Lunar: a Toolbox for More Efficient Universal and Updatable zkSNARKs and Commit-and-Prove Extensions

Matteo Campanelli[1], Antonio Faonio[2], Dario Fiore[3], Anaïs Querol[3,4], and Hadrián Rodríguez[3]

[1] Aarhus University, Aarhus, Denmark
matteo@cs.au.dk
[2] EURECOM, Sophia Antipolis, France
antonio.faonio@eurecom.fr
[3] IMDEA Software Institute, Madrid, Spain
{dario.fiore,anais.querol,hadrian.rodriguez}@imdea.org
[4] Universidad Politécnica de Madrid, Madrid, Spain

Abstract. We study how to construct zkSNARKs whose SRS is *universal* and *updatable*, i.e., valid for all relations within a size-bound and to which a dynamic set of participants can indefinitely add secret randomness. Our focus is: efficient universal updatable zkSNARKs with linear-size SRS and their commit-and-prove variants. We both introduce new formal frameworks and techniques, as well as systematize existing ones. We achieve a collection of zkSNARKs with different tradeoffs. One of our schemes achieves the smallest proof size and proving time compared to the state of art for proofs for arithmetic circuits. The language supported by this scheme is a variant of R1CS that we introduce, called R1CS-lite. Another of our constructions directly supports standard R1CS and achieves the fastest proving time for this type of constraints.
These results stem from different contributions: (1) a new algebraically-flavored variant of IOPs that we call *Polynomial Holographic IOPs* (PHPs); (2) a new compiler that combines our PHPs with *commit-and-prove zk-SNARKs* (CP-SNARKs) for committed polynomials; (3) pairing-based realizations of these CP-SNARKs for polynomials; (4) constructions of PHPs for R1CS and R1CS-lite. Finally, we extend the compiler in item (2) to yield commit-and-prove universal zkSNARKs.

Keywords: zkSNARK · universal SRS · polynomial commitments · IOP

1 Introduction

A zero-knowledge proof system [31] allows a prover to convince a verifier that a non-deterministic computation accepts without revealing more information than its input. In the last decade, there has been growing interest in zero-knowledge proof systems that additionally are *succinct* and *non interactive* [12, 29, 40, 46], dubbed zkSNARKs. These are computationally-sound proof systems (*arguments*) that are *succinct*, in that their proofs are short and efficient to verify: the proof size and verification time should be constant or polylogarithmic in the length of the non-deterministic witness. In circuit-based arguments for general

computations the verifier must at least read the statement to be proven which includes both the description of the computation (i.e., the circuit) and its input (i.e., public input). But this is not succinct; by reading the whole circuit, the verifier runs linearly in the size of the computation. *Preprocessing zkSNARKs* try and work around this problem [13, 28, 32, 44]. Here the verifier generates a *structured reference string* (SRS) that depends on a certain circuit C; it does this *once and for all*. This SRS can be used later to verify an unbounded number of proofs for the computation of C. This is a succinct system: while the cost of SRS generation does depend on $|C|$, proof verification does not have to.

Works on subversion-resistance show that CRS can be generated by a verifier with no impact on security [1, 3, 24]. But contexts with many verifiers, e.g. blockchains, require a trusted party. Solutions that mitigate this problem (e.g. MPC secure against dishonest majority [7]) are still expensive and often impractical as they should be carried out for each single C. To address this problem, Groth et al. [34] introduced the model of *universal and updatable SRS*. An SRS is *universal* if it can be used to generate and verify proofs for all circuits of some bounded size; it is *updatable* if any user can add randomness to it and a sequence of updates makes it secure if at least one user acted honestly. They proposed the first such zkSNARK, but their scheme requires an SRS of size *quadratic* in the number of multiplication gates of the supported arithmetic circuits (and similar quadratic update/verification time).

Recent works [18, 19, 21, 27, 45, 53] have improved on this result obtaining universal and updatable SRS whose size is *linear* in the largest supported circuit. In particular, the current Marlin [19] and PLONK [27] proof systems achieve a proving time concretely faster than that of Sonic [45] while retaining constant-size proofs ([18, 21, 53] have instead polylogarithmic-size proofs). We also mention the very recent works of Bünz, Fisch and Szepieniec [17], and Chiesa, Ojha and Spooner [20] that proposed zkSNARKs in the uniform random string (URS) model, that is implicitly universal and updatable; their constructions have a short URS and poly-logarithmic-size proofs. Yet another universal zkSNARK construction is that in [41] which, despite its proofs of 4 group elements and comparable proving time, has an SRS which is not updatable.

Many of these efficient constructions (and the ones in this work) follow a similar blueprint to build zkSNARKs, which we now overview.

The current landscape of zkSNARKs with universal SRS. A known modular paradigm to build efficient cryptographic arguments [36, 37] works in two distinct steps. First construct an information-theoretic protocol in an abstract model, e.g., interactive proofs [31], standard or linear PCPs [13], IOPs [9, 48]. Then apply a compiler that, taking an abstract protocol as input, transforms it into an efficient computationally sound argument via a cryptographic primitive. This approach has been successfully adopted to construct zkSNARKs with universal SRS in the recent works [17, 19, 27], in which the information theoretic object is an algebraically-flavored variant of Interactive Oracle Proofs (IOPs), while the cryptographic primitive are *polynomial commitments* [39]. Through polynomial commitments, a prover can compress a polynomial p (as a message

much shorter than all its concatenated coefficients) and can later open the commitment at evaluations of p, namely to convince a verifier that $y = p(x)$ for public points x and y. In these IOP abstractions—called *algebraic holographic proofs* (AHP) in [19] and *polynomial IOPs*[1] in [17]—a prover and a verifier interact, one providing oracle access to a set of polynomials and the other sending random challenges (if public-coin). At the end of the protocol the verifier asks for evaluations of these polynomials and decides to accept or reject based on the responses. The *idealized low-degree protocols* (ILDPs) abstraction of [27] proceeds similarly except that in the end the verifier asks to verify a set of polynomial identities over the oracles sent by the prover (which can be tested via evaluation on random points). To build a zkSNARK with universal SRS starting from AHPs/ILDPs we let the prover commit to the polynomials obtained from the AHP/ILDP prover, and then use the opening feature of polynomial commitments to respond to the evaluation queries in a sound way. As we detail later, our contribution revisits the aforementioned blueprint to construct universal zkSNARKs.

1.1 Our Contribution

In this work we propose Lunar, a *family* of new preprocessing zkSNARKs in the universal and updatable SRS model that have constant-size proofs and that improve on previous work [19, 27, 45] as to proof size and prover running time.

In Table 1, we present a detailed efficiency comparison between prior work and the best representatives of our schemes, when using arithmetic circuit satisfiability as common benchmark. LunarLite has the smallest proof size (384 bytes over curve BN128; 544 bytes over BLS12-381)[2] and the lowest proving time compared to the state of art of universal zkSNARKs with constant-size proofs for arithmetic circuits. As we explain later, LunarLite uses a new arithmetization of arithmetic circuit satisfiability that we call R1CS-lite, quite similar to rank-1 constraint systems (R1CS). A precise comparison to PLONK depends on the circuit structure and how the number m of nonzero entries of R1CS-lite matrices depends on the number a of addition gates[3]; for instance, PLONK is faster for circuits with only multiplication gates, but LunarLite is faster when $m \leq 3a$.

If we focus the comparison on solutions that directly support R1CS (of which Marlin [19] is the most performant among prior work), our scheme Lunar1cs (fast & short) offers the smallest SRS, the smallest proof and the fastest prover. This comes at the price of higher constants for the size of the (specialized) verification key and verification time[4]. Lunar1cs (short vk) offers a tradeoff: it has smaller verification key and faster verification time, but slightly larger proofs, $3\times$ larger SRS, and $5m$ more $\mathbb{G}_1$-exponentiations at proving time than Lunar1cs (fast & short). Even with this tradeoff, Lunar1cs (short vk) outperforms Marlin in all these measures. We implemented Lunar's building blocks and we confirm our observations experimentally (check full version).

[1] Hereinafter we use AHP/PIOPs interchangeably as they are almost the same notion.
[2] BN128 is 100-bits-secure while BLS12-381 has 128-bits-security.
[3] Applying [14] PLONK's proving time drops to $8n + 8a$, but our analysis still holds.
[4] In practice this overhead is negligible. Lunar1cs (fast & short) takes 7 pairings to verify ($\approx 35ms$); faster schemes, including some from this work, take 2 ($\approx 10ms$).

Table 1. Efficiency of universal and updatable *practical* zkSNARKs for arithmetic circuit satisfiability with $O(1)$ proofs. n: number of multiplication gates; a: number of addition gates; $m \geq n$: number of nonzero entries in R1CS(-lite) matrices encoding the circuit; N, N^*, A and M: largest supported values for $n, a+m, a$ and m respectively.

zkSNARK		**size**				**time**			
		$\|\mathsf{srs}\|$	$\|\mathsf{ek}_\mathsf{R}\|$	$\|\mathsf{vk}_\mathsf{R}\|$	$\|\pi\|$	KeyGen	Derive	Prove	Verify
Sonic [45]	$\mathbb{G}_1$	$4N$	$36n$	—	20	$4N$	$36n$	$273n$	7 pairings
	$\mathbb{G}_2$	$4N$	—	3	—	$4N$	—	—	
	$\mathbb{F}$	—	—	—	16	—	$O(m \log m)$	$O(m \log m)$	$O(\ell+\log m)$
Marlin [19]	$\mathbb{G}_1$	$3M$	$3m$	12	13	$3M$	$12m$	$14n+8m$	2 pairings
	$\mathbb{G}_2$	2	—	2	—	—	—	—	
	$\mathbb{F}$	—	—	—	8	—	$O(m \log m)$	$O(m \log m)$	$O(\ell+\log m)$
(small proof)	$\mathbb{G}_1$	$3N^*$	$3n+3a$	8	7	$3N^*$	$8n+8a$	$11n+11a$	
(fast prover)		N^*	$n+a$	8	9	N^*	$8n+8a$	$9n+9a$	
PLONK	$\mathbb{G}_2$	1	—	1	—	1	—	—	2 pairings
[27]	$\mathbb{F}$	—	—	—	7	—	$O((n+a)\log(n+a))$	$O((n+a)\log(n+a))$	$O(\ell+\log(n+a))$
LunarLite [this work]	$\mathbb{G}_1$	M	m	—	10	M	—	$8n+3m$	7 pairings
	$\mathbb{G}_2$	M	—	27	—	M	$24m$	—	
	$\mathbb{F}$	—	—	—	2	—	$O(m \log m)$	$O(m \log m)$	$O(\ell+\log m)$
Lunar1cs (fast & short)	$\mathbb{G}_1$	M	m	—	11	M	—	$9n+3m$	7 pairings
	$\mathbb{G}_2$	M	—	60	—	M	$57m$	—	
	$\mathbb{F}$	—	—	—	2	—	$O(m \log m)$	$O(m \log m)$	$O(\ell+\log m)$
Lunar1cs (short vk)	$\mathbb{G}_1$	$3M$	$3m$	12	12	$3M$	$12m$	$9n+8m$	2 pairings
	$\mathbb{G}_2$	1	—	1	—	1	—	—	
	$\mathbb{F}$	—	—	—	5	—	$O(m \log m)$	$O(m \log m)$	$O(\ell+\log m)$

Our main contribution to achieve this result is *to revisit the aforementioned blueprint to construct universal zkSNARKs* by proposing: (1) a new algebraically-flavored variant of IOPs, *Polynomial Holographic IOPs* (PHPs), and (2) a new compiler that builds universal zkSNARKs by using our PHPs together with *commit-and-prove zkSNARKs (CP-SNARKs) [18] for committed polynomials*. Additional contributions include: (3) pairing-based realizations of these CP-SNARKs for polynomials, (4) constructions of PHPs for both R1CS and a novel simplified variant of it, (5) a variant of the compiler (2) that yields a commit-and-prove universal zkSNARK. The latter is the first general compiler from (algebraic) IOPs to commit-and-prove zkSNARKs. A CP-SNARK permits to verify a proof through a commitment to an input (rather than an input in the clear) that, crucially, we can reuse among proofs[5]. Below we detail our contributions.

Polynomial Holographic IOPs (PHPs). Our PHPs generalize AHPs[6] as well as ILDPs. A PHP consists of an interaction between a verifier and a prover sending oracle polynomials, followed by a decision phase in which the verifier outputs a set of polynomial identities to be checked on the prover's polynomials (such as $a(X)b(X) - z \cdot c(X) \stackrel{?}{=} 0$, for oracle polynomials a, b, c and some scalar

[5] We compose CP-SNARKs as gadgets to modularly build complex schemes; as studied recently [18, 54], they are useful to prove properties of committed values [11, 35].

[6] PHPs generalize AHPs where the verifier is "algebraic", including all schemes in [19].

z), as well as a set of degree tests (e.g. $\mathsf{deg}(a(X)) < D$). The PHP model is close to ILDPs, but the two differ with respect to zero-knowledge formalizations: while ILDPs lack one altogether, we introduce and formalize a fine-grained notion of zero-knowledge—called $(\mathsf{b}_1, \ldots, \mathsf{b}_n)$-bounded zero-knowledge—where the verifier may learn up to b_i evaluations of the i-th oracle polynomial. When compared to AHPs, PHP has, again, a more granular notion of zero-knowledge, as well as verification queries that are more expressive than mere polynomial evaluations.

As we shall discuss next, these two properties of PHPs—expressive verifier's queries and a highly flexible zero-knowledge notion—naturally capture more (and more efficient) strategies when compiling into a cryptographic argument (e.g., we can weaken the required hiding property of the polynomial commitments and the zero-knowledge of the CP-SNARKs used in our compiler).

From PHPs to zkSNARKs through another model of polynomial commitments. We describe how to compile a (public-coin) PHP into a zkSNARK. For AHPs and ILDPs [19, 27], compilation works by letting the prover use polynomial commitments on the oracles and then open them to the evaluations asked by the verifier. Our approach, though similar, has a key distinction: *a different formalization of polynomial commitments with a modular design.*

Our notion of polynomial commitments is *modular*: rather than seeing them as a monolithic primitive—a tuple of algorithms for both commitment and proofs—we split them into two parts, i.e., a regular commitment scheme with polynomials as message space, and a collection of commit-and-prove SNARKs (CP-SNARKs) for proving relations over committed polynomials. We find several advantages in this approach.

As already argued in prior work on modular zkSNARKs through the commit-and-prove strategy [11, 18], one benefit of this approach is *separation of concerns*: commitments are required to do one thing independently of the context (committing), whereas what we need to prove about them may depend on *where* we are applying them. For example, we often want to prove evaluation of committed polynomials: given a commitment c and points x, y, prove that $y = p(x)$ and c opens to p. But to compile a PHP (or AHP/ILDP) we also need to be able to prove other properties about them, such as checking degree bounds or testing equations over committed polynomials. Because these properties—and the techniques to prove them—are somehow independent from each other, we argue they should not be bundled under a bloated notion of polynomial commitment. Going one step further in this direction, we formalize commitment extractability as a proof of knowledge of opening of a polynomial commitment. This modular design allows us to describe an abstract compiler that assumes generic CP-SNARKs for the three aforementioned relations—proof of knowledge of opening, degree bounds and polynomial equations—and can yield zkSNARKs with different tradeoffs depending on how we instantiate them.

