot q$ (if all is valid then $g(r) = 0$)		$\forall_i$ compute $z_i(r)$ $\mathsf{C}_i' := \mathsf{C}_i - t_i(r) \cdot \mathsf{C}^{(1)}$ $\mathsf{C}^* := \sum_{i=1}^k \rho^{i-1} z_i(r) \cdot \mathsf{C}_i'$ $\mathsf{C}_g := \mathsf{C}^* - z(r) \cdot \mathsf{C}_q$

$$\mathsf{Eval}\big(\mathcal{P}(g, \mathsf{open}_g, r), \mathcal{V}(\mathsf{C}_g, r, 0)\big)$$

$det(\mathbf{A}) = c - c' \neq 0 \bmod p$, both $\mathbf{T} \cdot \mathbf{A}$ is invertible over $\mathbb{F}$. There is a matrix $\mathbf{L} \in \mathbb{Z}^{2\times 2}$ such that $\mathbf{L} \cdot \mathbf{T} \cdot \mathbf{A} = \mathbf{D}$ is diagonal with entries d_1, d_2 such that $d_1 \neq 0 \bmod p$ and $d_2 \neq 0 \bmod p$. Let $\mathbf{L}_2$ denote the second row of $\mathbf{L}$, let $\mathbf{s} := (s, s')$, and let $\phi(\mathbf{s}) := (\phi(s), \phi(s'))$. The extractor obtains the witness $(d_2, \langle \mathbf{L}_2, \mathbf{s}\rangle) \in \mathbb{Z} \times \mathbb{G}_2$, which satisfies $d_2 \cdot y = \langle \mathbf{L}_2 \cdot \mathbf{T}, (z, z')\rangle = \langle \mathbf{L}_2, \phi(\mathbf{s})\rangle = \phi(\langle \mathbf{L}_2, \mathbf{s}\rangle)$. □

A.11 Halo proof recursion from PCS aggregation

Proof bootstrapping The construction we will describe is based on the recursive proving paradigm of Bitansky et. al. [BCCT13], also known as "proof bootstrapping", combined with a generalization of a technique described in the Halo protocol [BGH19]. A *recursive proof system* (S, P, V) for a path distributed computation with predicate $F : \mathbb{F}^{\ell_1} \times \mathbb{F}^{\ell_2} \to \mathbb{F}^{\ell_1}$ (informally) provides the ability to prove the statements $\phi(i, z_0, z_i)$ defined recursively, given $\mathsf{loc}_1, ..., \mathsf{loc}_i$, as:

> "there exists $z_{i-1} \in \mathbb{F}^{\ell_1}$, $w_i \in \mathbb{F}^{\ell_2}$, and a proof π_{i-1} such that $F(z_{i-1}, \mathsf{loc}_i) = z_i$ and the verification $V(\phi(i-1, z_0, z_{i-1}), \pi_{i-1})$ accepts"

Bitansky et. al. showed that starting from a SNARK system that has sublinear time verification it is possible to build a recursive proof system for path distributed computations, where the size and verification time of proofs, as well as the complexity of generating a proof for the incremental statement $\phi(i, z_0, z_i)$ given w_i and π_{i-1} are all independent of the recursion depth. Recently, Chiesa et. al. [COS20] improved upon this construction for the case of preprocessing SNARKs. A prover can use the recursive proof system to incrementally generate proofs for each step of the path distributed computation and publish only the last proof. This not only achieves a proof size and verification time independent of the depth, but also the prover's *space* complexity is independent of the depth t and its time complexity is linear in t. Bitansky et. al. call this a *complexity-preserving* SNARK, which was a primary motivation behind their recursive proof system. The system is used to "bootstrap" a normal SNARK (which may have expensive preprocessing, inefficient space complexity proportional to t, or a superlinear proving time) into a complexity-preserving one.

The main significance of the construction we will describe, which is based on the Halo protocol, is that we do not even start from an efficiently verifiable SNARK. Rather, we start with a succinct PCS, an efficient aggregation scheme for the PCS, and a Polynomial IOP (PIOP) for general programs (i.e., NP languages). Any PCS that is both succinct and *efficient* can be combined with any Polynomial IOP (PIOP) to build a SNARK [BFS20, CHM$^+$20], in which case the classical bootstrapping method works. However, in our case, it is ok for the PCS to have inefficient verification as long as it has an efficient aggregation scheme (i.e., with a good amortization ratio). Thus, the construction can be used to "bootstrap" this special class of PIOP-based SNARKs with inefficient verification into complexity-preserving SNARKs with efficient verification. Furthermore, in addition to enabling bootstrapping for a wider class of proof systems, the technique is also a practical improvement on the classical proof recursion method of Bitansky et. al. applied to PIOP-based SNARKs, when the underlying PCS has an efficient aggregation scheme. Based on the results of the prior sections, this includes any additive PCS, and even some non-additive schemes such as the FRI-based PCS.

Proof carrying data A recursive proof system has applications beyond bootstrapping. For example, it can be used inside the path distributed computation itself so that each node receives a proof along with the output of the previous node that attests to the correctness of all prior computations along the distributed path. Each node verifies this proof, performs its local computation, and produces an output along with a proof that it both verified the previous proof and performed the local computation correctly. This is called a *proof carrying data* (PCD) [CT10, BCCT13] system and generalizes to any DAG distributed computation. PCD systems also generalize *incrementally verifiable computation* (IVC), proposed by Valiant [Val08], where a machine outputs a proof after each step of computation that attests to the correct history of computation steps.

Private vs public aggregation An important distinction between the needs of proof bootstrapping and IVC versus PCD is that in the former the prover can retain an additional private state that helps it produce the next incremental proof whereas in the latter it cannot. In PCD any additional state must be included as a part of the proof because the next node/prover must be able to produce the next proof. In other words, PCD is sufficient but not necessary for proof bootstrapping, while a bootsrapping system is insufficient for PCD. This is relevant to our generalized Halo protocol: we present two variations of the protocol, one that uses public PCS aggregation and one that uses private PCS aggregation. Public PCS aggregation achieves a smaller communication between nodes of the PCD computation compared with private PCS aggregation. However, in our general constructions public aggregation is computationally more expensive than private aggregation. Private aggregation is superior for the purpose of constructing efficient SNARKs or IVC.

Bounded RAM programs A bounded RAM program is specified as a tuple (P, ℓ, m, t), where P is a program with a fixed-size read/write memory array of length ℓ, called the *work tape*, and t is an upper bound on the maximum number of steps for which P runs any input. The program is modeled as having a separate read-only memory array of length at most m that holds the inputs, called the *input tape*. In the context of a proof system, the input may be split into m_1 *public inputs* $\mathbf{x}$ and m_2 *private inputs* $\mathbf{w}$, where $m_1 + m_2 = m$. The verifier only receives $\mathbf{x}$ and the prover demonstrates existence/knowledge of $\mathbf{w}$ (also called the *witness*) such that the RAM program has a specified output (included as part of $\mathbf{x}$).

Bounded RAM computational reduction A proof system for bounded RAM programs (P, ℓ, m, t) that achieves verification time $\mathsf{poly}(m_1, |P|, \ell, \log t)$ for m_1 public inputs is sufficient to construct a computationally-sound proof system that achieves verification time $\mathsf{poly}(m_1, |P|, \log t)$ regardless of the memory bound [BEG+91, BCGT13]. Moreover, it is sufficient if the proof system assumes the bounded RAM program reads sequentially from the witness portion of the input. This construction uses Merkle trees. Any RAM computation P that runs for t steps using $O(t)$ space can be verified by a RAM program P', which uses only $O(\log t)$ space and runs for $O(t \log t)$ steps, provided a witness that contains Merkle proofs for the authenticated read/write operations of P.

Path distributed computation The iterated function $F : \mathbb{F}^\ell \to \mathbb{F}^\ell$ corresponds to a special case of a bounded RAM program that initially copies ℓ inputs to its work tape, and iterates for t steps on the work tape, never reading any more inputs. A more general bounded RAM program that reads sequentially from its input tape may be represented as a path distributed computation where each node along the path has ℓ_1 local inputs (coming from the input tape), ℓ_2 inputs that were outputs of the prior node, and ℓ_2 outputs. Each node computes the same function $F : \mathbb{F}^{\ell_1+\ell_2} \to \mathbb{F}^{\ell_2}$.

Preprocessing arithmetic circuits Theoretically, a proof system with these characteristics for bounded RAM programs making sequential witness reads is also sufficient to construct a preprocessing SNARK for arithmetic circuits. The preprocessing step produces a Merkle tree commitment to the wiring description of the circuit. A satisfying assignment to the circuit wires can be verified by a bounded RAM program that is provided an additional witness containing authenticated descriptions of each gate, which it verifies against the Merkle tree commitment.

Preprocessing SNARKs in the URS Model A preprocessing SNARK in the URS model consists of three algorithms (S, P, V). The URS is first sampled uniformly $urs \xleftarrow{\$} \{0,1\}^{\mathsf{poly}(\lambda)}$. This urs is an implicit input to S, P, and V, but we will drop it to avoid notational clutter. The setup algorithm S takes as input the description of any circuit C and outputs a verification key vk_C, a proving key pk_C. The prover algorithm P receives pk_C, $\mathbf{x}$, and $\mathbf{w}$ as input such that $C(\mathbf{x}, \mathbf{w}) = 1$ and outputs a proof π. The verification algorithm V receives vk_C, $\mathbf{x}$, and π as inputs and returns a binary output. There are two security properties, *completeness* and *knowledge-soundness*. The system is complete if with overwhelming probability in λ, over the randomness of the urs and keys returned by S, P will always succeed in creating a valid proof that V will accept when run on valid inputs. The system is knowledge-sound if for any circuit C, the non-interactive proof with the setup procedure that calls S on C is a proof of knowledge (Definition 12) for the relation of pairs $(\mathbf{x}, \mathbf{w})$ accepted by C. Technically, a SNARK is only required to be an *argument of knowledge*, which means that soundness holds only against efficient adversaries. We refer the reader to [COS20, BCI+13] for formal definitions of preprocessing SNARK completeness and knowledge-soundness properties. Preprocessing SNARKs also have complexity requirements. The algorithms S, P, and V are polynomial time. Let $|C|$ denote the length of the description of the circuit C that is an input to S, which returns (pk_C, vk_C). The size of proofs returned by the prover algorithm P running with pk_C must be $o(|C|)$. Some authors also require that the verifier algorithm V running with vk_C runs in time $o(|C|)$. However, we will distinguish such systems as *efficient* preprocessing SNARKs.

Proof recursion with preprocessing SNARKs Formally describing the proof system and a circuit that itself calls the code of the verifier requires care, especially since the setup procedure may need to preprocess the circuit. Proving the statements $\phi(i, z_0, z_i)$ for a path distributed computation, which were described informally in the introduction to this section, can be realized via a proof system for the following recursive program defined with respect to an efficient preprocessing SNARK (S, P, V) [COS20]:

Program $R(\mathbf{x}, \mathbf{w})$:

Public Input: Tuple $\mathbf{x} = (vk, i, z_i, z_0)$ where vk is a verification key and i is a counter.

Private Input: Tuple $\mathbf{w} = (z_{i-1}, \pi_{i-1}, \mathsf{loc}_i)$ where π_{i-1} is a SNARK proof.

Code: Output 1 if $i = 1$ and $z_1 = F(z_0, \mathsf{loc}_1)$, or if $i > 1$ and $z_i = F(z_{i-1}, \mathsf{loc}_i)$ and $V(vk, \mathbf{x}_{i-1}, \pi_{i-1}) = 1$, where $\mathbf{x}_{i-1} = (vk, i-1, z_{i-1}, z_0)$. Otherwise output 0.

The recursive proof system (S', P', V') for t steps[10] of the path distributed computation F with local inputs $(\mathsf{loc}_1, ..., \mathsf{loc}_t)$ operates as follows. S' runs S to preprocess a circuit description of $R(\mathbf{x}, \mathbf{w})$ to generate (pk_R, vk_R). The prover P' starts by computing $\pi_0 \leftarrow P(pk_R, (vk_R, 1, z_1, z_0), \perp)$. Then, given a valid proof π_{i-1} for the input $(vk_R, i-1, z_{i-1})$, and local input loc_i, P runs $P(pk_R, (vk_R, i, z_i, z_0), (z_{i-1}, \pi_{i-1}, \mathsf{loc}_i))$, which outputs a proof π_i. It does this for $i = 2, ..., t$. To verify the proof π_t, V' runs $V(vk_R, (vk_R, t, z_0, z_t), \pi_t)$. In the special case with no local inputs, the proof π_t attests to $F^{(t)}(z_0) = z_t$.

There remains a subtle catch. The circuit description of R requires a circuit description of the programs F and V. As the program V accepts arbitrarily large inputs, it cannot be described as a single circuit. Rather, V may be represented as a family of circuits $\{V_N : N \in \mathbb{N}\}$ where V_N runs on verification keys for circuits of size as most N. (The circuit size bound N also implicitly places an upper bound on the sizes of the verification key, input $\mathbf{x}$, and proof π). Finally, in order to successfully implement the proof recursion method above, S' must preprocess a circuit description

[10] Technically, for the proof system described here to be provably secure, the number of steps t must be a constant independent of the security parameter. This restriction comes from the security analysis in which the extractor requires a number of transcripts from the prover that grows exponentially in the recursion depth. This issue can be sidestepped by constructing a binary (or constant-arity) tree of recursive proofs where the leaves correspond to steps of the path distributed computation [BCCT13]. This way the recursion depth grows as $O(\log t)$, and is secure against sub-exponential adversaries (the extractor runs in superlinear time $\mathsf{poly}(\lambda, N)^{\log t}$). No known attacks exist on arbitrary depth recursion.

of R using some V_N (for sufficiently large $N \in \mathbb{N}$) such that the resulting circuit size is smaller than N. Otherwise, V_N would not accept vk_R as input. The size of R is approximately $|F| + |V_N|$ and $|R| < N$ only if $|V_N| < N - |F|$. Since the size of V_N is asymptotically $o(N)$, there exists sufficiently large N such that $|V_N| < N - |F|$.

This is captured in the following theorem.[11] Let $V_{N,\lambda,\ell}$ denote the verification circuit of the proof system (S, P, V) for security parameter λ, which accepts verification keys for circuits of size at most N and input instances of size at most ℓ. Given $F : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$, define the binary relation $\mathsf{PDC}(F, t)$ over instances $\mathbf{x} = (z_0, z_t) \in \mathbb{F}^{2\ell}$ paired with witnesses $\mathbf{w} = ((z_1, w_1), ..., (z_{t-1}, w_{t-1})) \in \mathbb{F}^{2\ell(t-1)}$ satisfying $\forall_{i>0}\ F(z_{i-1}, w_i) = z_i$. Given a preprocessing SNARK (S, P, V) in the URS model, let $(S', P', V') \leftarrow \mathbb{T}(S, P, V)$ denote the transformed proof system described above.

Theorem 11 ([COS20])**.** *If (S, P, V) is a preprocessing SNARK for binary relations in the URS model then for any function $F : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$ and constant $t \in \mathbb{N}$ the proof system $(S', P', V') = \mathbb{T}(S, P, V)$ is a preprocessing non-interactive argument of knowledge for the binary relation $\mathsf{PDC}(F, t)$. If $|V_{N,\lambda,\ell}| < N^{1-\epsilon} \cdot \mathsf{poly}(\lambda, \ell)$ for some $\epsilon \in (0, 1)$, then S', P', and V' run in time equal to S, P, and V respectively on circuits of size $|F| + O(\mathsf{poly}(\lambda, \ell)^{1/\epsilon})$ with inputs of size $O(\lambda + \ell)$.*

In particular, if the preprocessing SNARK has a polylogarithmic verifier, *i.e.*, $|V_{N,\lambda,\ell}| < \log^c N \cdot \mathsf{poly}(\lambda, \ell)$, then for any $\delta > 0$ the running times of S', P', and V' are upper bounded by the running times of S, P, and V respectively on circuits of size $|F| + o(\mathsf{poly}(\lambda, \ell)^{1+\delta})$ with inputs of size $O(\lambda + \ell)$.

We are now ready to describe how the Halo construction may be generalized to combine any Polynomial IOP (PIOP) with any *aggregatable* PCS.

There are two essential building blocks to Halo proof recursion. The first building block is a PIOP-based preprocessing SNARK $(\mathcal{S}, \mathcal{P}, \mathcal{V})$. This SNARK must be succinct, but is not strictly required to have efficient verification as we will see. A PIOP-based preprocessing SNARK compiles a preprocessing PIOP with a non-interactive polynomial commitment scheme $\mathcal{PCS}$. Let $\mathbb{G}$ denote the commitment group of $\mathcal{PCS}$. The SNARK proofs of $(\mathcal{S}, \mathcal{P}, \mathcal{V})$ have the form $(\boldsymbol{\rho}, \pi = (\mathbf{C}, \mathbf{u}, \mathbf{v}, \alpha))$ where $\mathbf{C} \in \mathbb{G}^k$ is a vector of polynomial commitments for a PCS scheme with commitment group $\mathbb{G}$ to polynomials $f_1, ..., f_k \in \mathbb{F}^{(<d)}[X]$, $\mathbf{u} \in \mathbb{F}^k$, $\mathbf{v} \in \mathbb{F}^k$, $f_i(u_i) = v_i$ for all $i \in [k]$, $\rho = (\rho_1, ..., \rho_k)$ is a vector of non-interactive evaluation proofs for $\mathcal{PCS}$ where ρ_i is an evaluation proof opening C_i at $u_i \in \mathbb{F}$ to $v_i \in \mathbb{F}$, and finally α is additional auxiliary content. Moreover, the verifier $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2)$ running on a public input $\mathbf{x}$, an input polynomial commitment vk called the verification key, and a proof π consists of two parts:

1. $\mathcal{V}_1(vk, \mathbf{x}, \pi)$ runs in time sublinear in the size of the circuit corresponding to the preprocessed verification key vk. It may also run in time linear in the size of the inputs $(vk, \mathbf{x}, \pi)$.

2. $\mathcal{V}_2(\boldsymbol{\rho}, \mathbf{C}, \mathbf{u}, \mathbf{v})$ verifies each $\mathsf{NI\text{-}Eval}$ proof ρ_i with (C_i, u_i, v_i) for each $i \in [k]$.

The second building block is a non-interactive efficient aggregation scheme for $\mathcal{PCS}$ (Definitions 8-9). We will discuss the implications of using a private vs public aggregation scheme later. Given that we require an aggregation scheme, we may actually assume that the proofs of $(\mathcal{S}, \mathcal{P}, \mathcal{V})$ consist of only one $\mathsf{NI\text{-}Eval}$ at a single point (i.e., $k = 1$) because otherwise the aggregation scheme can first be applied to achieve this. Letting $\mathsf{open} = (\mathsf{open}_1, ..., \mathsf{open}_k)$ denote $\mathcal{P}$'s opening strings for commitments $\mathbf{C}$, the new prover $\mathcal{P}'$ would do the following:

[11] The theorem quoted here is a special case of the more general theorem by Chiesa *et. al.* [COS20] for *proof carrying data* (PCD) systems.