We also find additional benefits of the modular abstraction. First, a CP-SNARK for testing equations over committed polynomials more faithfully captures the goal of the PHP verifier (as well as the AHP verifier in virtually all known constructions). Second, we can allow for realizations of CP-SNARKs for

equations over polynomials other than the standard one, which reduces the problem of (batched) polynomial evaluations via random point evaluation. As an application, we show a simple scheme for quadratic equations that can even have an empty proof (see below); our most efficient realizations exploit this fact.

From PHPs to zkSNARKs: fine-grained leakage requirements. Our second contribution on the compiler is to *minimize the requirements needed to achieve zero-knowledge*. As we shall discuss later, this allows us to obtain more efficient zkSNARKs. A straightforward compiler from PHPs to zkSNARKs would require *hiding* polynomial commitments and *zero-knowledge* CP-SNARKs; we weaken both requirements. Instead of "fully" hiding commitments, our compiler requires only *somewhat hiding* commitments. This new property guarantees, for each committed polynomial, leakage of at most one evaluation on a random point. Instead of compiling through "full" zero-knowledge CP-SNARKs, our compiler requires only $(\mathsf{b}_1, \ldots, \mathsf{b}_n)$-leaky zero-knowledge CP-SNARKs. This new notion is weaker than zero-knowledge and states that the verifier may learn up to b_i evaluations of the i-th committed polynomial.

We show that by using a somewhat-hiding commitment scheme and a $(\mathsf{b}_1, \ldots, \mathsf{b}_n)$-leaky zero-knowledge CP-SNARK that can prove the checks of the PHP verifier, one can compile a PHP that is $(\mathsf{b}_1 + 1, \ldots, \mathsf{b}_n + 1)$-bounded ZK into a fully-zero-knowledge succinct argument.

Although related ideas were used in constructions in previous works [27], our contribution is to systematically formalize (as well as expand) the properties needed on different fronts: the PHP, the commitment scheme, the CP-SNARKs used as building blocks and the interaction among all these in the compiler.

Pairing-based CP-SNARKs for committed polynomials. We consider the basic commitment scheme for polynomials consisting of giving a "secret-point evaluation in the exponent" [32, 39] and then show CP-SNARKs for various relations over that same commitment scheme. In particular, by using techniques from previous works [19, 27, 39] we show CP-SNARKs for: proof of knowledge of an opening in the algebraic group model [25] (which actually comes for free), polynomial evaluation, degree bounds, and polynomial equations. In addition to these, we propose a new CP-SNARK for proving opening of several commitments with a proof consisting of one single group element; the latter relies on the PKE assumption [32] in the random oracle model. Also, we show that for a class of quadratic equations over committed polynomials (notably capturing some of the checks of our PHPs), we can obtain an optimized CP-SNARK in which the proof is empty as the verifier can test the relation using a pairing with the inputs (the inputs are commitments, i.e., group elements). This technique is reminiscent of the compiler from [13] that relies on linear encodings with quadratic tests.

PHPs for constraint systems. We propose a variety of PHPs for the R1CS constraint system and for a simplified variant of it that we call R1CS-lite. In brief, R1CS-lite is defined by two matrices $\boldsymbol{L}, \boldsymbol{R}$ and accepts a vector $\boldsymbol{x}$ if there is a $\boldsymbol{w}$ such that, for $\boldsymbol{c} = (1, \boldsymbol{x}, \boldsymbol{w})$, $\boldsymbol{L} \cdot \boldsymbol{c} \circ \boldsymbol{R} \cdot \boldsymbol{c} = \boldsymbol{c}$. We show that R1CS-lite can express arithmetic circuit satisfiability with essentially the same complexity

of R1CS, and its simpler form allows us to design slightly simpler PHPs. We believe this characterization of NP problems to be of independent interest.

Part of our techniques stem from those in Marlin [19]: we adopt their encoding of sparse matrices; also one of our main building blocks is the sumcheck protocol from Aurora of Ben-Sasson *et. al.* [8]. But in our PHPs we explore a different protocol that proves properties of sparse matrices and we introduce a refined efficient technique for zero-knowledge in a univariate sumcheck. In a nutshell, compared to [8] we show how to choose the masking polynomial with a specific sparse distribution that has only a constant-time impact on the prover. This idea and analysis of this technique is possible thanks to our fine-grained ZK formalism for PHPs. By combining this basic skeleton with different techniques we can obtain PHPs with different tradeoffs (see Table 2).

Commit-and-prove zkSNARKs from PHPs. We propose the first general compiler from an information-theoretic object such as (algebraic) IOPs— and more in general PHPs—to Commit-and-Prove zkSNARKs[7]. Recall that the latter is a SNARK where the verifier's input includes one (or several) *reusable* hiding commitment(s), i.e., to check that $\mathsf{R}(\mathsf{u}_1, \ldots, \mathsf{u}_\ell)$ holds for a tuple of commitments $(\hat{c}_j)_{j\in[\ell]}$ such that $\hat{c}_i$ opens to u_i. By reusable we mean that these commitments could be used in multiple proofs and with different proof systems since their commitment key is generated before the setup of the proof system. To obtain a CP-SNARK we cannot apply the committing methods for polynomials used in [19, 27]: these require a known bound on how many times we will evaluate the polynomials. This is analogous to knowing a bound on the number of proofs over those same committed polynomials, which may be unknown at commitment time. Therefore we apply more stringent requirements and assume these commitments to be full-fledged hiding rather than just somewhat-hiding.

To obtain our commit-and-prove compiler we adapt our compiler to zkSNARKs to include the following key idea: we prove a "link" between the committed witnesses $(\mathsf{u}_j)_{j\in[\ell]}$—which open *hiding* commitments $(\hat{c}_j)_{j\in[\ell]}$—and the PHP polynomials $(p_j)_{j\in[n]}$—which open *somewhat-hiding* commitments $(c_j)_{j\in[n]}$. We design a specific CP-SNARK for this task, $\mathsf{CP}_{\mathsf{link}}$. Our construction works for pairing-based commitments and supports a wide class of linking relations which include those in our PHP constructions.

Simplifying a little bit, our techniques involve proving equality of images of distinct (committed) polynomials on distinct domains and they are of independent interest. In particular they can plausibly be adapted to compile other zkSNARKs with similar properties—e.g., Marlin or PLONK [19, 27]—into CP-SNARKs *with commitments that can be reused among different proofs.*

Efficient CP-SNARKs with a universal setup are strongly motivated by practical applications. One of them is *committing-ahead-of-time* [10, 18] in which we commit to a value possibly *before* we can predict what we are going to prove about it. A CP-SNARK with a universal SRS, like those in this work, can be a requirement in the context of committing-ahead-of-time: if the setting requires

[7] Here we do not consider the alternative approach of explicitly proving in the PHP a relation augmented with commitment opening; this is often too expensive [18].

committing to data *before* knowing what properties to prove about them (which can happen on-demand), the same setting can benefit from an (unspecialized) SRS string available *before* knowing what to prove about the committed data.

Our work improves on the efficiency of LegoUAC in [18], a modular CP-SNARK construction with universal setup for universal relations (and the only one in literature to the best of our knowledge). Our results are also complementary to those of [18] (in particular their *specialized* CP-SNARKs with universal setup) and to those of works on efficient composable CP-SNARKs on commitments in prime order groups, such as [11]: our universal CP-SNARK can be composed with the schemes in these works *as they can all be derived from the same SRS*, or with some of the transparent instantiations in [11].

1.2 Other Related Work

In this work we focus on practical zkSNARKs with a universal and updatable setup and constant-size proofs. Recent work builds on our formalizations to expand this area designing a fully algebraic framework for modular arguments [47]. Here we briefly survey other works that obtain universality through other approaches at the cost of a larger proof size.

Concurrent work in [42] proposes a new scheme with universal—but not updatable— SRS and an asymptotically linear prover (our prover is quasi-linear due to the use of FFT). By recursive composition they achieve an asymptotically $O_\lambda(1)$-size proof. In practice this is about $9\times$ larger than some of our proofs.

Spartan [49] obtains preprocessing arguments with a URS; it trades a transparent setup for larger arguments and less efficient verification, ranging from $O(\log^2(n)$ to $O(\sqrt{n})$, depending on the instantiation.

Concurrent work in [43] extends Spartan techniques obtaining a linear-time prover. They obtain asymptotically constant-sized proofs through one step of recursive composition with Groth16 [33]; they do not discuss concrete proof sizes. This, however, yields a scheme with universal but not updatable setup. It would require an existing scheme with universal and updatable setup to achieve the latter; their work can thus be seen as complementary to ours.

Other works obtain universal SNARGS through a transparent setup and by exploiting the structure of the computation for succinctness. They mainly use two classes of techniques: hash-based vector commitments applied to oracle interactive proofs [4, 5, 6] or multivariate polynomial commitments and doubly-efficient interactive proofs [51, 53, 55, 56, 57, 58].

Fractal [20] achieves transparent zkSNARKs with recursive composition—the ability of a SNARG to prove computations involving prior SNARGs. Their work also uses an algebraically-flavored variant of interactive oracle proofs that they call *Reed–Solomon encoded holographic IOPs*.

Another line of work, e.g., [2, 8, 15, 16, 26], obtains a restricted notion of succinctness with no preprocessing, a linear verifier and sublinear proof size.

1.3 Outline

See Section 2 for preliminaries. In Section 3 we define PHPs; we describe PHP constructions in Section 4. Section 5 describes how to compile PHPs to universal zkSNARKs. Concrete compilations for the Lunar zkSNARKs are in Section 6.

2 Preliminaries and Notation

Universal Relations. A *universal relation* $\mathcal{R}$ is a set of triples $(\mathsf{R}, \mathsf{x}, \mathsf{w})$ where R is a relation, $\mathsf{x} \in \mathcal{D}_{\mathsf{x}}$ is called the *instance* (or input), $\mathsf{w} \in \mathcal{D}_w$ the *witness*, and $\mathcal{D}_{\mathsf{x}}, \mathcal{D}_{\mathsf{w}}$ are domains that may depend on R. Given $\mathcal{R}$, the corresponding *universal language* $\mathcal{L}(\mathcal{R})$ is the set $\{(\mathsf{R}, \mathsf{x}) : \exists \mathsf{w} : (\mathsf{R}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}\}$. For a size bound $\mathsf{N} \in \mathbb{N}$, $\mathcal{R}_{\mathsf{N}}$ denotes the subset of triples $(\mathsf{R}, \mathsf{x}, \mathsf{w})$ in $\mathcal{R}$ such that R has size at most N, i.e. $|\mathsf{R}| \leq \mathsf{N}$. In our work, we also write $\mathcal{R}(\mathsf{R}, \mathsf{x}, \mathsf{w}) = 1$ (resp. $\mathsf{R}(\mathsf{x}, \mathsf{w}) = 1$) to denote $(\mathsf{R}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$ (resp. $(\mathsf{x}, \mathsf{w}) \in \mathsf{R}$).

When discussing schemes that prove statements on committed values we assume that $\mathcal{D}_{\mathsf{w}}$ can be split in two subdomains $\mathcal{D}_{\mathsf{u}} \times \mathcal{D}_{\omega}$, and sometimes we use an even more fine-grained splitting of $\mathcal{D}_{\mathsf{u}} := (\mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell)$ for some arity ℓ.

2.1 Algebraic Preliminaries

We denote by $\mathbb{F}$ a finite field, by $\mathbb{F}[X]$ the ring of univariate polynomials, and by $\mathbb{F}_{<d}[X]$ (resp. $\mathbb{F}_{\leq d}[X]$) the set of polynomials in $\mathbb{F}[X]$ of degree $< d$ (resp. $\leq d$).

We briefly describe some algebraic preliminaries (see full version for details):

Vanishing and Lagrange Basis Polynomials. For any subset $S \subseteq \mathbb{F}$ we denote by $\mathcal{Z}_S(X) := \prod_{s \in S}(X - s)$ the *vanishing polynomial* of S, and by $\mathcal{L}_s^S(X)$ the *s-th Lagrange basis polynomial*, which is the unique polynomial of degree at most $|S| - 1$ such that for any $s' \in S$ it evaluates to 1 if $s = s'$ and to 0 otherwise.

Multiplicative subgroups. If $\mathbb{H} \subseteq \mathbb{F}$ is a multiplicative subgroup of order n, then its vanishing polynomial has a compact representation $\mathcal{Z}_{\mathbb{H}}(X) = (X^{|\mathbb{H}|} - 1)$. Similarly, for such $\mathbb{H}$ it holds $\mathcal{L}_\eta^{\mathbb{H}}(X) = \frac{\eta}{|\mathbb{H}|} \cdot \frac{X^{|\mathbb{H}|}-1}{X-\eta}$ [38, 50, 52]. Both $\mathcal{Z}_{\mathbb{H}}(X)$ and $\mathcal{L}_\eta^{\mathbb{H}}(X)$ can be evaluated in $O(\log n)$ field operations. We assume that $\mathbb{H}$ comes with a bijection $\phi_{\mathbb{H}} : \mathbb{H} \to [n]$, and we use elements of $\mathbb{H}$ to index the entries of a matrix $\boldsymbol{M} \in \mathbb{F}^{n \times n}$, i.e., $\boldsymbol{M}_{\eta,\eta'}$ denotes $\boldsymbol{M}_{\phi_{\mathbb{H}}(\eta),\phi_{\mathbb{H}}(\eta')}$, and similarly for vectors. For any vector $\boldsymbol{v} \in \mathbb{F}^n$, we denote by $v(X)$ its *interpolating polynomial* in $\mathbb{H}$, i.e., the unique degree-$(|\mathbb{H}| - 1)$ polynomial such that, for all $\eta \in \mathbb{H}$, $v(\eta) = \boldsymbol{v}_\eta$.

Univariate sumcheck. We use the lemma from [8, 19], which shows that for any $p \in \mathbb{F}_d[X]$ and multiplicative subgroup $\mathbb{H} \subset \mathbb{F}$, $\sigma = \sum_{\eta \in \mathbb{H}} p(\eta)$ iff there exists $q(X), r(X)$ such that $p(X) = q(X)\mathcal{Z}_{\mathbb{H}}(X) + Xr(X) + \sigma/|\mathbb{H}|$ with $\mathsf{deg}(r) < n - 1$.

Polynomial masking. Given a subgroup $\mathbb{H} \subset \mathbb{F}$ and an integer $\mathsf{b} \geq 1$, $\mathsf{Mask}_{\mathsf{b}}^{\mathbb{H}}(p(X))$ is a shorthand for $p(X) + \mathcal{Z}_{\mathbb{H}}(X)\rho(X)$ for a randomly sampled $\rho(X) \leftarrow\!\!\$\, \mathbb{F}_{<\mathsf{b}}[X]$.

Definition 1 (Bivariate Lagrange polynomial). *The* bivariate Lagrange polynomial *for a multiplicative subgroup $\mathbb{H} \subseteq \mathbb{F}$ is* $\Lambda_{\mathbb{H}}(X, Y) := \frac{\mathcal{Z}_{\mathbb{H}}(X) \cdot Y - X \cdot \mathcal{Z}_{\mathbb{H}}(Y)}{n \cdot (X - Y)}$.

This polynomial has two properties useful for our work: for all $\eta \in \mathbb{H}$, $\Lambda_{\mathbb{H}}(X, \eta) = \mathcal{L}_{\eta}^{\mathbb{H}}(X)$, and it can be evaluated in $O(\log n)$ time (see full version).