1. Run $\mathcal{P}$ to get the original proof $(\boldsymbol{\rho}, \pi = (\mathbf{C}, \mathbf{u}, \mathbf{v}, \alpha))$.
2. Run $((\mathsf{open}', f'), (C', u', v'), \mathsf{tr}) \leftarrow \mathsf{NI\text{-}Aggregate}(\mathbf{f}, \mathsf{open}, \mathbf{C}, \mathbf{u}, \mathbf{v})$
3. Compute the aggregate opening $\rho' \leftarrow \mathsf{NI\text{-}Eval}((\mathsf{open}', f'), (C', u', v'))$.
4. Modify the SNARK proof by replacing the original commitments/openings with the aggregate commitment C' and opening ρ', and appending the original commitments, openings, along with the aggregation transcript to the auxiliary string. The new proof is (ρ', π') where $\pi' = (C', u', v', \alpha')$ where $\alpha' = (\alpha, \mathbf{C}, \mathbf{u}, \mathbf{v}, \mathsf{tr})$.

The new verification algorithm $\mathcal{V}_1'$ would still run $\mathcal{V}_1(vk, \mathbf{x}, (\mathbf{C}, \mathbf{u}, \mathbf{v}, \alpha))$ and additionally run the aggregation protocol verification of $(C', u', v', \mathsf{tr})$. $\mathcal{V}_2'$ receives (ρ', C', u', v') and verifies the single $\mathsf{NI\text{-}Eval}$ proof ρ'.

Protocol overview Given these two building blocks, the main idea in Halo is to only include $\mathcal{V}_1$, the sublinear component of $\mathcal{V}$, inside the recursion circuit, and to pass the "unverified" polynomial commitment evaluation tuple $(C, \mathsf{pt}) \in \mathbb{G} \times \mathbb{F}^2$ as an additional public input to the external verifier. Let us revisit the task of proving the iterated function $F^{(t)}(z_0) = z_t$ where $F : \mathbb{F}^\ell \to \mathbb{F}^\ell$. Consider first a strawman construction: at each ith step in the recursion chain the prover generates a new proof $(\rho_i, \pi_i) = (\rho_i, (C_i, \mathsf{pt}_i, \alpha_i))$ attesting to the next incremental step of the computation and knowledge of the last recursive proof $(\rho_{i-1}, \pi_{i-1}) = (\rho_{i-1}, (C_{i-1}, \mathsf{pt}_{i-1}, \alpha_{i-1}))$ such that $\mathcal{V}_1$ accepts the proof component $(C_{i-1}, \mathsf{pt}_{i-1}, \alpha_{i-1})$. This proof alone is not a sound proof of the computation's integrity. However, if the verifier were additionally provided $(\rho_{i-1}, C_{i-1}, \mathsf{pt}_{i-1})$, then it could run $\mathcal{V}_1$ verification of π_i along with the Eval verifications $\mathcal{V}_2(\rho_{i-1}, C_{i-1}, \mathsf{pt}_{i-1})$ and $\mathcal{V}_2(\rho_i, C_i, \mathsf{pt}_i)$. This would complete verification of the inner proof that is part of the witness and also the incremental step. Moreover, the two required Eval verifications could be combined using the aggregation scheme. Let $\mathsf{pt}_i = (u_i, v_i)$ for each $i \in \mathbb{N}$, the prover runs:

$$\mathsf{NI\text{-}Aggregate}((f_{i-1}, f_i), (\mathsf{open}_{i-1}, \mathsf{open}_i), (C_{i-1}, C_i), (u_{i-1}, u_i), (v_{i-1}, v_i))$$

The output to the prover is (open^*, f^*) and the public output is $(C^*, \mathsf{pt}^*, \mathsf{tr})$. If the prover includes $(C^*, \mathsf{pt}^*, \mathsf{tr})$ in its incremental proof and additionally provides an $\mathsf{NI\text{-}Eval}$ proof ρ^* for (C^*, pt^*), then the verifier could check that V_1 accepts $(C_i, \mathsf{pt}_i, \alpha_i)$, check that V_2 accepts $(\rho^*, C^*, \mathsf{pt}^*)$, and run the aggregation verification given tr. This leads to the following strategy. The tuple (C^*, pt^*), which is necessary for completing the verification of the ith computation transition, will be a public input to the proof for the $(i+1)$st transition. The next proof π_{i+1} will also prove (i.e., include as a part of the circuit) the verifier $\mathcal{V}_{\mathsf{agg}}$ of the aggregation step that produces the tuple (C^*, pt^*). The values $\mathsf{tr}, C_i, C_{i-1}, \mathsf{pt}_i$, and pt_{i-1} are witnesses. The circuit for π_{i+1} also includes the $\mathcal{V}_1$ verification of $(C_i, \mathsf{pt}_i, \alpha_i)$. This requires the prover to save $(C_{i-1}, \mathsf{pt}_{i-1})$ as part of the witness in addition to π_i. Likewise, both (C^*, pt^*) and π_{i+1} will become part of the witness for the next proof, and so on and so forth.

Efficiency If $\mathcal{V}_1$ and $\mathcal{V}_{\mathsf{agg}}$ are together sublinear in the size of circuits preprocessed by S, then by Theorem 11 the recursion circuit is well defined. This is guaranteed by the efficiency requirements on $\mathcal{V}_1$ and the aggregation scheme so long as the PIOP-based SNARK commits to polynomials of maximum degree linear in the size of the preprocessed circuit. (This is the case in all practical

PIOP constructions as otherwise the prover time is impractical). Moreover, if $|\mathcal{V}_1| + |\mathcal{V}_{\mathsf{agg}}|$ is poly-logarithmic in the size of preprocessed circuit then the recursion circuit has size approximately $|F| + O(\mathsf{poly}(\lambda, \ell))$.

The prover only needs to derive the $\mathsf{NI\text{-}Eval}$ proof for the last polynomial commitment output at the end of the recursive proof chain. This commitment is a single element in $\mathbb{G}$. The $\mathsf{NI\text{-}Eval}$ proofs for other intermediate commitments are never actually used. The verifier only needs to check this one final $\mathsf{NI\text{-}Eval}$ proof for the entire proof recursion, which is for a polynomial of degree proportional to $N = |F| + |\mathcal{V}_1| + |\mathcal{V}_{\mathsf{agg}}|$. The verifier also runs $\mathcal{V}_1$ once on the recursion circuit. Let $S_{\mathsf{agg}}(\lambda, d, k)$ denote the worst case size complexity of the aggregation protocol's output commitment given k input commitments to polynomials of degree d with $\mathcal{PCS}$ security parameter λ. This is at most the maximum representation size of a single group element in $\mathbb{G}$. In the special case that $|\mathcal{V}_1| + |\mathcal{V}_{\mathsf{agg}}|$ is poly-logarithmic, the final proof size is $\mathsf{polylog}(\lambda, |F|, \ell) + S_{\mathsf{agg}}(\lambda, d, 2)$. For any PCS that has a size-optimal linear combination scheme, there is an aggregation scheme with $S_{\mathsf{agg}}(\lambda, d, k) \leq S_{\mathsf{Eval}}(\lambda, d)$, where $S_{\mathsf{Eval}}(\lambda, d)$ denotes the maximum size of commitments to polynomials of degree d (Theorem 4).

When does Halo help? The generalized Halo technique we have described improves over the standard method of proof recursion, reducing the circuit complexity of the recursive statement, when $\mathcal{V}_{\mathsf{agg}}$ is smaller than $\mathcal{V}_2$ (i.e., the $\mathsf{NI\text{-}Eval}$ verifier). As a special case, in the case that $\mathcal{V}_2$ is inefficient (i.e, is not sublinear in the statement size) any efficient aggregation scheme will have this property. In such cases the classical proof recursion method does not work.

Halo without succinct Eval Since the evaluation proofs are not included inside the recursion and only produced in the last step, the protocol still works if $\mathcal{PCS}$ does not have a succinct evaluation protocol. The prover only needs to open the coefficients of one degree N polynomial for the entire proof chain. The final proof *size* will be both asymptotically and concretely larger than with a succinct PCS, but still proportional to $|F| + O(\mathsf{poly}(\lambda, \ell))$ rather than the depth of the recursion.

Detailed construction In more detail, can rewrite the recursive program $R(\mathbf{x}, \mathbf{w})$ for the program $F : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$ as follows:

Program $R'(\mathbf{x}, \mathbf{w})$:

Public Input: Tuple $\mathbf{x} = (vk, i, \mathsf{tr}_i, C_i^*, \mathsf{pt}_i^*, z_i, z_0)$ where vk is a verification key (a polynomial commitment), $i \in \mathbb{N}$ is a counter, C_i^* is a polynomial commitment, $\mathsf{pt}_i^* = (u_i, v_i) \in \mathbb{F}^2$, and tr_i is a $\mathsf{NI\text{-}Aggregate}$ transcript. Missing components of $\mathbf{x}$ are allowed and are indicated with $\perp$.

Private Input: Tuple $\mathbf{w} = (C_{i-1}^*, \mathsf{pt}_{i-1}^*, z_{i-1}, \pi_{i-1}, \mathsf{loc}_i)$ where $\pi_{i-1} = (C_{i-1}, \mathsf{pt}_{i-1}, \alpha_{i-1})$ is a SNARK proof, C_{i-1}^* is a polynomial commitment, and $\mathsf{pt}_{i-1}^* \in \mathbb{F}^2$. Missing components of $\mathbf{w}$ are indicated with $\perp$.