Sparse Matrix Encodings. For a matrix $\boldsymbol{M}$, $||\boldsymbol{M}||$ denotes the number of its nonzero entries, which we call its *density*. We occasionally use encodings for sparse matrices inspired to [19]. Let $\mathbb{K}$ be another multiplicative subgroup of $\mathbb{F}$ such that $|\mathbb{K}| \geq ||\boldsymbol{M}||$. In brief, a sparse encoding of a matrix $\boldsymbol{M}$ is a triple of polynomials $(\mathsf{val}_{\mathsf{M}}, \mathsf{row}_{\mathsf{M}}, \mathsf{col}_{\mathsf{M}})$ in $\mathbb{F}_{<|\mathbb{K}|}[X]$, where $\mathsf{row}_{\mathsf{M}} : \mathbb{K} \to \mathbb{H}$ (resp. $\mathsf{col}_{\mathsf{M}} : \mathbb{K} \to \mathbb{H}$) is the function such that $\mathsf{row}_{\mathsf{M}}(\kappa)$ (resp. $\mathsf{col}_{\mathsf{M}}(\kappa)$) is the row (resp. column) index of the κ-th nonzero entry of $\boldsymbol{M}$, and $\mathsf{val}_M : \mathbb{K} \to \mathbb{F}$ is the function that encodes the values of $\boldsymbol{M}$ in some arbitrary ordering. Hence it holds that for all $\kappa \in \mathbb{K}$, $\mathsf{val}_{\mathsf{M}}(\kappa) = \boldsymbol{M}_{\mathsf{row}_{\mathsf{M}}(\kappa), \mathsf{col}_{\mathsf{M}}(\kappa)}$. We define the *matrix-encoding polynomial* of $\boldsymbol{M}$ as the bivariate polynomial $V_M(X, Y) := \sum_{\kappa \in \mathbb{K}} \mathsf{val}_{\mathsf{M}}(\kappa) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}_{\mathsf{M}}(\kappa)}(X) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{col}_{\mathsf{M}}(\kappa)}(Y)$, and note that for all $\eta, \eta' \in \mathbb{H}$, $V_M(\eta, \eta') = \boldsymbol{M}_{\eta, \eta'}$.

The following lemma shows that a sparse encoding polynomial of a matrix $\boldsymbol{M}$ can be used to express linear transformations by $\boldsymbol{M}$. Proof in the full version.

Lemma 1 (Sparse Linear Encoding). *Let $\boldsymbol{M} \in \mathbb{F}^{n \times n}$ and let $V_M(X, Y)$ be its matrix-encoding polynomial. Let $\boldsymbol{v}, \boldsymbol{y} \in \mathbb{F}^n$ be two vectors and $v(X), y(X)$ be their interpolating polynomials over $\mathbb{H}$. Then $\boldsymbol{y} = \boldsymbol{M} \cdot \boldsymbol{v}$ if and only if $y(X) = \sum_{\eta \in \mathbb{H}} v(\eta) \cdot V_M(X, \eta)$.*

Joint Sparse Encodings for Multiple Matrices. When working with multiple matrices, it is sometimes convenient to use a sparse encoding that keeps track of entries that are nonzero in either of the matrices. This has the advantage of having a pair of $\mathsf{col}, \mathsf{row}$ polynomials common to all matrices. For example, for two matrices $\boldsymbol{L}, \boldsymbol{R}$, this encoding includes polynomials $\{\mathsf{val}_L, \mathsf{val}_R\}$ encoding their values, and polynomials $\{\mathsf{col}, \mathsf{row}\}$ that maintain the indices in which either of the matrix is nonzero. Namely, for any $\kappa \in \mathbb{K}$, we have $\mathsf{val}_L(\kappa) = \boldsymbol{L}_{\mathsf{row}(\kappa), \mathsf{col}(\kappa)}$ and $\mathsf{val}_R(\kappa) = \boldsymbol{R}_{\mathsf{row}(\kappa), \mathsf{col}(\kappa)}$. In this case though $|\mathbb{K}|$ is in the worst case $\geq ||\boldsymbol{L}|| + ||\boldsymbol{R}||$.

3 Polynomial Holographic IOPs

In this section we define our notion of Polynomial Holographic IOPs (PHP), that generalizes *algebraic holographic proofs* (AHPs) [19]. We show how to compile them into one another in the full version. In a nutshell, a PHP is an interactive oracle proof (IOP) system that works for a family of universal relations $\mathcal{R}$ that is specialized in two main ways. First, it is holographic, i.e., the verifier has oracle access to the relation encoding, a set of oracle polynomials created by a trusted party, the *holographic relation encoder* (or simply, *encoder*) $\mathcal{RE}$. Second, it is algebraic in the sense that the system works over a finite field $\mathbb{F}$: at each round the prover can send field elements or oracle polynomials to the verifier, while the latter can perform algebraic checks as queries over the prover's messages.

More formally, a Polynomial Holographic IOP is defined as follows.

Definition 2 (Polynomial Holographic IOP (PHP)). *Let $\mathcal{F}$ be a family of finite fields and let $\mathcal{R}$ be a universal relation. A Polynomial Holographic IOP over $\mathcal{F}$ for $\mathcal{R}$ is a tuple $\mathsf{PHP} = (\mathsf{r}, \mathsf{n}, \mathsf{m}, \mathsf{d}, \mathsf{n_e}, \mathcal{RE}, \mathcal{P}, \mathcal{V})$ where $\mathsf{r}, \mathsf{n}, \mathsf{m}, \mathsf{d}, \mathsf{n_e} : \{0,1\}^* \to \mathbb{N}$ are polynomial-time computable functions, and $\mathcal{RE}, \mathcal{P}, \mathcal{V}$ are three algorithms for the encoder, prover and verifier respectively, that work as follows.*

- **Offline phase:** *The encoder $\mathcal{RE}(\mathbb{F}, \mathsf{R})$ takes as input a field $\mathbb{F} \in \mathcal{F}$ and a relation description R, and returns $\mathsf{n}(0)$ polynomials $\{p_{0,j}\}_{j\in[\mathsf{n}(0)]}$ encoding R.*
- **Online phase:** *The prover $\mathcal{P}$ and verifier $\mathcal{V}$ run for $\mathsf{r}(|\mathsf{R}|)$ rounds and take respectively as input a tuple $(\mathsf{R}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$ and an instance x; the verifier has also oracle access to the polynomials encoding R.*
  In the i-th round, $\mathcal{V}$ sends a message $\rho_i \in \mathbb{F}$ to the prover, and $\mathcal{P}$ replies with $\mathsf{m}(i)$ messages $\{\pi_{i,j} \in \mathbb{F}\}_{j\in[\mathsf{m}(i)]}$, and $\mathsf{n}(i)$ oracle polynomials $\{p_{i,j} \in \mathbb{F}[X]\}_{j\in[\mathsf{n}(i)]}$, such that $\mathsf{deg}(p_{i,j}) < \mathsf{d}(|\mathsf{R}|, i, j)$.
- **Decision phase:** *After the $\mathsf{r}(|\mathsf{R}|)$-th round, the verifier outputs two sets of algebraic checks of the following type.*
  - *Degree checks: to check a bound on the degree of the polynomials sent by the prover. More in detail, let $\mathsf{n_p} = \sum_{k=1}^{\mathsf{r}(|\mathsf{R}|)} \mathsf{n}(k)$ and let $(p_1, \ldots, p_{\mathsf{n_p}})$ be the polynomials sent by $\mathcal{P}$. The verifier specifies a vector of integers $\boldsymbol{d} \in \mathbb{N}^{\mathsf{n_p}}$, which is satisfied if and only if $\forall k \in [\mathsf{n_p}] : \mathsf{deg}(p_k) \leq d_k$.*
  - *Polynomial checks: to check that certain polynomial identities hold for the oracle polynomials and the prover messages. Formally, let $\mathsf{n}^* = \sum_{k=0}^{\mathsf{r}(|\mathsf{R}|)} \mathsf{n}(k)$ and $\mathsf{m}^* = \sum_{k=1}^{\mathsf{r}(|\mathsf{R}|)} \mathsf{m}(k)$, and denote by $(p_1, \ldots, p_{\mathsf{n}^*})$ and $(\pi_1, \ldots, \pi_{\mathsf{m}^*})$ all the oracle polynomials (including the encoder's) and all the messages sent by the prover. The verifier can specify a list of $\mathsf{n_e}$ tuples, each of the form $(G, v_1, \ldots, v_{\mathsf{n}^*})$, where $G \in \mathbb{F}[X, X_1, \ldots, X_{\mathsf{n}^*}, Y_1, \ldots, Y_{\mathsf{m}^*}]$ and every $v_k \in \mathbb{F}[X]$. Then a tuple $(G, \boldsymbol{v})$ is satisfied if and only if $F(X) \equiv 0$ where*

$$F(X) := G(X, \{p_k(v_k(X))\}_{k\in[\mathsf{n}^*]}, \{\pi_k\}_{k\in[\mathsf{m}^*]})$$

  The verifier accepts if and only if all the checks are satisfied.

Efficiency Measures. *Given the functions $\mathsf{r}, \mathsf{d}, \mathsf{n}, \mathsf{m}$ in the tuple PHP, one can derive some efficiency measures of the protocol PHP such as the total number of oracles sent by the encoder, $\mathsf{n}(0)$, by the prover $\mathsf{n_p}$, by both in total, n^*; or the number of prover messages m^*. In addition to these, we define the following shorthands for two more measures of PHP; degree D, and proof length $\mathsf{L}(|\mathsf{R}|)$:*

$$\mathsf{D} := \max_{\substack{\mathsf{R}\in\mathcal{R} \\ i\in[0,\mathsf{r}(|\mathsf{R}|)] \\ j\in[\mathsf{n}(i)]}} (\mathsf{d}(|\mathsf{R}|, i, j)), \qquad \mathsf{L}(|\mathsf{R}|) := \sum_{\substack{i\in[\mathsf{r}(|\mathsf{R}|)] \\ j\in[\mathsf{n}(i)]}} \mathsf{m}(i) + \mathsf{d}(|\mathsf{R}|, i, j).$$

PHP *should satisfy completeness, (knowledge) soundness and zero-knowledge:*

Completeness. *If for all $\mathbb{F} \in \mathcal{F}$ and any $(\mathsf{R}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$, the checks returned by $\mathcal{V}^{\mathcal{RE}(\mathbb{F},\mathsf{R})}(\mathbb{F}, \mathsf{x})$ after interacting with (honest) $\mathcal{P}(\mathbb{F}, \mathsf{R}, \mathsf{x}, \mathsf{w})$ are always satisfied.*

Soundness. *A PHP is ϵ-sound if for every field $\mathbb{F} \in \mathcal{F}$, relation-instance tuple $(\mathsf{R}, \mathsf{x}) \notin \mathcal{L}(\mathcal{R})$ and prover $\mathcal{P}^*$ we have* $\Pr[\langle \mathcal{P}^*, \mathcal{V}^{\mathcal{RE}(\mathbb{F},\mathsf{R})}(\mathbb{F}, \mathsf{x})\rangle = 1] \leq \epsilon$.

Knowledge Soundness. *A PHP is ϵ-knowledge-sound if there exists a polynomial-time knowledge extractor $\mathcal{E}$ such that for any prover $\mathcal{P}^*$, field $\mathbb{F} \in \mathcal{F}$, relation R, instance x and auxiliary input z:*

$$\Pr\left[(\mathsf{R}, \mathsf{x}, \mathsf{w}) \in \mathsf{R} : \mathsf{w} \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathbb{F}, \mathsf{R}, \mathsf{x}, z)\right] \geq \Pr[\langle \mathcal{P}^*(\mathbb{F}, \mathsf{R}, \mathsf{x}, z), \mathcal{V}^{\mathcal{RE}(\mathbb{F},\mathsf{R})}(\mathbb{F}, \mathsf{x})\rangle = 1] - \epsilon$$

where $\mathcal{E}$ has oracle access to $\mathcal{P}^*$, i.e., it can query the next message function of $\mathcal{P}^*$ (and rewind it) and obtain all the messages and polynomials returned by it.

Zero-Knowledge. *A PHP is* ϵ-zero-knowledge *if there exists a PPT simulator $\mathcal{S}$ such that for every field $\mathbb{F} \in \mathcal{F}$, every triple $(\mathsf{R}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$, and every algorithm $\mathcal{V}^*$ the following random variables are within ϵ statistical distance:*

$$\mathsf{View}\big(\mathcal{P}(\mathbb{F}, \mathsf{R}, \mathsf{x}, \mathsf{w}) \ , \mathcal{V}^*\big) \approx_\epsilon \mathsf{View}\big(\mathcal{S}^{\mathcal{V}^*}(\mathbb{F}, \mathsf{R}, \mathsf{x})\big)$$

where $\mathsf{View}\big(\mathcal{P}(\mathbb{F}, \mathsf{R}, \mathsf{x}, \mathsf{w}) \ , \mathcal{V}^*\big)$ consists of $\mathcal{V}^*$'s randomness, $\mathcal{P}$'s messages $\pi_1, \ldots, \pi_{\mathsf{m}^*}$ (which do not include the oracles), and $\mathcal{V}^*$'s list of checks, while $\mathsf{View}\big(\mathcal{S}^{\mathcal{V}^*}(\mathbb{F}, \mathsf{R}, \mathsf{x})\big)$ consists of $\mathcal{V}^*$'s randomness followed by $\mathcal{S}$'s output, obtained after having straightline access to $\mathcal{V}^*$, and $\mathcal{V}^*$'s list of checks.

Remark 1 (About messages and constant polynomials). We explicitly model the prover's messages π_i, although they could be replaced by (degree-0) polynomial oracles evaluated on 0 during the checks. This is useful for zero-knowledge: while messages are supposed not to leak information on the witness (i.e., they must be simulated), this does not hold for the oracles. Thus, in our compiler, we will not need to hide messages π_i from the verifier, only the oracles.

On the class of polynomial checks. Above we describe the class of polynomial checks of the verifier in full generality; nonetheless, when possible, we use more convenient notations. We note that this class includes low-degree polynomials like $G(\{p_i(X)\}_i)$ (e.g., $p_1(X)p_2(X)p_3(X) + p_4(X)$), in which case each $v_i(X) = X$, polynomial evaluations $p_i(x)$, in which case $v_i(X) = x$, tests over $\mathcal{P}$ messages, e.g., $p_i(x) - \pi_j$, and combinations of all these.

Public coin and non-adaptive queries. In the rest of this work, we only consider PHPs that are public coin and non-adaptive: the messages of the verifier are random elements and its checks are independent of the prover's messages.

Below we define two additional properties that can be satisfied by a PHP:

Bounded Zero-Knowledge. This property will be useful for our compiler of Section 5. We require that zero-knowledge holds even if the view includes a bounded number of evaluations of oracle polynomials at given points.

However, we cannot allow evaluations on any possible point: specific points could leak bits of information of the witness. Thus we define a notion of "admissible" evaluations. Below we say that a list $\mathcal{L} = \{(i_1, y_1), \ldots\}$ is (b, C)-bounded where $\mathsf{b} \in \mathbb{N}^{\mathsf{n_p}}$ and C is a PT algorithm if $\forall i \in [\mathsf{n_p}] : |\{(i, y) : (i, y) \in \mathcal{L}\}| \leq \mathsf{b}_i$ and $\forall (i, y) \in \mathcal{L} : \mathsf{C}(i, y) = 1$.