Code: Output 1 if:

- $i = 1$ and $z_1 = F(z_0, \mathsf{loc}_1)$
- $i = 2$ and $z_2 = F(z_1, \mathsf{loc}_2)$ and $(C_2^*, \mathsf{pt}_2^*) = (C_1, \mathsf{pt}_1)$, and $V_1(vk, (vk, 1, \perp, \perp, \perp, z_2, z_0), (C_1, \mathsf{pt}_1, \alpha_1)) = 1$.
- $i = 3$ and $z_3 = F(z_2, \mathsf{loc}_3)$ and $\mathcal{V}_{\mathsf{agg}}(\mathsf{tr}_3, C_3^*, \mathsf{pt}_3^*, C_2, \mathsf{pt}_2, C_1, \mathsf{pt}_1) = 1$, and $V_1(vk, (vk, 2, \perp, C_1, \mathsf{pt}_1, z_2, z_0), (C_2, \mathsf{pt}_2, \alpha_2)) = 1$.
- $i > 3$ and $z_i = F(z_{i-1}, \mathsf{loc}_i)$ and $V_1(vk, \mathbf{x}_{i-1}, \pi_{i-1}) = 1$, where $\mathbf{x}_{i-1} = (vk, i-1, \mathsf{tr}_{i-1}, C_{i-1}^*, \mathsf{pt}_{i-1}^*, z_{i-1}, z_0)$, and $\mathcal{V}_{\mathsf{agg}}(\mathsf{tr}_i, C_i^*, \mathsf{pt}_i^*, C_{i-1}^*, \mathsf{pt}_{i-1}^*, C_{i-1}, \mathsf{pt}_{i-1}) = 1$.

Otherwise output 0.

The proof system $(\mathcal{S}', \mathcal{P}', \mathcal{V}')$ operates as follows. $\mathcal{S}'$ runs $\mathcal{S}$ to preprocess a circuit description of $R(\mathbf{x}, \mathbf{w})$ and outputs (pk_R, vk_R). The prover proceeds according to the following steps:

1. P' starts by computing $z_1 \leftarrow F(z_0, \mathsf{loc}_1)$ and $(C_1, \mathsf{pt}_1, \alpha_1) \leftarrow P(pk_R, \mathbf{x}_1, \mathbf{w}_1)$ where $\mathbf{x}_1 = (vk_R, 1, \perp, \perp, \perp, z_1, z_0)$ and $\mathbf{w}_1 = (\perp, \perp, \perp, \perp, \mathsf{loc}_1)$. It sets $\pi_1 := (C_1, \mathsf{pt}_1, \alpha_1)$.

   Intermediate verification: Given additionally a $\mathsf{NI\text{-}Eval}$ proof ρ_1 for (C_1, pt_1), the proof π_0 for the public inputs (z_1, z_0) could be verified using $V_1(vk_R, \mathbf{x}_1, \pi_1)$ and $V_2(\rho_1, C_1, \mathsf{pt}_1)$.

2. P' computes $z_2 \leftarrow F(z_1, \mathsf{loc}_2)$ and $(C_2, \mathsf{pt}_2, \alpha_2) \leftarrow P(pk_R, \mathbf{x}_2, \mathbf{w}_2)$ where $\mathbf{x}_2 = (vk_R, 2, \perp, C_1, \mathsf{pt}_1, z_2, z_0)$ and $\mathbf{w}_2 = (\perp, \perp, z_1, \pi_1, \mathsf{loc}_2)$. It sets $\pi_2 = (C_2, \mathsf{pt}_2, \alpha_2)$.

   Intermediate verification: Given additionally an aggregate $\mathsf{NI\text{-}Eval}$ tuple $(C_3^*, \mathsf{pt}_3^*, \rho_3^*)$ and aggregation transcript tr_3, the proof π_2 for public inputs (C_1, r_1, z_2, z_0) may be verified by running $\mathcal{V}_{\mathsf{agg}}(\mathsf{tr}, C_3^*, \mathsf{pt}_3^*, C_2, \mathsf{pt}_2, C_1, \mathsf{pt}_1)$, running $V_1(vk_R, \mathbf{x}_2, \pi_2)$, and running $V_2(\rho_3^*, C_3^*, \mathsf{pt}_3^*)$.

3. $\mathcal{P}'$ computes the aggregation:

   $$(\mathsf{open}_3^*, f_3^*, C_3^*, \mathsf{pt}_3^*, \mathsf{tr}_3) \leftarrow \mathsf{NI\text{-}Aggregate}(f_1, f_2, \mathsf{open}_1, \mathsf{open}_2, C_1, C_2, u_1, u_2, v_1, v_2)$$

   where $\mathsf{pt}_i = (u_i, v_i)$ and open_i is an opening of C_i to f_i such that $f_i(u_i) = v_i$ for $i \in \{1, 2\}$. It also computes $z_3 \leftarrow F(z_2, \mathsf{loc}_3)$ and $(C_3, \mathsf{pt}_3, \alpha_3) \leftarrow P(pk_R, \mathbf{x}_3, \mathbf{w}_3)$ where $\mathbf{x}_3 = (vk_R, 3, C_3^*, r_3^*, z_3, z_0)$ and $\mathbf{w}_3 = (C_1, \mathsf{pt}_1, z_2, \pi_2, \mathsf{loc}_3))$. It sets $\pi_3 := (C_3, \mathsf{pt}_3, \alpha_3)$.

Intermediate verification: Given $(C_4^*, \mathsf{pt}_4^*, \rho_4^*, \mathsf{tr}_4)$, the verifier checks the proof π_3 for public inputs $(C_3^*, \mathsf{pt}_3^*, z_3, z_0)$ by running $\mathcal{V}_{\mathsf{agg}}(\mathsf{tr}_4, C_4^*, \mathsf{pt}_4^*, C_3^*, \mathsf{pt}_3^*, C_3, \mathsf{pt}_3)$, running $V_1(vk_R, \mathbf{x}_3, \pi_3)$, and running $V_2(\rho_4^*, C_4^*, \mathsf{pt}_4^*)$.

4. For $i \geq 4$: given $\pi_{i-1} = (C_{i-1}, r_{i-1}, \alpha_{i-1})$ for the public input $(vk_R, i-1, C_{i-1}^*, \mathsf{pt}_{i-1}^*, z_{i-1}, z_0)$, $\mathcal{P}'$ computes:

$$(\mathsf{open}_i^*, f_i^*, C_i^*, \mathsf{pt}_i^*, \mathsf{tr}_i) \leftarrow \mathsf{NI\text{-}Aggregate}(f_{i-1}^*, f_{i-1}, \mathsf{open}_{i-1}^*, \mathsf{open}_{i-1}, C_{i-1}^*, C_i, u_{i-1}^*, u_i, v_{i-1}^*, v_i)$$

where $\mathsf{pt}_i = (u_i, v_i)$ and open_i is an opening of C_i to f_i such that $f_i(u_i) = v_i$. It also computes $z_i = F(z_{i-1}, \mathsf{loc}_i)$ and $\pi_i \leftarrow P(pk_R, \mathbf{x}_i, \mathbf{w}_i)$ where $\mathbf{x}_i = (vk_R, i, C_i^*, \mathsf{pt}_i^*, z_i, z_0)$ and $\mathbf{w}_i = (C_{i-1}^*, \mathsf{pt}_{i-1}^*, z_{i-1}, \pi_{i-1}, \mathsf{loc}_i)$.

Intermediate verification: Given $(C_{i+1}^*, \mathsf{pt}_{i+1}^*, \rho_{i+1}^*, \mathsf{tr}_{i+1})$, the verifier checks the proof π_i with public inputs $(C_i^*, \mathsf{pt}_i^*, z_i, z_0)$ by running $\mathcal{V}_{\mathsf{agg}}(\mathsf{tr}_{i+1}, C_{i+1}^*, \mathsf{pt}_{i+1}^*, C_i^*, \mathsf{pt}_i^*, C_i, \mathsf{pt}_i)$, running $\mathcal{V}_1(vk_R, x_i, \pi_i)$ and $\mathcal{V}_2(\rho_{i+1}^*, C_{i+1}^*, \mathsf{pt}_{i+1}^*)$.

5. **Final proof:** P outputs both $\pi_t = (C_t, \mathsf{pt}_t, \alpha_t)$ and (C_t^*, pt_t^*) along with a batch evaluation proof for both C_t and C_t^*, i.e. a $\mathsf{NI\text{-}Eval}$ proof ρ_{t+1}^* for the aggregate tuple:

$$(\mathsf{open}_{t+1}^*, f_{t+1}^*, C_{t+1}^*, \mathsf{pt}_{t+1}^*, \mathsf{tr}_{t+1}) \leftarrow \mathsf{NI\text{-}Aggregate}(f_t^*, f_t, \mathsf{open}_t^*, \mathsf{open}_t, C_t^*, C_i, u_t^*, u_t, v_t^*, v_t)$$

Final verification: Run $\mathcal{V}_1(vk_R, \mathbf{x}_t, \pi_t)$, run $\mathcal{V}_{\mathsf{agg}}(\mathsf{tr}_{t+1}, C_{t+1}^*, \mathsf{pt}_{t+1}^*, C_t^*, \mathsf{pt}_t^*, C_t, \mathsf{pt}_t)$, and lastly run $\mathcal{V}_2(\rho_{t+1}^*, C_{t+1}^*, \mathsf{pt}_{t+1}^*)$.