Definition 3 ((b,C)**-Zero-Knowledge).** *We say that* PHP *is* (b,C)*-Zero-Knowledge if for every triple* $(\mathsf{R},\mathsf{x},\mathsf{w}) \in \mathcal{R}$*, and every* (b,C)*-bounded list* $\mathcal{L}$*, the following random variables are within* ϵ *statistical distance:*

$$\big(\mathsf{View}\big(\mathcal{P}(\mathbb{F},\mathsf{R},\mathsf{x},\mathsf{w}),\mathcal{V}\big),(p_i(y))_{(i,y)\in\mathcal{L}}\big) \approx_\epsilon \mathcal{S}(\mathbb{F},\mathsf{R},\mathsf{x},\mathcal{V}(\mathbb{F},\mathsf{x}),\mathcal{L}).$$

where $p_1,\ldots,p_{\mathsf{n_p}}$ *are the polynomials returned by the prover* $\mathcal{P}$*.*

Moreover, we say that PHP *is* honest-verifier zero-knowledge with query bound b (b*-HVZK for short*) *if there exists a PT algorithm* C *such that* PHP *is* (b,C)*-ZK and for all* $i \in \mathbb{N}$ *we have* $\Pr_{y \leftarrow\!\$\, \mathbb{F}}[\mathsf{C}(i,y)=0] \in \mathsf{negl}(\lambda)$.

3.1 PHP Verifier Relation

We formalize the definition of an NP relation that models the PHP verifier's decision phase. We shall use it in our compiler in Section 5.

Let $\mathsf{PHP} = (\mathsf{r},\mathsf{n},\mathsf{m},\mathsf{d},\mathsf{n_e},\mathcal{RE},\mathcal{P},\mathcal{V})$ be a PHP protocol over a finite field family $\mathcal{F}$ for a universal relation $\mathcal{R}$, with D as its maximum degree. We define $\mathcal{R}_{\mathsf{php}}$ as a family of relations that express the checks of $\mathcal{V}$ over the oracle polynomials, which can be formally defined as follows.

Let $\mathsf{n_p},\mathsf{n}^* \in \mathbb{N}$ be two positive integers, and consider the following relations:

$$\mathsf{R}_{\mathsf{deg}}\left((d_j)_{j\in[\mathsf{n_p}]},(p_j)_{j\in[\mathsf{n_p}]}\right) := \bigwedge\nolimits_{j\in[\mathsf{n_p}]} \mathsf{deg}(p_j) \overset{?}{\leq} d_j$$

$$\mathsf{R}_{\mathsf{eq}}\left((G',\boldsymbol{v}),(p_j)_{j\in[\mathsf{n}^*]}\right) := G'\left(X,(p_j(v_j(X)))_{j\in[\mathsf{n}^*]}\right) \overset{?}{\equiv} 0$$

where $G \in \mathbb{F}[X,X_1,\ldots,X_{\mathsf{n}^*}]$ and $\boldsymbol{v} = (v_1,\ldots,v_{\mathsf{n}^*}) \in \mathbb{F}[X]^{\mathsf{n}^*}$. For a PHP verifier that returns a polynomial check $(G',\boldsymbol{v})$, R_{eq} expresses such check if one considers G' as the partial evaluation of G at $(Y_1=\pi_1,\ldots,Y_{\mathsf{m}^*}=\pi_{\mathsf{m}^*})$. $\mathsf{R}_{\mathsf{deg}}$ instead expresses the degree checks of a PHP verifier.

Given relations $\mathsf{R}_A \subset \mathcal{D}_{\mathsf{x}} \times \mathcal{D}_{\mathsf{w}}$ and $\mathsf{R}_B \subset \mathcal{D}'_{\mathsf{x}} \times \mathcal{D}_{\mathsf{w}}$ with a common domain $\mathcal{D}_{\mathsf{w}}$ for the witness, consider the product $\mathsf{R}_A \times \mathsf{R}_B \subset \mathcal{D}_{\mathsf{x}} \times \mathcal{D}'_{\mathsf{x}} \times \mathcal{D}_{\mathsf{w}}$ containing all the tuples $(\mathsf{x}_A,\mathsf{x}_B,\mathsf{w})$ where $(\mathsf{x}_A,\mathsf{w}) \in \mathsf{R}_A$ and $(\mathsf{x}_B,\mathsf{w}) \in \mathsf{R}_B$. For $\mathsf{n_e} \in \mathbb{N}$, let

$$\mathsf{R}_{\mathsf{n}^*,\mathsf{n_p},\mathsf{n_e}} := \mathsf{R}_{\mathsf{deg}} \times \overbrace{\mathsf{R}_{\mathsf{eq}} \times \cdots \times \mathsf{R}_{\mathsf{eq}}}^{\mathsf{n_e}\text{ times}} \quad \text{and} \quad \mathcal{R}_{\mathsf{php}} := \left\{\mathsf{R}_{\mathsf{n}^*(|\mathsf{R}|),\mathsf{n_p}(|\mathsf{R}|),\mathsf{n_e}(|\mathsf{R}|)} : \mathsf{R} \in \mathcal{R}\right\}$$

where $\mathsf{n}^*(|\mathsf{R}|) = \sum_{i=0}^{\mathsf{r}(|\mathsf{R}|)} \mathsf{n}(i)$ and $\mathsf{n_p}(|\mathsf{R}|) = \sum_{i=1}^{\mathsf{r}(|\mathsf{R}|)} \mathsf{n}(i)$ are the number of total and prover oracle polynomials respectively, when running PHP with relation R.

4 Our PHP Constructions

We propose a collection of PHP constructions for two types of constraint systems: the by now standard rank-1 constraint systems [28] and an equally expressive variant we introduce in Section 4.1 called R1CS-lite.

R1CS-lite can be seen as a simplified version of R1CS with only two matrices. In brief, an R1CS-lite relation is defined by two matrices $\boldsymbol{L},\boldsymbol{R}$ and is satisfied if

Table 2. Comparison of our PHP constructions, all with complexities: $O(m \log m)$ for $\mathcal{RE}$, $O(m \log m + n \log n)$ for $\mathcal{P}$ and $O(\ell + \log m + \log n)$ for $\mathcal{V}$. To have a simpler table, we assume $|\mathbb{K}| = m > 2n$, which is often the case. We call $|\pi| = 5n + 2m - 2\ell + 2\mathsf{b}_a + 2\mathsf{b}_b + 2\mathsf{b}_s + 6\mathsf{b}_q - 4$, and $|\pi'| = |\pi| + n - \ell + \mathsf{b}_w + 7\mathsf{b}_q$. For verifier checks, we denote by "deg" the number of degree checks that require a tight bound; the last two columns show the degree of the two polynomial checks: in the first one we have all $v_j(X) = y$ and in the second one all $v_j(X) = X$. "Rk" ("full") denote remark (resp. full version).

PHP		**degree**	**oracles**		**msgs**	**proof**	$\mathcal{V}$ **checks**		
name	ref.		$\mathcal{RE}$	$\mathcal{P}$		**length**	deg	$\mathsf{deg}_{X,\{X_i\}}(G_1)$	$\mathsf{deg}_{X,\{X_i\}}(G_2)$
$\mathsf{PHP}_{\mathsf{lite1}}$	4.1	$2m$	8	7	1	$\lvert\pi\rvert + 2m$	2	2	2
$\mathsf{PHP}_{\mathsf{lite1x}}$	Rk.2	$2m$	5	7	1	$\lvert\pi\rvert + 2m$	2	2	3
$\mathsf{PHP}_{\mathsf{lite2}}$	full	m	24	7	1	$\lvert\pi\rvert$	2	2	2
$\mathsf{PHP}_{\mathsf{lite2x}}$	full	m	16	7	1	$\lvert\pi\rvert$	2	2	3
$\mathsf{PHP}_{\mathsf{r1cs1}}$	full	$3m$	9	8	1	$\lvert\pi'\rvert + 4m$	2	2	2
$\mathsf{PHP}_{\mathsf{r1cs1x}}$	full	$3m$	6	8	1	$\lvert\pi'\rvert + 4m$	2	2	3
$\mathsf{PHP}_{\mathsf{r1cs2}}$	full	m	57	8	1	$\lvert\pi'\rvert$	2	2	2
$\mathsf{PHP}_{\mathsf{r1cs2x}}$	full	m	42	8	1	$\lvert\pi'\rvert$	2	2	3
$\mathsf{PHP}_{\mathsf{r1cs3}}$	full	$3m$	12	8	1	$\lvert\pi'\rvert$	2	2	5

there exists a vector $\boldsymbol{c}$ such that $(\boldsymbol{L} \cdot \boldsymbol{c}) \circ (\boldsymbol{R} \cdot \boldsymbol{c}) = \boldsymbol{c}$. We show that R1CS-lite is as expressive as R1CS since it can represent arithmetic circuit satisfiability with essentially the same complexity as R1CS (see full version)[8]. It allows us to obtain PHP constructions (and resulting zkSNARKs) that are simpler and more efficient. More formally, R1CS-lite is defined as follows.

Definition 4 (R1CS-lite). *Let $\mathbb{F}$ be a finite field and $n, m \in \mathbb{N}$ be positive integers. The universal relation $\mathcal{R}_{R1CS\text{-}lite}$ is the set of triples*

$$(\mathsf{R}, \mathsf{x}, \mathsf{w}) := ((\mathbb{F}, n, m, \ell, \{\boldsymbol{L}, \boldsymbol{R}\}), \boldsymbol{x}, \boldsymbol{w})$$

where $\boldsymbol{L}, \boldsymbol{R} \in \mathbb{F}^{n \times n}$, $\max\{||\boldsymbol{L}||, ||\boldsymbol{R}||\} \leq m$, the first ℓ rows of $\boldsymbol{R}$ are $(-1, 0, \ldots, 0) \in \mathbb{F}^{1 \times n}$, $\boldsymbol{x} \in \mathbb{F}^{\ell - 1}$, $\boldsymbol{w} \in \mathbb{F}^{n - \ell}$, and for $\boldsymbol{c} := (1, \boldsymbol{x}, \boldsymbol{w})$, it holds $(\boldsymbol{Lc}) \circ (\boldsymbol{Rc}) = \boldsymbol{c}$.

In this section, we present one PHP for R1CS-lite relations and give the intuition to obtain other PHP variants. The PHPs for the R1CS language follow the same bare-bone protocol, differing mainly in the number of matrices and an additional witness vector. In Table 2 we give a summary of all our PHPs and their measures.

4.1 Our PHPs for R1CS-lite

In all our constructions we use a variant of R1CS-lite in which we slightly expand the witness, and we express the witnesses and the check in polynomial form.

[8] Comparing to R1CS, the number of columns in R1CS-lite matrices do not change and the number of rows increase by the amount of public inputs, for the same circuit. The count of nonzero entries in R1CS-lite is smaller for virtually every circuit.

Definition 5 (Polynomial R1CS-lite). *Let $\mathbb{F}$ be a finite field, and $n, m \in \mathbb{N}$ be positive integers. The universal relation $\mathcal{R}_{polyR1CS\text{-}lite}$ is the set of triples*

$$\big((\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}\}, \ell), \boldsymbol{x}, (a'(X), b'(X))\big)$$

where $\boldsymbol{L}, \boldsymbol{R} \in \mathbb{F}^{n\times n}$, $\max\{||\boldsymbol{L}||, ||\boldsymbol{R}||\} \leq m$, $\boldsymbol{x} \in \mathbb{F}^{\ell-1}$, $a'(X), b'(X) \in \mathbb{F}_{\leq n-\ell-1}[X]$, and such that, for $\mathbb{L} := \{\phi_{\mathbb{H}}^{-1}(i)\}_{i\in[\ell]}$, $\boldsymbol{x}' = (1, \boldsymbol{x})$, $a(X) := \sum_{\eta\in\mathbb{L}} \boldsymbol{x}'_{\phi_{\mathbb{H}}(\eta)} \cdot \mathcal{L}_\eta^{\mathbb{H}}(X) + a'(X) \cdot \mathcal{Z}_{\mathbb{L}}(X)$ and $b(X) := 1 + b'(X) \cdot \mathcal{Z}_{\mathbb{L}}(X)$, it holds, over $\mathbb{F}[X, Z]$,

$$a(X) + Z \cdot b(X) + \sum\nolimits_{\eta,\eta'\in\mathbb{H}} (\boldsymbol{L}_{\eta,\eta'} + Z \cdot \boldsymbol{R}_{\eta,\eta'}) \cdot a(\eta') \cdot b(\eta') \cdot \mathcal{L}_\eta^{\mathbb{H}}(X) = 0 \quad (1)$$

In the full version we prove that $\mathcal{L}(\mathcal{R}_{R1CS-lite}) \equiv \mathcal{L}(\mathcal{R}_{polyR1CS\text{-}lite})$.

Our PHP $\mathsf{PHP}_{\mathsf{lite1}}$. We describe the main ideas of our protocol $\mathsf{PHP}_{\mathsf{lite1}}$. The prover's goal is to convince the verifier that the polynomials $a(X), b(X)$ satisfy equation (1). To this end, the relation encoder $\mathcal{RE}$ encodes the matrices $\boldsymbol{L}, \boldsymbol{R}$ by using a joint sparse encoding (see Section 2.1), which consists of four polynomials $(\mathsf{val}_L, \mathsf{val}_R, \mathsf{col}, \mathsf{row})$ in $\mathbb{F}_{<|\mathbb{K}|}[X]$. In this case we use a multiplicative subgroup $\mathbb{K} \subseteq \mathbb{F}$ of minimal cardinality such that $|\mathbb{K}| \geq 2m \geq ||\boldsymbol{L}|| + ||\boldsymbol{R}||$.

By applying the sparse linear encoding of Lemma 1 to the matrices $\boldsymbol{L}$ and $\boldsymbol{R}$ and using the property of the bivariate Lagrange polynomial that $\Lambda_{\mathbb{H}}(X, \eta) = \mathcal{L}_\eta^{\mathbb{H}}(X)$, equation (1) can be expressed as

$$\begin{aligned} 0 &= a(X) + Z \cdot b(X) + \sum\nolimits_{\eta\in\mathbb{H}} a(\eta) \cdot b(\eta) \cdot (V_L(X, \eta) + Z \cdot V_R(X, \eta)) \\ &= \sum\nolimits_{\eta\in\mathbb{H}} (a(\eta) + Z \cdot b(\eta)) \cdot \Lambda_{\mathbb{H}}(X, \eta) + a(\eta) \cdot b(\eta) \cdot V_{LR}(X, \eta, Z) \end{aligned} \quad (2)$$

where, exploiting the use of $\mathsf{col}, \mathsf{row}$ common to $\boldsymbol{L}, \boldsymbol{R}$, $V_{LR}(X, Y, Z)$ equals:

$$V_L(X, Y) + Z \cdot V_R(X, Y) = \sum\nolimits_{\kappa\in\mathbb{K}} (\mathsf{val}_L(\kappa) + Z \cdot \mathsf{val}_R(\kappa)) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(X) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{col}(\kappa)}(Y)$$

In order to show that $a(X), b(X)$ satisfy equation (2), the verifier draws random points $x, \alpha \leftarrow_{\$} \mathbb{F}$ that are used to "compress" the equation from $\mathbb{F}[X, Z]$ to $\mathbb{F}$. Then, the prover's task becomes to show that

$$\sum\nolimits_{\eta\in\mathbb{H}} (a(\eta) + \alpha \cdot b(\eta)) \cdot \Lambda_{\mathbb{H}}(x, \eta) + a(\eta) \cdot b(\eta) \cdot V_{LR}(x, \eta, \alpha) = 0$$

This is done via a univariate sumcheck over $p(X) := (a(X) + \alpha \cdot b(X)) \cdot \Lambda_{\mathbb{H}}(x, X) + a(X) \cdot b(X) \cdot V_{LR}(x, X, \alpha)$. However, since $p(X)$ depends on the witness, we make the sumcheck zero-knowledge by doing it over $p(X) + s(X)$ for a random polynomial $s(X)$ sent by the prover in the first round. Although this resembles the zero-knowledge sumcheck technique of [8], we propose an optimized way to randomly sample a sparse $s(X)$, which is sufficient for the bounded zero-knowlegde of our PHP. So, for the sumcheck the prover sends two polynomials $q(X), r(X)$ such that $s(X) + p(X) = q(X) \cdot \mathcal{Z}_{\mathbb{H}}(X) + X \cdot r(X)$. The verifier checks this equation by evaluating all the polynomials on a random point $y \leftarrow_{\$} \mathbb{F} \setminus \mathbb{H}$. To do this, the verifier can compute on its own (in $O(\log n)$ time) the polynomials

$\Lambda_{\mathbb{H}}(x,y)$, $\mathcal{Z}_{\mathbb{H}}(y)$, and query all the others, except for $V_{LR}(x,y,\alpha)$. For the latter the prover sends a candidate value σ and runs a univariate sumcheck to convince the verifier that $\sigma = \sum_{\kappa\in\mathbb{K}}(\mathsf{val}_L(\kappa)+\alpha\cdot\mathsf{val}_R(\kappa))\cdot\mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(x)\cdot\mathcal{L}^{\mathbb{H}}_{\mathsf{col}(\kappa)}(y)$.