Stateless Halo for Proof Carrying Data In the original Halo protocol the prover was *stateless*. In other words, the prover could output the proof π_i for $i < t$, and another prover could produce π_{i+1} without knowing anything about the first prover's internal state. This property is not critical when applying this protocol to obtain SNARKs for bounded RAM programs. However, this property is critical for PCD and its many applications.

The simplest solution is to include the prover's private state as part of the proof. This consists of two polynomial commitment openings. Specifically, the prover for the ith incremental statement receives the proof $\pi_{i-1} = (C_{i-1}, \mathsf{pt}_{i-1}, \alpha_{i-1})$, the values $(C_{i-1}^*, \mathsf{pt}_{i-1}^*, z_{i-1}, \mathsf{loc}_i)$, and additionally the previously private polynomials and opening strings $(f_{i-1}^*, f_{i-1}, \mathsf{open}_{i-1}^*, \mathsf{open}_{i-1})$ for the commitments C_{i-1}^* and C_{i-1} respectively.

The proof sizes are now linear in the predicate F, but still independent of the computation depth. This achieves the same proof size as implementing Halo without a succinct PCS (i.e., a PCS with succinct commitments but linear size evaluation proofs). In fact, there is little benefit to using a PCS with a sublinear evaluation proof when employing this trivial method for stateless proving. An PCS with an efficient aggregation protocol would suffice (e.g., using any homomorphic hash function).

However, there is a more efficient method of achieving stateless proving, which uses a public aggregation scheme instead of a private aggregation scheme.

Public aggregation method Recall (from Theorem 7) that if the PCS is additively succinct then its private aggregation scheme can be efficiently compiled into a public aggregation scheme with only a small additive overhead, logarithmic in the maximum degree of the committed polynomials. This includes all the additive schemes in Section 3.2, but excludes FRI.

Replacing the private non-interactive aggregation scheme with a public non-interactive aggregation scheme in the protocol described above is straightforward. The main difference is that the public aggregation algorithm requires as input $\mathsf{NI\text{-}Eval}$ proofs instead of commitment openings. Instead of running the private aggregation algorithm, which requires $(f^*_{i-1}, f_{i-1}, \mathsf{open}^*_{i-1}, \mathsf{open}_{i-1})$, at each step $i \in \mathbb{N}$ the ith prover runs the public aggregation algorithm:

$$(\mathsf{open}^*_i, f^*_i, C^*_i, \mathsf{pt}^*_i, \mathsf{tr}_i) \leftarrow \mathsf{NI\text{-}Aggregate}(C^*_{i-1}, C_i, u^*_{i-1}, u_i, v^*_{i-1}, v_i, \rho^*_{i-1}, \rho_i)$$

where ρ^*_{i-1} is an $\mathsf{NI\text{-}Eval}$ proof for the tuple $(C^*_{i-1}, u^*_{i-1}, v^*_{i-1})$ and ρ_{i-1} is an $\mathsf{NI\text{-}Eval}$ proof for the tuple $(C_{i-1}, u_{i-1}, v_{i-1})$. The value ρ_{i-1} is already part of π_{i-1} and ρ^*_{i-1} is included in the output of the $i-1$st prover. The ith prover uses $(\mathsf{open}^*_i, f^*_i)$ to produce the evaluation proof ρ^*_i for (C^*_i, pt^*_i). The public output of the ith prover is $(\rho^*_i, C^*_i, \mathsf{pt}^*_i, z_i, \pi_i)$. Crucially, the prover does *not* need to include $(\mathsf{open}^*_i, f^*_i)$ as part of the public output.

B Our results for the KZG scheme and applications to pairing-based SNARKs

In this section, we focus on batch evaluation for the original PCS of Kate, Zaverucha and Goldberg [KZG10] and its implications for the PLONK zk-SNARK [GWC19]. [KZG10] presented a pairing-based scheme where an opening proof π consists of a single $\mathbb{G}_1$ group element.

Related previous results [MBKM19], who introduced the use of [KZG10] for universal and updatable SNARKs, modified the PCS of [KZG10] in the random oracle model, so that a single $\mathbb{G}_1$ element can be an opening proof for several polynomials *at the same point* $z \in \mathbb{F}$. [Gab19, CHM+20, GWC19] followed and used similar single-point multi-polynomial batching protocols.

[KZG10] give in their paper a less known version of their scheme allowing for a one $\mathbb{G}_1$ element opening proof for *one* polynomial at several evaluation points.

For the case of multiple polynomials and evaluation points, [CHM+20, GWC19] use randomized techniques for batching pairing equations to improve verification efficiency; however opening proof size and prover computation still grow linearly with the number of distinct points.

Our results In this section, we give two extensions of KZG for multiple evaluation points and polynomials.

- In our first scheme the opening proof is only a single $\mathbb{G}_1$ element, but verifier operations are considerably heavier than previous variants of [KZG10] when the number of distinct evaluation points is large (cf. Lemma 16).
- In our second scheme the opening proof is two $\mathbb{G}_1$ elements, and the verifier complexity is somewhat better than previous multipoint variants of [KZG10] (cf. Lemma 17). This scheme is simply the instantiation for KZG of our private aggregation for any additive PCS; however, we give a simplified proof for this instance using the algebraic group model of [FKL18] (which is needed in any case to prove the basic PCS of [KZG10] is secure).

We compare the performance of our PCS to a more straightforward batched version of the [KZG10] scheme as in [GWC19]. For simplicity, we look at the restricted case where we want to open t polynomials all with the same degree bound n, each at one *distinct* point. See Lemma 16 and 17 for the more detailed efficiency properties in the general case (where each polynomial is opened at a subset of points, and the subsets may repeat).

Table 1: Comparison of opening complexity for t polynomials on t distinct points. In prover/verifier work columns $\mathbb{G}_i$ means scalar multiplication in $\mathbb{G}_i$, $\mathbb{F}$ means addition or multiplication in $\mathbb{F}$, and $\mathbf{P}$ means pairing.

	SRS size	prover work	proof length	verifier work
KZG as in [GWC19]	$n\ \mathbb{G}_1$, $2\ \mathbb{G}_2$	$t \cdot n\ \mathbb{G}_1$, $O(t \cdot n \log n)\ \mathbb{F}$	$t\ \mathbb{G}_1$	$3t-2\ \mathbb{G}_1$, $2\ \mathbf{P}$
This work, ver. 1	$n\ \mathbb{G}_1$, $t+1\ \mathbb{G}_2$	$n\ \mathbb{G}_1$, $O(t \cdot n + n \log n)\ \mathbb{F}$	$1\ \mathbb{G}_1$	$t-1\ \mathbb{G}_1$, $t^2\ \mathbb{G}_2$, $t+1\ \mathbf{P}$
This work, ver. 2	$n\ \mathbb{G}_1$, $2\ \mathbb{G}_2$	$2n\ \mathbb{G}_1$, $O(t \cdot n + n \log n)\ \mathbb{F}$	$2\ \mathbb{G}_1$	$t+3\ \mathbb{G}_1$, $2\ \mathbf{P}$

Application to `PLONK`: The `PLONK` proving system [GWC19] allows generating proofs of knowledge for assignments to fan-in two arithmetic circuits with a universal and updatable SRS (see the paragraph on this topic in Section B.1.1). Most of the prover computation involves committing to several polynomials and opening them at two distinct evaluation points. Plugging in our first PCS to `PLONK` allows saving in proof length and prover work related to the opening proof of the second evaluation point (we do not give full details, but all that is needed is repreating the transformation of Lemma 4.7 in [GWC19] using the PCS of Lemma 16 instead of the PCS used there to obtain the new result).

We compare the `PLONK` scheme when using the [KZG10]-based PCS in [GWC19] and the first PCS of this paper in Table 2. As in [GWC19] we present two versions of `PLONK` where one optimizes fast proving, and the other small proof length.

Table 2: Comparison of `PLONK` efficiency for fan-in two circuit with n gates.

	SRS size	prover group exponentations	proof length	verifier work
[GWC19] (fast)	$n\ \mathbb{G}_1$, $2\ \mathbb{G}_2$	$9n\ \mathbb{G}_1$ exp	$9\ \mathbb{G}_1$, $7\ \mathbb{F}$	$18\ \mathbb{G}_1$, $2\ \mathbf{P}$
This work (fast)	$n\ \mathbb{G}_1$, $3\ \mathbb{G}_2$	$8n\ \mathbb{G}_1$ exp	$8\ \mathbb{G}_1\ 7\ \mathbb{F}$	$18\ \mathbb{G}_1$, $4\ \mathbb{G}_2$, $3\ \mathbf{P}$
[GWC19] (small)	$3n\ \mathbb{G}_1$, $2\ \mathbb{G}_2$	$11n\ \mathbb{G}_1$ exp	$7\ \mathbb{G}_1$, $7\ \mathbb{F}$	$16\ \mathbb{G}_1$, $2\ \mathbf{P}$
This work (small)	$3n\ \mathbb{G}_1$, $3\ \mathbb{G}_2$	$10n\ \mathbb{G}_1$ exp	$6\ \mathbb{G}_1\ 7\ \mathbb{F}$	$16\ \mathbb{G}_1$, $4\ \mathbb{G}_2$, $3\ \mathbf{P}$

`SHPLONK`? Our second PCS does not give interesting tradeoffs for `PLONK` as two evaluation points are not enough for its advantages to "kick in". However, in a scenario where constraints between *more than two* evaluation points are used, e.g. [Dra], the advantages of both of our new schemes will become more prominent. Thus, the PCS of this paper encourage designing constraint systems using multiple SHifts and Permutations over Largange bases for Oecumenical Noninteractive arguments of Knowledge.