In what follows we give a detailed description of the PHP protocol $\mathsf{PHP}_{\mathsf{lite1}}$.

Offline phase $\mathcal{RE}(\mathbb{F},n,m,\{\boldsymbol{L},\boldsymbol{R}\},\ell)$. The holographic relation encoder takes as input a description of the specific relation and outputs eight polynomials

$$\big\{\mathsf{col}(X),\mathsf{row}(X),\mathsf{cr}(X),\mathsf{col}'(X),\mathsf{row}'(X),\mathsf{cr}'(X),\mathsf{vcr}_L(X),\mathsf{vcr}_R(X)\big\}\in\mathbb{F}_{\leq|\mathbb{K}|}[X].$$

The polynomials $\{\mathsf{col},\mathsf{row},\mathsf{val}_L,\mathsf{val}_R\}$ are the joint sparse encoding of $\{\boldsymbol{L},\boldsymbol{R}\}$. The holographic relation encoder computes:

$$\mathsf{cr}(X) := \sum\nolimits_{\kappa\in\mathbb{K}}\mathsf{col}(\kappa)\cdot\mathsf{row}(\kappa)\cdot\mathcal{L}^{\mathbb{K}}_{\kappa}(X)$$
$$\{\mathsf{vcr}_M(X) := \sum\nolimits_{\kappa\in\mathbb{K}}\mathsf{val}_M(\kappa)\cdot\mathsf{cr}(\kappa)\cdot\mathcal{L}^{\mathbb{K}}_{\kappa}(X)\}_{M\in\{L,R\}}$$

$$\mathsf{col}'(X) := X\cdot\mathsf{col}(X),\quad \mathsf{row}'(X) := X\cdot\mathsf{row}(X),\quad \mathsf{cr}'(X) := X\cdot\mathsf{cr}(X)$$

Essentially, the polynomials $\mathsf{cr}(X),\mathsf{vcr}_L(X)$ and $\mathsf{vcr}_R(X)$ are low-degree extensions of the evaluations in $\mathbb{K}$ of $\big(\mathsf{col}(X)\cdot\mathsf{row}(X)\big)$, $\big(\mathsf{val}_L(X)\cdot\mathsf{col}(X)\cdot\mathsf{row}(X)\big)$ and $\big(\mathsf{val}_R(X)\cdot\mathsf{col}(X)\cdot\mathsf{row}(X)\big)$ respectively, while $\mathsf{col}',\mathsf{row}'$ and cr' are a shifted version of $\mathsf{col},\mathsf{row}$ and cr each. The intuition behind expanding the sparse encoding of $\boldsymbol{L}$, $\boldsymbol{R}$ in this way is to keep the polynomial checks of the verifier of the lowest possible degree. In particular we are interested in obtaining a PHP where $\mathsf{deg}_{X,\{X_i\}}(G)\leq 2$ as it enables interesting instantiations of our compiler. As an example, by adding $\mathsf{cr}(X)$ we can replace terms involving $\mathsf{col}(X)\cdot\mathsf{row}(X)$ with $\mathsf{cr}(X)$. This shall become more clear when looking at the decision phase.

Online phase $\langle\mathcal{P}\left((\mathbb{F},n,m,\{\boldsymbol{L},\boldsymbol{R}\},\ell),\boldsymbol{x},(a'(X),b'(X))\right),\mathcal{V}(\mathbb{F},n,m,\boldsymbol{x})\rangle$.

Round 1: $\mathcal{P}\xrightarrow{\{\hat{a}'(X),\hat{b}'(X),s(X)\}}\mathcal{V}$

The prover samples polynomials $q_s(X)\leftarrow_{\$}\mathbb{F}_{\mathsf{b}_s+\mathsf{b}_q-1}[X]$ and $r_s(X)\leftarrow_{\$}\mathbb{F}_{\mathsf{b}_r+\mathsf{b}_q-1}[X]$, and sets $s(X) := q_s(X)\cdot\mathcal{Z}_{\mathbb{H}}(X)+X\cdot r_s(X)$. Note that, whenever $\mathsf{b}_r+\mathsf{b}_q\leq n$, the pair $q_s(X),r_s(X)$ is a unique decomposition of $s(X)$, and also $s(X)\in\mathbb{F}_{\leq n+\mathsf{b}_s+\mathsf{b}_q-1}[X]$. $\mathcal{P}$ sends $s(X)$ to $\mathcal{V}$ together with randomized versions of the witness polynomials $\hat{a}'(X)\leftarrow_{\$}\mathsf{Mask}^{\mathbb{H}\setminus\mathbb{L}}_{\mathsf{b}_a+\mathsf{b}_q}(a'(X))\in\mathbb{F}_{\leq n-\ell+\mathsf{b}_a+\mathsf{b}_q-1}[X]$ and $\hat{b}'(X)\leftarrow_{\$}\mathsf{Mask}^{\mathbb{H}\setminus\mathbb{L}}_{\mathsf{b}_b+\mathsf{b}_q}(b'(X))\in\mathbb{F}_{\leq n-\ell+\mathsf{b}_b+\mathsf{b}_q-1}[X]$.

Round 2: $\mathcal{V}\xrightarrow{x,\alpha}\mathcal{P}\xrightarrow{\{q(X),r(X)\}}\mathcal{V}$

The verifier sends two random points $x,\alpha\leftarrow_{\$}\mathbb{F}$. The prover uses the pair x,α to "compress" the check of equation (1) over $\mathbb{F}[X,Z]$ into the sumcheck statement $\sum_{\eta\in\mathbb{H}}p(\eta)=0$ over $\mathbb{F}$ for the polynomial $p(X) := (\hat{a}(X)+\alpha\cdot\hat{b}(X))\cdot\Lambda_{\mathbb{H}}(x,X)+$

$\hat{a}(X) \cdot \hat{b}(X) \cdot V_{LR}(x, X, \alpha)$ where, for $\boldsymbol{x}' = (1, \boldsymbol{x})$, we have

$$\hat{a}(X) := \hat{a}'(X) \cdot \mathcal{Z}_{\mathbb{L}}(X) + \sum\nolimits_{\eta \in \mathbb{L}} \boldsymbol{x}'_{\phi_{\mathbb{H}}(\eta)} \cdot \mathcal{L}^{\mathbb{H}}_{\eta}(X) \in \mathbb{F}_{\leq n+\mathsf{b}_a+\mathsf{b}_q-1}[X],$$
$$\hat{b}(X) := \hat{b}'(X) \cdot \mathcal{Z}_{\mathbb{L}}(X) + 1 \in \mathbb{F}_{\leq n+\mathsf{b}_b+\mathsf{b}_q-1}[X],$$

Next, $\mathcal{P}$ computes and sends polynomials $q(X) \in \mathbb{F}_{\leq 2n+\mathsf{b}_a+\mathsf{b}_b+2\mathsf{b}_q-3}[X]$ and $r(X) \in \mathbb{F}_{\leq n-2}[X]$—such that $s(X) + p(X) = q(X) \cdot \mathcal{Z}_{\mathbb{H}}(X) + X \cdot r(X)$—to prove the univariate sumcheck $\sum_{\eta \in \mathbb{H}} s(\eta) + p(\eta) = 0$. Note that by construction $\sum_{\eta \in \mathbb{H}} s(\eta) = 0$; its role here is to (sufficiently) randomize $q(X), r(X)$ in a way that their evaluations do not leak information about the witness (Theorem 2).

Round 3: $\mathcal{V} \xrightarrow{\quad y \quad} \mathcal{P} \xrightarrow{\quad \sigma,\ \{\ q'(X), r'(X)\ \} \quad} \mathcal{V}$

The verifier sends a random point $y \leftarrow\!\!\$\ \mathbb{F} \setminus \mathbb{H}$. The prover uses y to compute $\sigma \leftarrow V_{LR}(x, y, \alpha)$ and then defines the degree-$(|\mathbb{K}| - 1)$ polynomial

$$p'(X) := \sum\nolimits_{\kappa \in \mathbb{K}} (\mathsf{val}_L(\kappa) + \alpha \cdot \mathsf{val}_R(\kappa)) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(x) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{col}(\kappa)}(y) \cdot \mathcal{L}^{\mathbb{K}}_{\kappa}(X)$$

The goal of the prover is to convince the verifier that $\sum_{\kappa \in \mathbb{K}} p'(\kappa) = \sigma$ and

$$\forall \kappa \in \mathbb{K} : p'(\kappa) = (\mathsf{val}_L(\kappa) + \alpha \cdot \mathsf{val}_R(\kappa)) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(x) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{col}(\kappa)}(y)$$

These two statements can be combined in such a way that $\mathcal{P}$ does not need to send $p'(X)$, which is implicitly known by $\mathcal{V}$ as it depends on $\mathcal{RE}$ polynomials.

The first claim, since $\mathsf{deg}(p') < |\mathbb{K}|$, reduces to proving that its constant term is $\frac{\sigma}{|\mathbb{K}|}$, for which $\mathcal{P}$ sends $r'(X) \in \mathbb{F}_{\leq |\mathbb{K}|-2}[X]$ such that $p'(X) = X \cdot r'(X) + \frac{\sigma}{|\mathbb{K}|}$.

The second claim, by definition of $\mathcal{L}^{\mathbb{H}}(\cdot)$, means proving that $\forall\ \kappa \in \mathbb{K}$:

$$n^2 p'(\kappa)(x - \mathsf{row}(\kappa))(y - \mathsf{col}(\kappa)) = \big(\mathsf{val}_L(\kappa) + \alpha \mathsf{val}_R(\kappa)\big)\mathsf{row}(\kappa)\mathsf{col}(\kappa)\mathcal{Z}_{\mathbb{H}}(x)\mathcal{Z}_{\mathbb{H}}(y).$$

By definition of $p'(X)$ and of the relation polynomials, $\mathcal{P}$ can define

$$t(X) := \frac{\sigma}{|\mathbb{K}|} \cdot n^2 \cdot (xy + \mathsf{cr}(X) - x \cdot \mathsf{col}(X) - y \cdot \mathsf{row}(X)) + r'(X) \cdot n^2 \cdot (xy \cdot X + \mathsf{cr}'(X) -$$
$$x \cdot \mathsf{col}'(X) - y \cdot \mathsf{row}'(X)) - \big(\mathsf{vcr}_L(X) + \alpha \cdot \mathsf{vcr}_R(X)\big) \cdot \mathcal{Z}_{\mathbb{H}}(x) \cdot \mathcal{Z}_{\mathbb{H}}(y) \in \mathbb{F}_{\leq 2|\mathbb{K}|-2}[X]$$

that equals 0 on any $\kappa \in \mathbb{K}$. This way, $\mathcal{P}$ computes $q'(X) := \frac{t(X)}{\mathcal{Z}_{\mathbb{K}}(X)} \in \mathbb{F}_{\leq |\mathbb{K}|-2}[X]$ and sends σ and $\{q'(X), r'(X)\}$ to $\mathcal{V}$.

Decision phase. The verifier outputs the following degree checks

$$\mathsf{deg}(\hat{a}'), \mathsf{deg}(\hat{b}'), \mathsf{deg}(s), \mathsf{deg}(q), \mathsf{deg}(q') \overset{?}{\leq} \mathsf{D}_{\mathsf{snd}} \tag{3}$$
$$\mathsf{deg}(r) \overset{?}{\leq} n - 2 \tag{4}$$
$$\mathsf{deg}(r') \overset{?}{\leq} |\mathbb{K}| - 2 \tag{5}$$

and the following two polynomial checks:

$$s(y) + \Big(\hat{a}'(y) \cdot \mathcal{Z}_{\mathbb{L}}(y) + \sum_{\eta \in \mathbb{L}} \boldsymbol{x'}_{\phi_{\mathbb{H}}(\eta)} \cdot \mathcal{L}^{\mathbb{H}}_{\eta}(y) \Big) \Big(\Lambda_{\mathbb{H}}(x,y) + \sigma (\hat{b}'(y) \cdot \mathcal{Z}_{\mathbb{L}}(y) + 1) \Big)$$
$$+ \ (\hat{b}'(y) \cdot \mathcal{Z}_{\mathbb{L}}(y) + 1) \cdot \alpha \cdot \Lambda_{\mathbb{H}}(x,y) - q(y)\, \mathcal{Z}_{\mathbb{H}}(y) - y\, r(y) \stackrel{?}{=} 0 \quad (6)$$

$$\frac{\sigma}{|\mathbb{K}|} \cdot n^2 \cdot (xy + \mathsf{cr}(X) - x \cdot \mathsf{col}(X) - y \cdot \mathsf{row}(X))$$
$$+ \ r'(X) \cdot n^2 \cdot (xy \cdot X + \mathsf{cr}'(X) - x \cdot \mathsf{col}'(X) - y \cdot \mathsf{row}'(X))$$
$$- \ (\mathsf{vcr}_L(X) + \alpha \cdot \mathsf{vcr}_R(X)) \cdot \mathcal{Z}_{\mathbb{H}}(x) \cdot \mathcal{Z}_{\mathbb{H}}(y) - q'(X) \cdot \mathcal{Z}_{\mathbb{K}}(X) \stackrel{?}{=} 0 \quad (7)$$

Above, we highlight the oracle polynomials in *gray*, the prover messages in blue, and the coefficients of the verifier's polynomial checks in red. This is to help seeing how the above checks fit the form described in Definition 2.

In the first degree check, $\mathsf{D}_{\mathsf{snd}}$ is an integer that can be chosen by the verifier and governs the soundness error as shown in Theorem 1. While for correctness we need $\mathsf{D}_{\mathsf{snd}} \geq \mathsf{D} - 1$, where D is the degree of the PHP (shown below), this bound does not need to be tight (i.e., $\mathsf{D}_{\mathsf{snd}} = \mathsf{D} - 1$) as is the case for the degree checks on r and r'. This observation has an impact on our compiler where, by choosing $\mathsf{D}_{\mathsf{snd}}$ to be the maximal degree supported by the commitment scheme, one does not need to create a proof for degree checks of the form "$\leq \mathsf{D}_{\mathsf{snd}}$".