B.1 Additonal Preliminaries

We introduce additional terminology and material convenient for our analysis in the algebraic group model.

B.1.1 Terminology and conventions

We assume our field $\mathbb{F}$ is of prime order. We denote by $\mathbb{F}_{<d}[X]$ the set of univariate polynomials over $\mathbb{F}$ of degree smaller than d. In expressions involving both polynomials and constants, we will write $f(X)$ instead of f for to help distinguish the two; but in contexts where it is clear f is a polynomial, we will simply write f for brevity.

We assume all algorithms described receive as an implicit parameter the security parameter λ.

Whenever we use the term "efficient", we mean an algorithm running in time $\mathsf{poly}(\lambda)$. Furthermore, we assume an "object generator" $\mathcal{O}$ that is run with input λ before all protocols, and returns all fields and groups used. Specifically, in our protocol $\mathcal{O}(\lambda) = (\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2, g_t)$ where

- $\mathbb{F}$ is a prime field of super-polynomial size $r = \lambda^{\omega(1)}$.
- $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ are all groups of size r, and e is an efficiently computable non-degenerate pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$.
- g_1, g_2 are uniformly chosen generators such that $e(g_1, g_2) = g_t$.

We usually let the λ parameter be implicit, i.e. write $\mathbb{F}$ instead of $\mathbb{F}(\lambda)$. We write $\mathbb{G}_1$ and $\mathbb{G}_2$ additively. We use the notations $[x]_1 := x \cdot g_1$ and $[x]_2 := x \cdot g_2$.

We often denote by $[n]$ the integers $\{1, \ldots, n\}$. We use the acronym e.w.p for "except with probability"; i.e. e.w.p γ means with probability *at least* $1 - \gamma$.

Universal SRS-based public-coin protocols We describe public-coin (meaning the verifier messages are uniformly chosen) interactive protocols between a prover and verifier; when deriving results for non-interactive protocols, we implicitly assume we can get a proof length equal to the total communication of the prover, using the Fiat-Shamir transform/a random oracle. Using this reduction between interactive and non-interactive protocols, we can refer to the "proof length" of an interactive protocol.

We allow our protocols to have access to a structured reference string (SRS) that can be derived in deterministic $\mathsf{poly}(\lambda)$-time from an "SRS of monomials" of the form $\left\{[x^i]_1\right\}_{a \le i \le b}, \left\{[x^i]_2\right\}_{c \le i \le d}$, for uniform $x \in \mathbb{F}$, and some integers a, b, c, d with absolute value bounded by $\mathsf{poly}(\lambda)$. It then follows from Bowe et al. [BGM17] that the required SRS can be derived in a universal and updatable setup[GKM+18] requiring only one honest participant; in the sense that an adversary controlling all but one of the participants in the setup does not gain more than a $\mathsf{negl}(\lambda)$ advantage in its probability of producing a proof of any statement.

For notational simplicity, we sometimes use the SRS srs as an implicit parameter in protocols, and do not explicitly write it.

B.1.2 Analysis in the AGM model

For security analysis we will use the Algebraic Group Model of Fuchsbauer, Kiltz and Loss [FKL18]. In our protocols, by an *algebraic adversary* $\mathcal{A}$ in an SRS-based protocol we mean a $\mathsf{poly}(\lambda)$-time algorithm which satisfies the following.

- For $i \in \{1,2\}$, whenever $\mathcal{A}$ outputs an element $A \in \mathbb{G}_i$, it also outputs a vector v over $\mathbb{F}$ such that $A =< v, \mathsf{srs_i} >$.

Idealized verifier checks for algebraic adversaries We introduce some terminology to capture the advantage of analysis in the AGM.

First we say our srs *has degree* Q if all elements of $\mathsf{srs_i}$ are of the form $[f(x)]_i$ for $f \in \mathbb{F}_{<Q}[X]$ and uniform $x \in \mathbb{F}$. In the following discussion let us assume we are executing a protocol with a degree Q SRS, and denote by $f_{i,j}$ the corresponding polynomial for the j'th element of $\mathsf{srs_i}$.

Denote by a, b the vectors of $\mathbb{F}$-elements whose encodings in $\mathbb{G}_1, \mathbb{G}_2$ an algebraic adversary $\mathcal{A}$ outputs during a protocol execution; e.g., the j'th $\mathbb{G}_1$ element output by $\mathcal{A}$ is $[a_j]_1$.

By a "real pairing check" we mean a check of the form

$$(a \cdot T_1) \cdot (T_2 \cdot b) = 0$$

for some matrices T_1, T_2 over $\mathbb{F}$. Note that such a check can indeed be done efficiently given the encoded elements and the pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$.

Given such a "real pairing check", and the adversary $\mathcal{A}$ and protocol execution during which the elements were output, define the corresponding "ideal check" as follows. Since $\mathcal{A}$ is algebraic when he outputs $[a_j]_i$ he also outputs a vector v such that, from linearity, $a_j = \sum v_\ell f_{i,\ell}(x) = R_{i,j}(x)$ for $R_{i,j}(X) := \sum v_\ell f_{i,\ell}(X)$. Denote, for $i \in \{1,2\}$ the vector of polynomials $R_i = (R_{i,j})_j$. The corresponding ideal check, checks as a polynomial identity whether

$$(R_1 \cdot T_1) \cdot (T_2 \cdot R_2) \equiv 0$$

The following lemma is inspired by [FKL18]'s analysis of [Gro16b], and tells us that for soundness analysis against algebraic adversaries it suffices to look at ideal checks. Before stating the lemma we define the Q-DLOG assumption similarly to [FKL18].

Definition 11. *Fix an integer Q. The Q-*DLOG assumption for $(\mathbb{G}_1, \mathbb{G}_2)$ *states that given*

$$[1]_1, [x]_1, \ldots, \left[x^Q\right]_1, [1]_2, [x]_2, \ldots, \left[x^Q\right]_2$$

for uniformly chosen $x \in \mathbb{F}$, the probability of an efficient $\mathcal{A}$ outputting x is $\mathsf{negl}(\lambda)$.

The following lemma is proved in [GWC19]-based on the arguments of [FKL18].

Lemma 12. *Assume the Q-DLOG for $(\mathbb{G}_1, \mathbb{G}_2)$. Given an algebraic adversary $\mathcal{A}$ participating in a protocol with a degree Q SRS, the probability of any real pairing check passing is larger by at most an additive* $\mathsf{negl}(\lambda)$ *factor than the probability the corresponding ideal check holds.*

Knowlege soundness in the Algebraic Group Model We say a protocol $\mathscr{P}$ between a prover $\mathbf{P}$ and verifier $\mathbf{V}$ for a relation $\mathcal{R}$ has *Knowledge Soundness in the Algebraic Group Model* if there exists an efficient E such that the probability of any algebraic adversary $\mathcal{A}$ winning the following game is $\mathsf{negl}(\lambda)$.

1. $\mathcal{A}$ chooses input x and plays the role of $\mathbf{P}$ in $\mathscr{P}$ with input x.

2. E given access to all of $\mathcal{A}$'s messages during the protocol (including the coefficients of the linear combinations) outputs ω.

3. $\mathcal{A}$ wins if

   (a) $\mathbf{V}$ outputs acc at the end of the protocol, and

   (b) $(\mathsf{x}, \omega) \notin \mathcal{R}$.

B.1.3 Polynomial commitment schemes in the algebraic group model

For a simple presentation of our schemes in the context of KZG, it will be convenient to define polynomial commitment schemes similarly to [GWC19]. Specifically

- We define the open procedure analogously to the Eval procedure from the definition in Section 2.2, and it directly deals with the batch evaluation setting.

- We define knowledge soundness specfically against "algebraic" adversaries.

On advantage of using the same definition as in [GWC19], is that it enables directly pluggin in our batch evaluation result into the machinery of [GWC19] to obtain the improved prover time stated above. In the context of multiple points, it will be more convenient to assume the alleged evaluations of a polynomial f on a set $S \subset \mathbb{F}$ are given as a *polynomial* $r \in \mathbb{F}_{<|S|}[X]$ with $r(z) = f(z)$ for each $z \in S$. Under this convention, the condition that the evaluations are correct; i.e. $r(z) = f(z)$ for each $z \in S$, is equivalent to $f(X) - r(X)$ being divisble by $Z_S(X)$; where $Z_S(X) := \prod_{z \in S}(X - z)$.

Definition 13. *A polynomial commitment scheme is a triplet* $\mathscr{S} = (\mathsf{gen}, \mathsf{com}, \mathsf{open})$ *such that*

- $\mathsf{gen}(d)$ - *is a randomized algorithm that given positive integer d outputs a structured reference string (SRS)* srs*.*

- $\mathsf{com}(f, \mathsf{srs})$ - *is an algorithm that given a polynomial* $f \in \mathbb{F}_{<d}[X]$ *and an output* srs *of* $\mathsf{gen}(d)$ *returns a commitment* cm *to* f*.*

- open *is a public coin protocol between parties* P_{PC} *and* V_{PC}*.* P_{PC} *is given* $f_1, \ldots, f_k \in \mathbb{F}_{<d}[X]$*.* P_{PC} *and* V_{PC} *are both given*

   1. *positive integers* $d, t = \mathsf{poly}(\lambda)$*,*
   2. $\mathsf{srs} = \mathsf{gen}(d)$*,*
   3. *a subset* $T = \{z_1, \ldots, z_t\} \subset \mathbb{F}$*,*
   4. *subsets* $S_1, \ldots, S_k \subset T$*,*
   5. $\mathsf{cm}_1, \ldots, \mathsf{cm}_k$ *- the alleged commitments to* $f_1, \ldots, f_k$*,*
   6. $\left\{r_i \in \mathbb{F}_{<|S_i|}[X]\right\}_{i \in [k]}$ *- the polynomials describing the alleged correct openings, i.e. having* $r_i(z) = f_i(z)$ *for each* $i \in [k], z \in S_i$*.*

   At the end of the protocol V_{PC} *outputs* acc *or* rej*; such that*

   – ***Completeness:*** *Fix any* $k, t = \mathsf{poly}(\lambda)$*,* $T = \{z_1, \ldots, z_t\} \subset \mathbb{F}$*,* $S_1, \ldots, S_k \subset T$*,* $f_1, \ldots, f_k \in \mathbb{F}_{<d}[X], \left\{r_i \in \mathbb{F}_{<|S_i|}[X]\right\}_{i \in [k]}$*. Suppose that for each* $i \in [k]$*,* $\mathsf{cm}_i = \mathsf{com}(f_i, \mathsf{srs})$*, and for each* $i \in [k]$ *we have* $Z_{S_i} | (f_i - r_i)$*. Then if* P_{PC} *follows* open *correctly with these values,* V_{PC} *outputs* acc *with probability one.*

– ***Knowledge soundness in the algebraic group model:*** *There exists an efficient E such that for any algebraic adversary $\mathcal{A}$ and any choice of $d = \mathsf{poly}(\lambda)$ the probability of $\mathcal{A}$ winning the following game is $\mathsf{negl}(\lambda)$ over the randomness of $\mathcal{A}$, V_{PC} and gen.*

1. *Given d and $\mathsf{srs} = \mathsf{gen}(d)$, $\mathcal{A}$ outputs $\mathsf{cm}_1, \ldots, \mathsf{cm}_k \in \mathbb{G}_1$.*
2. *E, given access to the messages of $\mathcal{A}$ during the previous step, outputs $f_1, \ldots, f_k \in \mathbb{F}_{<d}[X]$.*
3. *$\mathcal{A}$ outputs $T = \{z_1, \ldots, z_t\} \subset \mathbb{F}$, $S_1, \ldots, S_k \subset T$, $\left\{r_i \in \mathbb{F}_{<|S_i|}[X]\right\}_{i\in[k]}$.*
4. *$\mathcal{A}$ takes the part of P_{PC} in the protocol open with the inputs $\mathsf{cm}_1, \ldots, \mathsf{cm}_k, T, S_1, \ldots, S_k$, $\{r_i\}$.*
5. *$\mathcal{A}$ wins if*
   * *V_{PC} outputs acc at the end of the protocol.*
   * *For some $i \in [k]$, $Z_{S_i} \nmid (f_i - r_i)$.*

B.2 Our first scheme

We first state the following straightforward claim that will allow us to efficiently "uniformize" checks on different evaluation points.

Claim 14. *Fix subsets $S \subset T \subset \mathbb{F}$, and a polynomial $g \in \mathbb{F}_{<d}[X]$. Then $Z_S(X)$ divides $g(X)$ if and only if $Z_T(X)$ divides $Z_{T\setminus S}(X) \cdot g(X)$.*

We also use the following claim, which is part of Claim 4.6 in [GWC19] where a proof of it can be found.

Claim 15. *Fix $F_1, \ldots, F_k \in \mathbb{F}_{<n}[X]$. Fix $Z \in \mathbb{F}_{<n}[X]$ that decomposes to distinct linear factors over $\mathbb{F}$. Suppose that for some $i \in [k]$, $Z \nmid F_i$. Then, e.w.p $k/|\mathbb{F}|$ over uniform $\gamma \in \mathbb{F}$, Z does not divide*

$$G := \sum_{j=1}^{k} \gamma^{j-1} \cdot F_j.$$

We present our first PCS.

1. $\mathsf{gen}(d)$ - choose uniform $x \in \mathbb{F}$. Output $\mathsf{srs} = \left([1]_1, [x]_1, \ldots, \left[x^{d-1}\right]_1, [1]_2, [x]_2, \ldots, \left[x^t\right]_2\right)$.
2. $\mathsf{com}(f, \mathsf{srs}) := [f(x)]_1$.
3. $\mathsf{open}\left(d, t, \{\mathsf{cm}_i\}_{i\in[k]}, T = \{z_1, \ldots, z_t\} \subset \mathbb{F}, \{S_i \subset T\}_{i\in[k]}, \{r_i\}_{i\in[k]}\right)$:
   (a) V_{PC} sends a random $\gamma \in \mathbb{F}$.
   (b) P_{PC} computes the polynomial
   $$h(X) := \sum_{i\in[k]} \gamma^{i-1} \cdot \frac{f_i(X) - r_i(X)}{Z_{S_i(X)}}$$
   and using srs computes and sends $W := [h(x)]_1$.
   (c) V_{PC} computes for each $i \in [k]$, $Z_i := \left[Z_{T\setminus S_i}(x)\right]_2$.

(d) V_{PC} computes

$$F := \prod_{i\in[k]} e\left(\gamma^{i-1}\cdot(\mathsf{cm}_i - [r_i(x)]_1), Z_i\right).$$

(e) V_{PC} outputs acc if and only if

$$F = e(W, [Z_T(x)]_2).$$

We argue knowledge soundness for the above protocol. More precisely, we argue the existence of an efficient E such that an algebraic adversary $\mathcal{A}$ can only win the KS game described in Section B.1.3 w.p. $\mathsf{negl}(\lambda)$.

Let $\mathcal{A}$ be such an algebraic adversary.

$\mathcal{A}$ begins by outputting $\mathsf{cm}_1, \ldots, \mathsf{cm}_k \in \mathbb{G}_1$. Each cm_i is a linear combination $\sum_{j=0}^{d-1} a_{i,j}\left[x^j\right]_1$. E, who is given the coefficients $\{a_{i,j}\}$, simply outputs the polynomials

$$f_i(X) := \sum_{j=0}^{d-1} a_{i,j}\cdot X^j.$$

$\mathcal{A}$ now outputs $T = \{z_1, \ldots, z_t\} \subset \mathbb{F}, \{S_i \subset T\}_{i\in[k]}, \{r_i\}_{i\in[k]}$. Assume that for some $i^* \in [k]$,, we have $Z_{S_{i^*}} \nmid (f_{i^*} - r_{i^*})$. We show that for any strategy of $\mathcal{A}$ from this point, $\mathrm{V}_{\mathsf{poly}}$ outputs acc w.p. $\mathsf{negl}(\lambda)$.

In the first step of open, $\mathrm{V}_{\mathsf{poly}}$ chooses a random $\gamma \in \mathbb{F}$. Let

$$f(X) := \sum_{i\in[t]} \gamma^{i-1}\cdot Z_{T\setminus S_i}(X)\cdot(f_i(X) - r_i(X)).$$

We know from Claim 14 that $F_{i^*} := Z_{T\setminus S_{i^*}}\cdot(f_{i^*} - r_{i^*})$ is not divisible by Z_T. Thus, using Claim 15, we know that e.w.p $k/|\mathbb{F}|$ over γ, f is not divisible by Z_T. Now $\mathcal{A}$ outputs $W = [H(x)]_1$ for some $H \in \mathbb{F}_{<d}[X]$. According to Lemma 12, it suffices to upper bound the probability that the ideal check corresponding to the real pairing check in the protocol passes. It has the form

$$f(X) \equiv H(X)Z_T(X).$$

The check passing implies that $f(X)$ is divisible by Z_T. Thus the ideal check can only pass w.p. $k/|\mathbb{F}| = \mathsf{negl}(\lambda)$ over the randomness of $\mathrm{V}_{\mathsf{poly}}$, which implies the same thing for the real check according to Lemma 12.

We summarize the efficiency properties of the scheme.

Lemma 16. *There is a PCS $\mathscr{S} = (\mathsf{gen}, \mathsf{com}, \mathsf{open})$ such that*

1. *For positive integer d, $\mathsf{srs} = \mathsf{gen}(d)$ consists of d $\mathbb{G}_1$ elements and $t+1$ $\mathbb{G}_2$ elements.*
2. *For integer $n \le d$ and $f \in \mathbb{F}_{<n}[X]$, computing $\mathsf{com}(f, \mathsf{srs})$ requires n $\mathbb{G}_1$-exponentiations.*
3. *Given $T := (z_1, \ldots, z_t) \in \mathbb{F}^t, f_1, \ldots, f_k \in \mathbb{F}_{<d}[X], \{S_i\}_{i\in[k]}$, denote by k^* the number of distinct subsets $\{S_1^*, \ldots, S_{k^*}^*\}$ in $\{S_i\}$; and let $K := t + \sum_{i\in[k^*]}(t - |S_i^*|)$. and denote $n := \max\{\deg(f_i)\}_{i\in[k]}$. Let $\mathsf{cm}_i = \mathsf{com}(f_i)$. Then $\mathsf{open}(\{\mathsf{cm}_i\}, \{f_i\}, T, \{S_i \subset T\}, \{r_i\}, \mathsf{srs})$ requires*

(a) *A single* $\mathbb{G}_1$ *element to be passed from* $\mathrm{P}_{\mathsf{poly}}$ *to* $\mathrm{V}_{\mathsf{poly}}$.