Security Analysis. We state knowledge soundness and zero-knowledge of $\mathsf{PHP}_{\mathsf{lite1}}$; full proofs are in the full version.

Theorem 1 (Knowledge Soundness). *Our protocol* $\mathsf{PHP}_{\mathsf{lite1}}$ *is* ϵ*-knowledge-sound with* $\epsilon = \frac{|\mathbb{H}|}{|\mathbb{F}|} + \frac{2\mathsf{D}_{\mathsf{snd}} + |\mathbb{H}|}{|\mathbb{F} \setminus \mathbb{H}|}$.

Theorem 2 (Zero-Knowledge). *Our PHP protocol* $\mathsf{PHP}_{\mathsf{lite1}}$ *is perfect zero-knowledge. Furthermore, it is perfect* b*-HVZK with* $\mathsf{b} = (\mathsf{b}_a, \mathsf{b}_b, \mathsf{b}_s, \mathsf{b}_q, \mathsf{b}_r, \infty, \infty)$.

For an intuition about soundness we refer to the intuitive description of the construction. For b-HVZK, we present the main ideas. Following a rather standard argument, we have that up to b_a (resp. b_b) evaluations of $\hat{a}'$ (resp. $\hat{b}'$) are randomly distributed due to their construction through Mask. Instead, up to b_q (resp. b_r) evaluations of q (resp. r) can be argued random thanks to the randomness of the polynomials q_s and r_s defining $s(X) = q_s(X) \cdot \mathcal{Z}_{\mathbb{H}}(X) + X \cdot r_s(X)$. In particular, this uses that for $\gamma \in \mathbb{F} \setminus \mathbb{H}$, $s(X)$ is $(\mathsf{b}_s + \mathsf{b}_q)$-wise independent even conditioned on $r_s(X)$, and that the honest $q(X)$ is determined by $(p(X) + s(X) - Xr(X))/\mathcal{Z}_{\mathbb{H}}(X)$, where $p(X)$ is that defined in round 2.

Remark 2 ($\mathsf{PHP}_{\mathsf{lite1x}}$*: a variant with fewer relation polynomials).* We present a variant of $\mathsf{PHP}_{\mathsf{lite1}}$, that we call $\mathsf{PHP}_{\mathsf{lite1x}}$, which has fewer relation polynomials.

In particular, the $\mathcal{RE}$ of $\mathsf{PHP}_{\mathsf{lite1x}}$ does not output $\mathsf{col}'(X), \mathsf{row}'(X)$ and $\mathsf{cr}'(X)$, and the second polynomial check, of degree 3 with a public term X, becomes:

$$n^2 \cdot \left(X \cdot r'(X) + \frac{\sigma}{|\mathbb{K}|} \right) \cdot \left(xy + \mathsf{cr}(X) - x \cdot \mathsf{col}(X) - y \cdot \mathsf{row}(X) \right) - \left(\mathsf{vcr}_L(X) + \alpha \cdot \mathsf{vcr}_R(X) \right) \cdot \mathcal{Z}_{\mathbb{H}}(x) \cdot \mathcal{Z}_{\mathbb{H}}(y) - q'(X) \cdot \mathcal{Z}_{\mathbb{K}}(X) \stackrel{?}{=} 0 \quad (8)$$

$\mathsf{PHP}_{\mathsf{lite2}}$: Separate Sparse Matrix Encodings. We propose another PHP for R1CS-lite called $\mathsf{PHP}_{\mathsf{lite2}}$. $\mathsf{PHP}_{\mathsf{lite2}}$ is very similar to $\mathsf{PHP}_{\mathsf{lite1}}$, indeed its first two rounds of the online phase are identical. The main difference is that in $\mathsf{PHP}_{\mathsf{lite2}}$ the matrices $\{\boldsymbol{L}, \boldsymbol{R}\}$ are encoded in sparse form separately. Namely, $\boldsymbol{L}, \boldsymbol{R}$ are represented with the functions $\{\mathsf{val}_M, \mathsf{row}_M, \mathsf{col}_M\}_{M \in \{L,R\}}$ so that, for any $\kappa \in \mathbb{K}$, $\mathsf{val}_M(\kappa) = \boldsymbol{M}_{\mathsf{row}_M(\kappa), \mathsf{col}_M(\kappa)}$. The main benefit of this choice is that we can work with a subgroup $\mathbb{K} \subset \mathbb{F}$ such that $|\mathbb{K}| \geq m \geq \max\{||\boldsymbol{L}||, ||\boldsymbol{R}||\}$, which is half the size of the one needed in $\mathsf{PHP}_{\mathsf{lite1}}$. Using this encoding, the $V_{LR}(X, Y, Z)$ polynomial in equation (2) here becomes

$$\sum_{\kappa \in \mathbb{K}} \left(\mathsf{val}_L(\kappa) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}_L(\kappa)}(X) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{col}_L(\kappa)}(Y) + Z \cdot \mathsf{val}_R(\kappa) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}_R(\kappa)}(X) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{col}_R(\kappa)}(Y) \right)$$

So, in round 3 of $\mathsf{PHP}_{\mathsf{lite2}}$ the prover's goal is to show that $\sigma = V_{LR}(x, y, \alpha)$ for the equation above. This is done analogously to $\mathsf{PHP}_{\mathsf{lite1}}$ except that here $\{\mathsf{val}_M, \mathsf{row}_M, \mathsf{col}_M\}_{M \in \{L,R\}}$ are expanded in a total of 24 relation polynomials for the goal of keeping 2 the degree of the second polynomial check. See Table 2 for a summary of $\mathsf{PHP}_{\mathsf{lite2}}$ measures and its variant $\mathsf{PHP}_{\mathsf{lite2x}}$, and the full version for a detailed description.

5 Compiler from PHPs to Universal zkSNARKs

We start with the definitions for our compiler. Some of the following notions are standard or were introduced in previous works, while some others are new. For space reasons, we defer to the full version for formal definitions .

Commitment Schemes. In our work we use the notion of *type-based commitments*, introduced by Escala and Groth [22]: these are a generalization of regular commitments that unify several committing methods into the same scheme. As done in [11], in this work we exploit the formalism of type-based commitments to describe commit-and-prove zero-knowledge proofs that work with commitments of different types, tailoring different properties for the same message space.

More in detail, a type-based commitment scheme is a tuple of algorithms $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCom})$ that works as a commitment scheme with the difference that the Commit and VerCom algorithms take an extra input type that represents the type of c. All the possible types are included in the type space $\mathcal{T}$. Having different types helps for a more granular description of the security properties of the commitment scheme. For example, a commitment scheme for

a set of types $\{\mathsf{type}_1, \mathsf{type}_2\}$ could be trapdoor hiding for commitments of type type_1 and could be computationally hiding for commitments of type type_2. In this case, we say that the commitment scheme is type_1-trapdoor hiding and type_2-computationally hiding. We assume succinct commitments.

zkSNARKs with Universal and Specializable SRS. A zkSNARK with specializable universal SRS for a family of relations $\{\mathcal{R}_\mathsf{N}\}_{\mathsf{N}\in\mathbb{N}}$, introduced by Groth et al. [34], is a tuple of algorithms $\Pi = (\mathsf{KeyGen}, \mathsf{Derive}, \mathsf{Prove}, \mathsf{Verify})$ where KeyGen is probabilistic and upon input public parameters and size bound N produces the srs and a trapdoor td_k, Derive is deterministic and upon input srs and $\mathsf{R} \in \mathcal{R}_\mathsf{N}$ produces $\mathsf{ek}_\mathsf{R}, \mathsf{vk}_\mathsf{R}$, and the prover Prove and verifier Verify act as usual. We require the standard notions of completeness, succinctness, knowledge-soundness and zero-knowledge.

Universal CP-SNARKs. We adapt the notion of commit-and-prove SNARKs of [18] to universal relations. Very roughly speaking, a universal CP-SNARK for a family of relationships $\mathcal{R}$ and a commitment scheme CS is a universal SNARK for a family of relations $\mathcal{R}^{\mathsf{Com}}$ which includes all the relations $\mathsf{R}^{\mathsf{Com}}$ such that $\mathsf{R}^{\mathsf{Com}}(\mathsf{x}, c, \mathsf{w}) = 1$ if and only if $\mathsf{R}(\mathsf{x}, \mathsf{w}) = 1$ and c *is a commitment that opens to* w and $\mathsf{R} \in \mathcal{R}$. As in [18], in the definition we add syntactic sugar to this idea to handle relations where the domain of the witness is more fine grained and split over $\ell + 1$ subdomains for a fixed $\ell \in \mathbb{N}$.

More in detail, we denote a universal CP-SNARK as a tuple of algorithms $\mathsf{CP} = (\mathsf{KeyGen}, \mathsf{Derive}, \mathsf{Prove}, \mathsf{Verify})$ where: (i) $\mathsf{KeyGen}(\mathsf{ck}, \mathsf{N}) \to \mathsf{srs} := (\mathsf{ek}, \mathsf{vk})$ generates the structured reference string. (ii) $\mathsf{Derive}(\mathsf{srs}, \mathsf{R}) \to (\mathsf{ek}_\mathsf{R}, \mathsf{vk}_\mathsf{R})$ is a deterministic algorithm that takes as input an srs produced by $\mathsf{KeyGen}(\mathsf{ck}, \mathsf{N})$, and a relation $\mathsf{R} \in \mathcal{R}_\mathsf{N}$. (iii) $\mathsf{Prove}(\mathsf{ek}, \mathsf{x}, (c_j)_{j\in[\ell]}, (\mathsf{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega) \to \pi$ outputs the proof for $(\mathsf{x}, \mathsf{w}) \in \mathsf{R}$ and $\mathsf{w} = (\mathsf{u}_1, \ldots, \mathsf{u}_\ell, \omega)$. (iv) $\mathsf{Verify}(\mathsf{vk}_\mathsf{R}, \mathsf{x}, (c_j)_{j\in[\ell]}, \pi) \to \{0, 1\}$ rejects or accepts the proof. Sometimes we use a more general notion of knowledge soundness for CP-SNARKs introduced by Benarroch *et al.* [11] named *knowledge soundness with partial opening*. The intuition is to consider adversaries that explicitly return valid openings for a subset of the commitments that they return, thus enabling to formally define knowledge soundness in the context where not all the commitments need to be extracted.

In the basic completeness notion of Universal CP-SNARKs, the CP-SNARK is required to work with commitments of any type. We also define a weaker notion of completeness in which the CP-SNARK works only when certain witnesses are committed with a specific type. We call this notion T*-restricted completeness*. This is useful if we want to use a CP-SNARK that supports only a subset T of the types of the commitment scheme. We give a few examples. Suppose the commitment scheme has two different types, $\mathsf{type}_1, \mathsf{type}_2$, and there exists a CP-SNARK that only works with commitments of type_1. Alternatively, a CP-SNARK for a relation with $\ell_1 + \ell_2$ committed witnesses could work only when the first ℓ_1 commitments are of type type_1 and the subsequent ℓ_2 commitments are of type type_2. And clearly, more fine-grained combinations are possible.

Commitment-only SRS. We say that a universal CP-SNARK has a *commitment-only* SRS if the key generation algorithm is deterministic. Notice that for Universal CP-SNARK with commitment-only SRS the notion of zero-knowledge in the SRS model is not achievable. In fact, formally speaking, the commitment key ck is part of the description of a relation; thus, the actual SRS of the CP-SNARK would be the empty string. However, the classical result of [30] shows that NIZK in the plain model exists only for trivial languages. Therefore we consider a weaker notion of zero-knowledge for these CP-SNARKs, that we call trapdoor-commitment zero-knowledge in the SRS model , where the trapdoor necessary for simulation comes from the commitment key of CS.

5.1 Compiler's Building Blocks

Commitments to Polynomials. Recall that a PHP verifier has access to two sets of oracle polynomials: those from the relation encoder (which describe the relation) and those from the prover (which should supposedly convince the verifier to accept a public input x). The compiler commits to polynomials in both sets; it requires all these commitments to be binding, but not to fully hide any of these polynomials.

The commitments for the relation encoding polynomials—whose type we denote by `rel`—do not need to hide anything: they open to polynomials representing the relation, which is public information. The polynomial commitments of type `rel` have weaker requirements for one more reason. Besides not requiring them to be hiding, we will not require them to be extractable (i.e., we do not assume a CP-SNARK that has knowledge soundness for them, here is the reason to use the notion of knowledge soundness with partial opening).

Above, we ignored leakage when committing to relation encoding polynomials; we cannot do the same when committing to the polynomials from the PHP prover as they contain information about the witness. If we do not prevent *some* leakage we will lose zero-knowledge. At the same time we will show that we do not need full hiding for these polynomials either, just a relaxed property that may hold even for a deterministic commitment algorithm. We call this property *somewhat-hiding*—defined below— and denote its type by `swh`.

In the remainder of this section we will assume CS to be a polynomial commitment scheme; i.e., a commitment scheme in which the message space $\mathcal{M}$ is $\mathbb{F}_{\leq d}[X]$ for a finite field $\mathbb{F} \in \mathcal{F}$ and an integer $d \in \mathbb{N}$. Without loss of generality we assume d to be an input parameter of Setup.

Definition 6 (Somewhat-Hiding Polynomial Commitments). *Let* $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCom})$ *be a type-based commitment scheme for a class of polynomials* $\mathbb{F}_{\leq d}[X]$ *and a class of types* $\mathcal{T}$*, and that works as in Type-Based Commitment Schemes, but where we allow* Commit *to be deterministic. Then* CS *is said to be* type-typed somewhat-hiding *if there exist three algorithms* $(\mathsf{ck}, \mathsf{td} = (\mathsf{td}', s)) \leftarrow \mathcal{S}_{\mathsf{ck}}(s)$ *where* $s \in \mathbb{F}$*,* $(c, st) \leftarrow \mathsf{TdCom}(\mathsf{td}, \gamma)$ *and* $o \leftarrow \mathsf{TdOpen}(\mathsf{td}, st, c, f)$ *such that: (1) the distribution of the commitment key returned by* $\mathcal{S}_{\mathsf{ck}}$ *with a uniformly random* $s \leftarrow_{\$} \mathbb{F}$ *as input is identical to the one of the key returned by* Setup*;*

(2) for any $f \in \mathbb{F}_{<d}[X]$, $(c, o) \approx (c', o')$ *where* $(c, o) \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathsf{type}, f)$, $(c', st) \leftarrow \mathsf{TdCom}(\mathsf{td}, f(s))$ *and* $o' \leftarrow \mathsf{TdOpen}(\mathsf{td}, st, c', f)$.

CP-SNARKs for the Commitment Scheme. We assume that the commitment scheme CS is equipped with a CP-SNARK $\mathsf{CP}_{\mathsf{php}} = (\mathsf{KeyGen}_{\mathsf{php}}, \mathsf{Prove}_{\mathsf{php}}, \mathsf{Verify}_{\mathsf{php}})$ for a relation family $\mathcal{R}' \supseteq \mathcal{R}_{\mathsf{php}}$ (we defined $\mathcal{R}_{\mathsf{php}}$ in Section 3.1), and with a CP-SNARK $\mathsf{CP}_{\mathsf{opn}} = (\mathsf{KeyGen}_{\mathsf{opn}}, \mathsf{Prove}_{\mathsf{opn}}, \mathsf{Verify}_{\mathsf{opn}})$ for the (trivial) relation family $\mathcal{R}_{\mathsf{opn}} = \{\psi, (p_j)_{j\in[\ell]} : \ell \in \mathbb{N}\}$ whose instance is the empty string ψ and witnesses are tuples of polynomials. A CP-SNARK for $\mathcal{R}_{\mathsf{opn}}$ is essentially a *proof of knowledge* of the openings of ℓ commitments.

Leaky Zero-Knowledge. We define a weaker zero-knowledge notion that is sufficient to be satisfied by the $\mathsf{CP}_{\mathsf{php}}$ CP-SNARK in our compiler. This new property allows better efficiency and flexibility of the compiled protocols. Intuitively, a CP-SNARK for relations over committed polynomials is *leaky zero-knowledge* if its proofs may leak information about a bounded number of evaluations of these polynomials. More in detail, a CP-SNARK is (b, C)-leaky zero-knowledge if there exists a ZK simulator that has access to a list of leaked values $\{\mathsf{u}_{i_j}(y_j)\}_{(i,j)}$ where the list $\{(i_j, y_j)\}_{j\in\mathbb{N}}$ is (b, C)-bounded (see Section 3).

5.2 The Compiler

At a high level, we follow the known paradigm stemming from [40, 46] in which the prover commits to the oracles, answers the verifier's queries generated using a random oracle and proves correctness of these answers. A high-level description of the compiled SNARK $\Pi = (\mathsf{KeyGen}, \mathsf{Derive}, \mathsf{Prove}, \mathsf{Verify})$ follows:

- The KeyGen algorithm runs the setup of the commitment scheme CS and generates keys for the auxiliary CP-SNARKs.
- The Derive algorithm, when deriving a specialized SRS for a specific relation R, commits to all the polynomials returned by the relation encoder $\mathcal{RE}(\mathsf{R})$ using `rel`-typed commitments.
- The prover Prove algorithm executes internally the PHP prover $\mathcal{P}$, at each round of $\mathcal{P}$ it commits the polynomials from $\mathcal{P}$ using `swh`-typed commitments; it proves it knows their opening using $\mathsf{CP}_{\mathsf{opn}}$; concatenate the commitments, the proofs and the rest of the messages from $\mathcal{P}$. It computes a hash of the partial transcript, which it then uses as the next message to feed to the $\mathcal{P}$. At the last round it uses $\mathsf{CP}_{\mathsf{php}}$ to prove that the PHP verifier $\mathcal{V}$ would accept.
- The verifier checks all the CP-SNARK proofs of opening for the commitments and executes the decision stage of $\mathcal{V}$ with input the instance and the random oracle hash values computed over the partial transcripts. It thus generates an instance for $\mathsf{CP}_{\mathsf{php}}$ and checks the related CP-SNARK proof.

For compactness in the exposition, we state the main result of the section in one theorem, however in the full version we restate the theorem in two steps: first we compile to universal *interactive* argument systems, and secondly we compile the latter argument systems to SNARKs using the Fiat-Shamir transform—thus the following theorem holds in the random oracle model.

Theorem 3. *Let* $\mathsf{PHP} = (\mathsf{r}, \mathsf{n}, \mathsf{m}, \mathsf{d}, \mathsf{n_e}, \mathcal{RE}, \mathcal{P}, \mathcal{V})$ *be a non-adaptive public-coin PHP over a finite field family* $\mathcal{F}$ *and for a universal relation* $\mathcal{R}$*. Let* CS *be a type-based commitment scheme for a class of polynomials* $\mathbb{F}_{<d}[X]$ *and a class of types* $\mathcal{T} = \{\mathtt{rel}, \mathtt{swh}\}$ *that is* $\mathcal{T}$*-binding and* $\mathtt{swh}$*-somewhat-hiding and equipped with CP-SNARKs* $\mathsf{CP}_{\mathsf{opn}}$ *for* $\mathcal{R}_{\mathsf{opn}}$ *and* $\mathsf{CP}_{\mathsf{php}}$ *for* $\mathcal{R}_{\mathsf{php}}$*.*

- *The scheme* $\Pi = (\mathsf{KeyGen}, \mathsf{Derive}, \mathsf{Prove}, \mathsf{Verify})$ *is a zkSNARK with specializable universal SRS for the family of relations* $\mathcal{R}$*.*
- *If* $\mathsf{CP}_{\mathsf{opn}}$ *is TP-ZK, and, for a checker* C*,* PHP *(resp.* $\mathsf{CP}_{\mathsf{php}}$*) is* $(\mathsf{b} + \mathbf{1}, \mathsf{C})$*-bounded honest-verifier zero-knowledge (resp.* (b, C)*-leaky zero-knowledge) then* Π *is zero-knowledge in the SRS model.*

Remark 3 (On completeness). It is sufficient for $\mathsf{CP}_{\mathsf{php}}$ to be T-restricted complete, with $T = \left((\mathtt{rel})^{\mathsf{n}(0)} \| (\mathtt{swh})^{\mathsf{n_p}}\right) \in \mathcal{T}^{\mathsf{n}^*}$, to obtain the completeness of Π.

Remark 4 (On updatable SRS). If the commitment key generated by Setup is updatable [**?**, 34], and $\mathsf{CP}_{\mathsf{opn}}$ and $\mathsf{CP}_{\mathsf{php}}$ have commitment-only SRS then the SRS of Π is updatable.

Intuition on Security Proof. The proof of knowledge soundness follows the standard argument of simulating a prover for the PHP extracting the polynomials from the commitments sent by the adversary and use the binding property of the commitments together with the knowledge soundness of $\mathsf{CP}_{\mathsf{php}}$ to prove that the verifier of the PHP protocol would indeed accept.

We now provide an intuition about zero-knowledge; for simplicity we describe it as if the protocol involved a single committed polynomial. First, observe that we assume a PHP with $b+1$-bounded ZK—i.e., we can simulate interaction with an honest prover even after we have leaked $b+1$ evaluations of the polynomial. Since we assume a commitment scheme that is only somewhat-hiding (Definition 6), we are actually leaking one evaluation of the committed polynomial (in particular on a random point). We now combine this fact with the ZK property we are assuming on the CP-SNARKs in the compiler—b-leaky ZK— and this allows us to still simulate an interaction with an honest prover that is indistinguishable after further b leaked evaluations.

Compiler to Universal CP-SNARK. We briefly explain how to adapt our compiler to turn PHPs into CP-SNARKs. More details appear in the full version.

We consider a natural sub-class of PHP where the extractor for the knowledge soundness satisfies a stronger property usually denoted as *straight-line extractability* in the literature. In particular, we assume there exists an extractor $\mathsf{WitExtract}$ that on input the polynomials sent by a malicious prover interacting with the verifier can extract the valid witness.

Recall that the instances for CP-SNARKs are tuples of the form $(\mathsf{x}, \hat{c}_1, \ldots, \hat{c}_\ell)$ for a value $\ell \in \mathbb{N}$, where x is an instance for the relation and $\hat{c}_1, \ldots, \hat{c}_\ell$ commits to chunks of the witness. The commitments $\hat{c}_1, \ldots, \hat{c}_\ell$ are just classical commitments (in the sense that they are hiding and binding, but there are no restrictions on

other properties they might have). Therefore we consider CP-SNARKs for typed-commitment schemes with class of types $\mathcal{T} = \{\mathtt{rel}, \mathtt{swh}, \mathtt{lnk}\}$, where the latter type is reserved for the input commitments (and thus the commitment scheme is `lnk`-typed hiding and `lnk`-typed binding).

The compiler to a CP-SNARK is exactly the same as the compiler presented before but where the prover, after having computed all the commitments $c_1, \ldots, c_{\mathsf{n_p}}$ (and the proofs for $\mathsf{CP_{opn}}$ and $\mathsf{CP_{php}}$), additionally computes a CP-SNARK proof for the relation $\mathcal{R}_{\mathsf{link}}$ that says that the commitments $\hat{c}_1, \ldots, \hat{c}_\ell$ open to values $\mathsf{u}_1, \ldots, \mathsf{u}_\ell$ and the commitments $c_1, \ldots, c_{\mathsf{n_p}}$ open to polynomials $p_1, \ldots, p_{\mathsf{n_p}}$ such that $\mathsf{WitExtract}(p_1, \ldots, p_{\mathsf{n_p}}) = (\mathsf{u}_1, \ldots, \mathsf{u}_\ell, \omega)$, therefore creating a link between the computed proof and the input commitments $\hat{c}_1, \ldots, \hat{c}_\ell$.

6 Instantiating Our Compiler: Our Universal zkSNARKs

We propose different instantiations of the building blocks needed by our compiler of Section 5: *(i)* (type-based) pairing-based commitment schemes for polynomials; *(ii)* a collection of CP-SNARKs for various relations over such committed polynomials. Next, we describe different options to combine them together in our compiler, when applied to our PHP constructions (see Table 2). The resulting zkSNARKs offer different tradeoffs in terms of SRS size, proof size, and verification time. Table 1 summarizes the most interesting among these schemes.

We denote a bilinear group setting by a tuple $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where $\mathbb{G}_1, \mathbb{G}_2$, $\mathbb{G}_T$ are additive groups of prime order q, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable, non-degenerate, bilinear map. We focus on Type-3 groups and use the bracket notation of [23], i.e., for $g \in \{1, 2, T\}$ and $a \in \mathbb{Z}_q$, we write $[a]_g$ to denote $a \cdot P_g \in \mathbb{G}_g$, where P_g is a fixed generator of $\mathbb{G}_g$.

6.1 Pairing-Based Commitment Schemes for Polynomials

We show two type-based commitment schemes, denoted CS_1 and CS_2 respectively, with type set $\{\mathtt{rel}, \mathtt{swh}\}$ and for degree-d polynomials. The commitment of a polynomial p is essentially the *evaluation in the exponent* of p in a secret point s, following the scheme of Groth [32] and Kate et al. [39]. Slightly more in detail, in both schemes, the commitment key ck contains encodings of powers of a secret point s, and a commitment of type `swh` to a polynomial $p(X)$ is a group element $[p(s)]_1$. The only difference between the two schemes are the commitments of type `rel`, which in CS_1 are $[p(s)]_1$ whereas in CS_2 are $[p(s)]_2$. As discussed in the next section, the advantage of having some polynomials committed in $\mathbb{G}_2$ is that one immediately gets a way to test quadratic equations over committed polynomials where each quadratic term involves exactly one polynomial of type `rel`. Both types of commitments are computationally binding under the power-discrete logarithm assumption [44]; we prove commitments of type `swh` to also be somewhat hiding.

Remark 5 (On updatability of our SNARKs). Since the commitment schemes CS_1 and CS_2 that we work upon generate keys that only contain monomials

in the exponent, our constructions are updatable in the sense that participants can easily re-randomize them at will. Pointing to previous works on updatable SNARKs, *"a CRS that consists solely of monomials (...) is updatable"* [34].

6.2 Pairing-Based CP-SNARKs for CS_1 and CS_2

We show CP-SNARKs for various relations over polynomials committed using CS_1 or CS_2. Our CP-SNARKs work over both commitment schemes unless explicitly stated otherwise. A full description of these schemes is in the full version.

Proof of Knowledge: "I know p and c opens to p". We show two schemes. (i) $\mathsf{CP}_{\mathsf{opn}}^{\mathsf{AGM}}$ is a trivial scheme in which the proof is the empty string and is knowledge-sound in the algebraic group model [25]; this is an observation already done in previous work, e.g., [19, 27]. (ii) $\mathsf{CP}_{\mathsf{opn}}^{\mathsf{PKE}}$, is *novel* and provides extractability based on the mPKE assumption and, when used on more than one commitment, on the random oracle heuristic. In a nutshell, this scheme uses the classical technique of giving as a proof a group element $\pi_{\mathsf{opn}} = \gamma \cdot c$, where $\gamma \in \mathbb{F}$ is a secret but such that π_{opn} can be publicly computed if one knows the opening of c. What is new in our scheme is a way to batch this proof for ℓ commitments in such a way that we have only one extra group element as a proof, instead of ℓ elements.

Polynomials Evaluation: "$\big(p_i(x_i) = y_i\big)_{i\in[\ell]}$". We first give a CP-SNARK for single polynomial evaluation—"$p(x) = y$"—$\mathsf{CP}_{\mathsf{eval},1}$, secure under the d-SDH assumption and the extractability of $\mathsf{CP}_{\mathsf{opn}}$, and then we extend it into a CP-SNARK $\mathsf{CP}_{\mathsf{eval}}$ to support batching. Both schemes stem from techniques in [39].

Polynomial Equations: A CP-SNARK $\mathsf{CP}_{\mathsf{eq}}$ for general polynomial equations, e.g., $a(X)b(X) - 2c(X)d(X)e(X) = 0$), relying mainly on $\mathsf{CP}_{\mathsf{opn}}$ and $\mathsf{CP}_{\mathsf{eval}}$. It is based on the idea of doing evaluations on a random point, with optimizations from [27], based on the linearity of the commitment, to minimize proof size.

Quadratic Polynomial Equations: A novel CP-SNARK for quadratic polynomial equations[9] specific to commitment scheme CS_2; although less general than $\mathsf{CP}_{\mathsf{eq}}$, $\mathsf{CP}_{\mathsf{qeq}}$ is more efficient since its proof may simply be empty, while verification consists of some pairing checks over the commitments. The basic intuition is simple: to check that $G(p_1(X), \ldots, p_\ell(X)) = 0$ for a quadratic polynomial G it is possible to homomorphically compute G over the values $(p_1(s), \ldots, p_\ell(s))$ in the target group using pairings and the linear property of the commitments. For this to be possible, for each quadratic monomial $p_i(X)p_j(X)$, we need at least one of $[p_i(s)]_2$ or $[p_j(s)]_2$ in $\mathbb{G}_2$. This holds if they are committed through different types, i.e., one as `rel` and the other as `swh`. Otherwise, if they are both in the same group, we let the prover create one of the two polynomials committed in the "symmetric" group. Interestingly, for carefully designed equations, the $\mathsf{CP}_{\mathsf{qeq}}$ proof can be empty and all the verifier needs to do is verifying a pairing product.

Degree Bound: "$(\mathsf{deg}(p_i) \le d_i)_{i\in[\ell]}$". Two CP-SNARKs—$\mathsf{CP}_{\mathsf{deg}}^{(\star)}$ and $\mathsf{CP}_{\mathsf{deg}}^{(2)}$—such that $\mathsf{CP}_{\mathsf{deg}}^{(\star)}$ works over both commitment schemes while $\mathsf{CP}_{\mathsf{deg}}^{(2)}$ works only over

[9] Here "quadratic" means it supports products of at most two polynomials.

CS_2. The basic idea is to commit to the shifted polynomial $p^*(X) = X^{D-d}p(X)$ and then prove that the polynomial equation $X^{D-d} \cdot p(X) - p^*(X) = 0$ using a CP-SNARK for polynomial equations, either $\mathsf{CP}_{\mathsf{eq}}$ or $\mathsf{CP}_{\mathsf{qeq}}$. This idea is extended in order to batch together these proofs for several polynomials.