(b) *At most* n $\mathbb{G}_1$*-exponentiations of* $\mathrm{P}_{\mathsf{poly}}$.

(c) $k-1$ $\mathbb{G}_1$*-exponentiations,* K $\mathbb{G}_2$*-exponentiations and* k^*+1 *pairings of* $\mathrm{V}_{\mathsf{poly}}$.

B.3 Reducing verifier operations at the expense of proof length

We describe a variant of the scheme of Section B.2 where we eliminate the verifier's $\mathbb{G}_2$ operations and reduce the number of pairings to two. This comes at the cost of an extra $\mathbb{G}_1$ element sent by the prover. Roughly speaking, while in Section B.2 V_{PC} used $\mathbb{G}_2$ and pairing operations to compute the evaluation of a certain polynomial f encoded in the target group $\mathbb{G}_t$, in this protocol P_{PC} gives V_{PC} this evaluation encoded in $\mathbb{G}_1$, accompanied by a proof that it is correct. As mentioned, this is simply an instantiation of our general private aggregation scheme in the context of the KZG PCS. However, we will take advantage of the algebraic group model to simplify the security proof. We first describe the PCS, and end the section by stating the obtained final result.

1. $\mathsf{gen}(d)$ outputs $\mathsf{srs} = ([1]_1, [x]_1, \ldots, \left[x^{d-1}\right]_1, [1]_2, [x]_2)$ for a random $x \in \mathbb{F}$.
2. $\mathsf{com}(f_i) = [f_i(x)]_1$.
3. We describe the open procedure twice below. First, in a way that will be convenient for the security analysis, and later in an equivalent more concise way that also optimizes verifier operations, .e.g. moves operations from $\mathbb{G}_2$ into $\mathbb{G}_1$ when possible.

$\underline{\mathsf{open}(\{\mathsf{cm}_i\}, T, \{S_i\}, \{r_i\})}$:

1. V_{PC} sends random $\gamma \in \mathbb{F}$.
2. P_{PC} computes the polynomial
$$f(X) := \sum_{i\in[k]} \gamma^{i-1} \cdot Z_{T\setminus S_i}(X) \cdot (f_i(X) - r_i(X)).$$
Recall that f is divisible by Z_T according to Claim 15, and define $h(X) := f(X)/Z_T(X)$. Using srs, P_{PC} computes and sends $W := [h(x)]_1$.
3. V_{PC} sends random $z \in \mathbb{F}$.
4. P_{PC} computes the polynomial
$$L(X) := f_z(X) - Z_T(z) \cdot h(X),$$
where
$$f_z(X) := \sum_{i\in[k]} \gamma^{i-1} \cdot Z_{T\setminus S_i}(z) \cdot (f_i(X) - r_i(z))$$
Note that $L(z) = f(z) - Z_T(z)\cdot h(z) = 0$, and thus $(X-z)$ divides L. P_{PC} sends $W' := \left[\frac{L(x)}{x-z}\right]_1$.
5. V_{PC} computes:
$$F := \sum_{i\in[k]} \gamma^{i-1} \cdot Z_{T\setminus S_i}(z) \cdot (\mathsf{cm}_i - [r_i(z)]_1) - Z_T(z) \cdot W$$

6. V_{PC} outputs acc if and only if

$$e(F, [1]_2) = e(W', [x - z]_2).$$

We argue knowledge soundness for the above protocol. More precisely, we argue the existence of an efficient E such that an algebraic adversary $\mathcal{A}$ can only win the KS game w.p. $\mathsf{negl}(\lambda)$. The proof begins identically to the previous one.

Let $\mathcal{A}$ be such an algebraic adversary.

$\mathcal{A}$ begins by outputting $\mathsf{cm}_1, \ldots, \mathsf{cm}_k \in \mathbb{G}_1$. Each cm_i is a linear combination $\sum_{j=0}^{d-1} a_{i,j} \left[x^j\right]_1$. E, who is given the coefficients $\{a_{i,j}\}$, simply outputs the polynomials

$$f_i(X) := \sum_{j=0}^{d-1} a_{i,j} \cdot X^j.$$

$\mathcal{A}$ now outputs $T = \{z_1, \ldots, z_t\} \subset \mathbb{F}, \{S_i \subset T\}_{i \in [k]}, \{r_i\}_{i \in [k]}$. Assume that for some $i^* \in [k]$, we have $Z_{S_{i^*}} \nmid (f_{i^*} - r_{i^*})$. We show that for any strategy of $\mathcal{A}$ from this point, $\mathrm{V}_{\mathsf{poly}}$ outputs acc w.p. $\mathsf{negl}(\lambda)$.

In the first step of open, $\mathrm{V}_{\mathsf{poly}}$ chooses a random $\gamma \in \mathbb{F}$. Let

$$f(X) := \sum_{i \in [k]} \gamma^{i-1} \cdot Z_{T \setminus S_i} \cdot (f_i(X) - r_i(X)).$$

We know from Claim 14 that $F_{i^*} := Z_{T \setminus S_{i^*}}(f_{i^*} - r_{i^*})$ is not divisible by Z_T. Thus, using Claim 15, we know that e.w.p $k/|\mathbb{F}|$ over γ, f is not divisble by Z_T. Assume we are in this case. Now $\mathcal{A}$ outputs $W = [H(x)]_1$ for some $H \in \mathbb{F}_{<d}[X]$, followed by V_{PC} sending uniform $z \in \mathbb{F}$. Since we are in the case that f is not divisble by Z_T, we know there are at most $2d$ values $z \in \mathbb{F}$ such that $f(z) = H(z) \cdot Z_T(z)$; and thus z chosen by V_{PC} is of this form only w.p. $\mathsf{negl}(\lambda)$. Assume we are in the case that z sent by V_{PC} is not of this form. P_{PC} now outputs $W' = [H'(x)]_1$ for some $H' \in \mathbb{F}_{<d}[X]$. According to Lemma 12, it suffices to upper bound the probability that the ideal check corresponding to the real pairing check in step 6 passes. Denoting

$$L'(X) := \sum_{i \in [k]} \gamma^{i-1} Z_{T \setminus S_i}(z) \cdot (f_i(X) - r_i(z)) - Z_T(z) \cdot H(X),$$

the ideal check has the form

$$L'(X) \equiv H'(X) \cdot (X - z),$$

and thus can pass for some $H' \in \mathbb{F}_{<d}[X]$ only if L' is divisible by $(X - z)$, which means $L'(z) = 0$. However

$$L'(z) = \sum_{i \in [k]} \gamma^{i-1} Z_{T \setminus S_i}(z) \cdot (f_i(z) - r_i(z)) - Z_T(z) \cdot H(z) = f(z) - Z_T(z) \cdot H(z),$$

and we are in the case where $f(z) \neq Z_T(z) \cdot H(z)$. In summary, the ideal check can only pass w.p. $\mathsf{negl}(\lambda)$ over the randomness of V_{PC}, which implies the same thing for the real check according to Lemma 12.

B.4 The open procedure, "cleaned up" and optimized

$\underline{\mathsf{open}(\{\mathsf{com}(f_i)\}, \{S_i\}, \{r_i\})}$:

1. V_{PC} sends a random challenge $\gamma \in \mathbb{F}$.
2. P_{PC} sends $W := [(f/Z_T)(x)]_1$ where

$$f := \sum_{i \in [k]} \gamma^{i-1} \cdot Z_{T \setminus S_i}(f_i - r_i).$$

3. V_{PC} sends a random evaluation point $z \in \mathbb{F}$
4. P_{PC} sends $W' := [(L(x)/(x-z)]_1$ where

$$L := \sum_{i \in [k]} \gamma^{i-1} Z_{T \setminus S_i}(z) \cdot (f_i - r_i(z)) - Z_T(z) \cdot (f/Z_T).$$

5. V_{PC} outputs acc iff $e(F + zW', [1]_2) = e(W', [x]_2)$, where

$$F := \sum_{i \in [k]} \gamma^{i-1} Z_{T \setminus S_i}(z) \cdot \mathsf{cm}_i - \left[\sum_{i \in [k]} \gamma^{i-1} Z_{T \setminus S_i}(z) r_i(z) \right]_1 - Z_T(z) W.$$

From the description and analysis we obtain

Lemma 17. *There is a PCS* $\mathscr{S} = (\mathsf{gen}, \mathsf{com}, \mathsf{open})$ *such that*

1. *For positive integer d,* $\mathsf{srs} = \mathsf{gen}(d)$ *consists of d $\mathbb{G}_1$ elements and 2 $\mathbb{G}_2$ elements.*
2. *For integer $n \le d$ and $f \in \mathbb{F}_{<n}[X]$, computing* $\mathsf{com}(f, \mathsf{srs})$ *requires n $\mathbb{G}_1$-exponentiations.*
3. *Given $T := (z_1, \ldots, z_t) \in \mathbb{F}^t, f_1, \ldots, f_k \in \mathbb{F}_{<d}[X], \{S_i\}_{i \in [k]}$ and denote $n := \max\{\deg(f_i)\}_{i \in [k]}$. Let $\mathsf{cm}_i = \mathsf{com}(f_i)$. Then* $\mathsf{open}(\{\mathsf{cm}_i\}, \{f_i\}, T, \{S_i \subset T\}, \{r_i\}, \mathsf{srs})$ *requires*
   (a) *2 $\mathbb{G}_1$ elements sent from P_{PC} to V_{PC}.*
   (b) *at most $2n+1$ $\mathbb{G}_1$-exponentiations of P_{PC}.*
   (c) *$k+3$ $\mathbb{G}_1$-exponentiations and 2 pairings of V_{PC}.*