6.3 Available Options to Compile Our PHPs

We discuss how to combine the aforementioned CP-SNARKs for committed polynomials to obtain CP-SNARKs for the $\mathcal{R}_{\mathsf{php}}$ relations corresponding to our PHPs. All our PHPs have a similar structure in which the verifier checks consist of one vector $\boldsymbol{d}$ of degree checks, and two polynomial checks $((G'_1, \boldsymbol{v}_1), (G'_2, \boldsymbol{v}_2))$. Hence, for each PHP the corresponding relation $\mathcal{R}_{\mathsf{php}}$ can be expressed as:

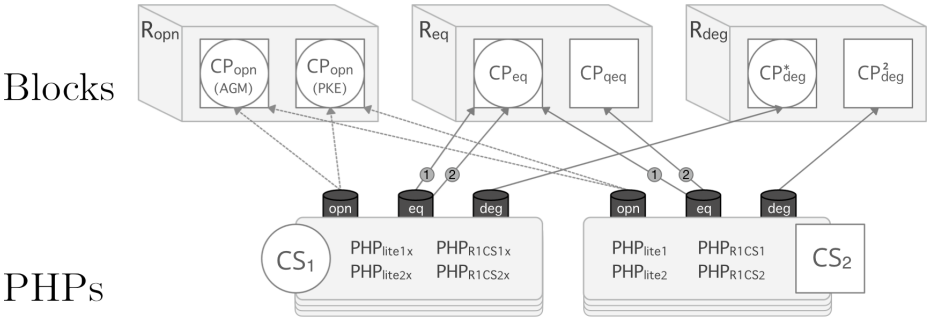
$$\mathsf{R}_{\mathsf{deg}}((d_j)_{j\in[\mathsf{n_p}]}, (p_j)_{j\in[\mathsf{n}(0)+1,\mathsf{n}^*]}) \ \wedge \ \left\{\mathsf{R}_{\mathsf{eq}}((G'_i, \boldsymbol{v_i}), (p_j)_{j\in[\mathsf{n}^*]})\right\}_{i\in\{1,2\}}$$

where G'_i is the partial evaluation of G_i on the prover message σ.

In all the PHPs, in the first polynomial check the $\boldsymbol{v}_{1,j}(X)$ are constant polynomials (in particular, they all encode the same point, i.e., $\forall j : \boldsymbol{v}_{1,j}(X) = y$), while in the second check they are the identity, i.e., $\forall j : \boldsymbol{v}_{2,j}(X) = X$. Furthermore, in those PHPs where $\mathsf{deg}_{X,\{X_i\}}(G_2) = 2$, the second R_{eq} relation can be replaced by its specialization for quadratic equations.

We use two main compilation options for our PHPs (outlined in Figure 1):

Fig. 1. Different options to compile our PHPs. We mark compatibility with commitment schemes CS_1 and CS_2 respectively by a circle and a square (both shapes mean full compatibility). Dotted lines mean either option is possible. An index 1 or 2 for an arrow to R_{eq} denotes whether it refers to the first or second polynomial check.



6.4 Zero-knowledge Bounds when Instantiating PHPs

Our compiler assumes a CP-SNARK $\mathsf{CP}_{\mathsf{php}}$ that can be (b, C)-leaky-ZK to compile a PHP protocol that is $(\mathbf{1} + \mathsf{b})$-bounded ZK (see Theorem 3), as the commitments reveals one evaluation per oracle polynomial.Among the CP-SNARKs

we propose to realize $\mathsf{CP}_{\mathsf{php}}$, the only one that is leaky-ZK is the $\mathsf{CP}_{\mathsf{eq}}$ scheme. Its leakage is due to the fact that the proof includes evaluations of those polynomials that end up in the set S used to optimize the proof size. Note that this concern arises only when using it to prove the first polynomial check. Indeed, in all of our schemes *the second polynomial check involves only oracle polynomials that are not related to the witness*, and thus for those polynomials the amount of leakage does not matter. We discuss how to choose b for the b-leaky-ZK of $\mathsf{CP}_{\mathsf{eq}}$ when proving the first polynomial check in all of our PHPs.

PHPs for R1CS-lite. The first polynomial check is the same in both constructions. Through the syntax for relation R_{eq} we write the polynomial G'_1 as

$$G'_1(X_a, X_b, X_s, X_q, X_r) := X_a \cdot X_b \cdot g_{a,b} + X_a \cdot g_a + X_b \cdot g_b + X_q \cdot g_q + X_r \cdot g_r + X_s + g_0$$

where the goal is to prove that on a given y, $G'_1((p_j(y))_{j\in[5]}) = 0$, that is:

$$\hat{a}'(y)\hat{b}'(y) \cdot g_{a,b} + \hat{a}'(y) \cdot g_a + \hat{b}'(y) \cdot g_b + s(y) + q(y) \cdot g_q + r(y) \cdot g_r + g_0 \stackrel{?}{=} 0$$

To this end, $\mathsf{CP}_{\mathsf{eq}}$ chooses a set S of size 1; for instance it reveals $\hat{b}'(y)$ and nothing more. Thus, $\mathsf{CP}_{\mathsf{eq}}$ for this polynomial check is b-leaky-ZK with $\mathsf{b} = (\mathsf{b}_a, \mathsf{b}_b, \mathsf{b}_s, \mathsf{b}_q, \mathsf{b}_r) = (0, 1, 0, 0, 0)$. From Theorem 3, $\mathsf{PHP}_{\mathsf{lite1}}$ and $\mathsf{PHP}_{\mathsf{lite2}}$ need to be $(1, 2, 1, 1, 1)$-bounded ZK, and we can optimize the degrees and instantiate $\mathsf{PHP}_{\mathsf{lite}*}$ with $\hat{a}' \in \mathbb{F}_{\leq n+1}[X]$, $\hat{b}' \in \mathbb{F}_{\leq n+2}[X]$, $q_s \in \mathbb{F}_{\leq 1}[X]$, $r_s \in \mathbb{F}_{\leq 1}[X]$.

PHPs for R1CS. All these constructions need to be $(1, 2, 1, 1, 1, 1)$-bounded ZK. The analysis is the same as for R1CS-lite; we omit details for lack of space.

6.5 Our Resulting zkSNARKs and CP-SNARKs

In the full version we provide a table with the efficiency of all the zkSNARKs obtained through the different options to instantiate the compiler on all of our PHPs. We also discuss how those measures are obtained and give the costs for the CP-SNARKs resulting from the commit-and-prove compiler. We recall that the most representative zkSNARKs (in the algebraic group model) are shown in Table 1 together with a comparison with the state of the art. We recall that all our constructions are universal and updatable.

We note that instantiating our proofs under the mPKE assumption (instead of the AGM) is significantly more efficient than for those in [19]. The overhead of instantiating our proofs under mPKE is: for us, have 4 more $\mathbb{G}_1$ elements and the prover needs up to $3n + 6m$ more $\mathbb{G}_1$ exponentiations: in [19], 11 more $\mathbb{G}_1$ elements in the proof and $11n + 5m$ more exponentiations to the prover.

Acknowledgments

This work has received funding in part from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation

program under project PICOCRYPT (grant agreement No. 101001283), by the Spanish Government under projects SCUM (ref. RTI2018-102043-B-I00), CRYPTOEPIC (ref. EUR2019-103816), and SECURITAS (ref. RED2018-102321-T), by the Madrid Regional Government under project BLOQUES (ref. S2018/TCS-4339), and by research grants from Protocol Labs, and by Nomadic Labs and the Tezos Foundation. The first and second authors were at the IMDEA Software Institute while developing part of this work. Additionally, the project that gave rise to these results received the support of a fellowship from "la Caixa" Foundation (ID 100010434). The fellowship code is LCF/BQ/ES18/11670018.

References

1. Abdolmaleki, B., Baghery, K., Lipmaa, H., Zajac, M.: A Subversion-Resistant SNARK. In: ASIACRYPT 2017, Part III. pp. 3–33 (2017)
2. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In: ACM CCS 2017. pp. 2087–2104 (2017)
3. Baghery, K.: Subversion-Resistant Simulation (Knowledge) Sound NIZKs. In: 17th IMA International Conference on Cryptography and Coding. pp. 42–63 (2019)
4. Ben-Sasson, E., Bentov, I., Chiesa, A., Gabizon, A., Genkin, D., Hamilis, M., Pergament, E., Riabzev, M., Silberstein, M., Tromer, E., Virza, M.: Computational Integrity with a Public Random String from Quasi-Linear PCPs. In: EUROCRYPT 2017, Part III. pp. 551–579 (2017)
5. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable Zero Knowledge with No Trusted Setup. In: CRYPTO 2019, Part III. pp. 701–732 (2019)
6. Ben-Sasson, E., Chiesa, A., Goldberg, L., Gur, T., Riabzev, M., Spooner, N.: Linear-Size Constant-Query IOPs for Delegating Computation. In: TCC 2019, Part II. pp. 494–521 (2019)
7. Ben-Sasson, E., Chiesa, A., Green, M., Tromer, E., Virza, M.: Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs. In: 2015 IEEE Symposium on Security and Privacy. pp. 287–304 (2015)
8. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent Succinct Arguments for R1CS. In: EUROCRYPT 2019, Part I. pp. 103–128 (2019)
9. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive Oracle Proofs. In: TCC 2016-B, Part II. pp. 31–60 (2016)
10. Benarroch, D., Campanelli, M., Fiore, D.: Commit-and-Prove Zero-Knowledge Proof Systems. ZKProof.org (2020)
11. Benarroch, D., Campanelli, M., Fiore, D., Gurkan, K., Kolonelos, D.: Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular. In: Financial Cryptography and Data Security (2021)
12. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again. In: ITCS 2012. pp. 326–349 (2012)
13. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct Non-interactive Arguments via Linear Interactive Proofs. In: TCC 2013. pp. 315–333
14. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive, Report 2020/081 (2020)

15. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting. In: EUROCRYPT 2016, Part II. pp. 327–357 (2016)
16. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short Proofs for Confidential Transactions and More. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334 (2018)
17. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK Compilers. In: EUROCRYPT 2020, Part I. pp. 677–706 (2020)
18. Campanelli, M., Fiore, D., Querol, A.: LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs. In: ACM CCS 2019. pp. 2075–2092 (2019)
19. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.P.: Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In: EUROCRYPT 2020, Part I. pp. 738–768 (2020)
20. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and Transparent Recursive Proofs from Holography. In: EUROCRYPT 2020, Part I. pp. 769–793 (2020)
21. Daza, V., Ràfols, C., Zacharakis, A.: Updateable Inner Product Argument with Logarithmic Verifier and Applications. In: PKC 2020, Part I. pp. 527–557 (2020)
22. Escala, A., Groth, J.: Fine-Tuning Groth-Sahai Proofs. In: PKC 2014. pp. 630–649
23. Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.: An Algebraic Framework for Diffie-Hellman Assumptions. In: CRYPTO 2013, Part II. pp. 129–147 (2013)
24. Fuchsbauer, G.: Subversion-Zero-Knowledge SNARKs. In: PKC 2018, Part I. pp. 315–347 (2018)
25. Fuchsbauer, G., Kiltz, E., Loss, J.: The Algebraic Group Model and its Applications. In: CRYPTO 2018, Part II. pp. 33–62 (2018)
26. Gabizon, A.: AuroraLight: Improved prover efficiency and SRS size in a Sonic-like system. Cryptology ePrint Archive, Report 2019/601 (2019)
27. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Report 2019/953 (2019)
28. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic Span Programs and Succinct NIZKs without PCPs. In: EUROCRYPT 2013. pp. 626–645 (2013)
29. Gentry, C., Wichs, D.: Separating Succinct Non-Interactive Arguments From All Falsifiable Assumptions. In: 43rd ACM STOC. pp. 99–108 (2011)
30. Goldreich, O., Oren, Y.: Definitions and Properties of Zero-Knowledge Proof Systems. Journal of Cryptology (1), 1–32 (1994)
31. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing (1), 186–208 (1989)
32. Groth, J.: Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In: ASIACRYPT 2010. pp. 321–340 (2010)
33. Groth, J.: On the Size of Pairing-Based Non-interactive Arguments. In: EUROCRYPT 2016, Part II. pp. 305–326 (2016)
34. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and Universal Common Reference Strings with Applications to zk-SNARKs. In: CRYPTO 2018, Part III. pp. 698–728 (2018)
35. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N.: Zcash Protocol Specification. Tech. rep. 2016–1.10. Zerocoin Electric Coin Company, Tech. Rep. (2016)
36. Ishai, Y.: Efficient Zero-Knowledge Proofs: A Modular Approach. Simons Institute. Lecture (2019)
37. Ishai, Y.: Zero-Knowledge Proofs from Information-Theoretic Proof Systems - Part I. ZKProof.org, Blog entry (2020)

38. Ivanov, K.G., Saff, E.B.: Behavior of the Lagrange Interpolants in the Roots of Unity, pp. 81–87. Springer Berlin Heidelberg, Berlin, Heidelberg (1990)
39. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-Size Commitments to Polynomials and Their Applications. In: ASIACRYPT 2010. pp. 177–194 (2010)
40. Kilian, J.: A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract). In: 24th ACM STOC. pp. 723–732 (1992)
41. Kosba, A.E., Papadopoulos, D., Papamanthou, C., Song, D.: MIRAGE: Succinct Arguments for Randomized Algorithms with Applications to Universal zk-SNARKs. In: USENIX Security 2020. pp. 2129–2146 (2020)
42. Kothapalli, A., Masserova, E., Parno, B.: Poppins: A Direct Construction for Asymptotically Optimal zkSNARKs. Cryptology ePrint Archive, Report 2020/1318 (2020)
43. Lee, J., Setty, S., Thaler, J., Wahby, R.: Linear-time zero-knowledge SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/030 (2021)
44. Lipmaa, H.: Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments. In: TCC 2012. pp. 169–189 (2012)
45. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings. In: ACM CCS 2019. pp. 2111–2128 (2019)
46. Micali, S.: Computationally Sound Proofs. SIAM Journal on Computing **30**(4), 1253–1298 (2000)
47. Ràfols, C., Zapico, A.: An Algebraic Framework for Universal and Updatable SNARKs. In: CRYPTO 2021, Part I. pp. 774–804 (2021)
48. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-Round Interactive Proofs for Delegating Computation. In: 48th ACM STOC. pp. 49–62 (2016)
49. Setty, S.: Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In: CRYPTO 2020, Part III. pp. 704–737 (2020)
50. Trefethen, L., Berrut, J.P.: Barycentric Lagrange Interpolation. SIAM Review **46(3)**, 501–517 (2004)
51. Wahby, R.S., Tzialla, I., shelat, a., Thaler, J., Walfish, M.: Doubly-Efficient zk-SNARKs Without Trusted Setup. In: 2018 IEEE Symposium on Security and Privacy. pp. 926–943 (2018)
52. Wu, H., Zheng, W., Chiesa, A., Popa, R.A., Stoica, I.: DIZK: A Distributed Zero Knowledge Proof System. In: USENIX Security 2018. pp. 675–692 (2018)
53. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In: CRYPTO 2019, Part III. pp. 733–764 (2019)
54. Yamashita, K., Tibouchi, M., Abe, M.: On The Impossibility of NIZKs for Disjunctive Languages from Commit-and-Prove NIZKs. IEEE Access (2021)
55. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In: 2020 IEEE Symposium on Security and Privacy. pp. 859–876 (2020)
56. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases. In: 2017 IEEE Symposium on Security and Privacy. pp. 863–880 (2017)
57. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: A Zero-Knowledge Version of vSQL. Cryptology ePrint Archive, Report 2017/1146 (2017)
58. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vRAM: Faster Verifiable RAM with Program-Independent Preprocessing. In: 2018 IEEE Symposium on Security and Privacy. pp. 908–925 (2018)