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

Matteo Campanelli[1] matteo.campanelli@imdea.org
Antonio Faonio[2] antonio.faonio@eurecom.fr
Dario Fiore[1] dario.fiore@imdea.org
Anaïs Querol[1,3] anais.querol@imdea.org
Hadrián Rodríguez[1] hadrian.rodriguez@imdea.org

[1] IMDEA Software Institute, Madrid, Spain
[2] EURECOM, Sophia Antipolis, France
[3] Universidad Politécnica de Madrid, Spain

Abstract

We address the problem of constructing zkSNARKs whose SRS is *universal*—valid for all relations within a size-bound—and *updatable*—a dynamic set of participants can add secret randomness to it indefinitely thus increasing confidence in the setup. We investigate formal frameworks and techniques to design efficient universal updatable zkSNARKs with linear-size SRS and their commit-and-prove variants.

We achieve a collection of zkSNARKs with different tradeoffs. One of our constructions 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, called R1CS-lite, introduced by this work. Another of our constructions supports directly standard R1CS and improves on previous work achieving the fastest proving time for this type of constraint systems.

We achieve this result via the combination of 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 zkSNARKs* for committed polynomials, (3) pairing-based realizations of these CP-SNARKs for polynomials, (4) constructions of PHPs for R1CS and R1CS-lite, (5) a variant of the compiler that yields a commit-and-prove universal zkSNARK.

Contents

1 Introduction

A zero-knowledge proof system [GMR89] 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 *succinct* zero-knowledge proof systems [Kil92, Mic00, GW11, BCCT12], also called zkSNARKs: zero-knowledge succinct non-interactive arguments of knowledge. These proofs are *succinct* in that they are short and efficient to verify: proof size and verification time should be constant or polylogarithmic in the length of the non-deterministic witness.

It may be surprising that we can achieve real succinctness at all. In fact the verifier's running time must at least read the statement to be proven which includes both the description of the computation and its input (i.e., public input). In some models of computation this seems to rule out succinctness. For example, if a verifier needs to read a whole circuit, its running time is the same as that of the actual computation. To work around this problem, some works adopted the model of *preprocessing zkSNARKs* [Gro10, Lip12, GGPR13, BCI+13], in which the verifier generates a *structured reference string* (SRS) that depends on a certain circuit C *once and for all*; this SRS can be used later to verify an unbounded number of proofs for the computation of C. This yields succinctness because while the cost of SRS generation does depend on the size of C, proof verification does not have to.

A succinct proof system needs a secure SRS, but generating one is hard. In order for a non-interactive zero-knowledge proof system (or a zkSNARK) to be fully secure (both zero-knowledge and sound) it is crucial that the SRS is generated by a trusted party. But often—e.g, cryptocurrencies and blockchains in general—there is no such trusted party and we need to simulate one, for example using MPC protocols [BCG+15]. As long as at least one of the participants is honest the SRS is secure. It is important that the MPC procedure is scalable—the more the parties involved the higher the confidence in the SRS. But in spite of improvements [BGG19, BGM17], protocols for SRS generation are still expensive: all users must be carefully coordinated and each of them requires at least one round of communication; executing a protocol involving a hundred users can take up to a few months. Finally, as mentioned, SRS generation needs to be carried once for each single[1] computation C, which makes the problem even harder.

To address this problem, Groth et al. [GKM+18] 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, and is *updatable* in the sense that a user can update an existing SRS into a new one with the property that an SRS is secure if it was obtained through a sequence of updates in which at least one update was performed by a honest user. Groth et al. [GKM+18] proposed the first zkSNARK with a universal and updatable SRS. Their scheme though required an SRS of size *quadratic* in the number of multiplication gates of the supported arithmetic circuits (and similar quadratic update/verification time).

Recent works [MBKM19, CFQ19, XZZ+19, GWC19, CHM+20, DRZ20] have improved this result and proposed zkSNARKs where the universal and updatable SRS has size only *linear* in the largest circuit to be supported. In particular, the current Marlin and PLONK proof systems achieve a proving time concretely faster than that of Sonic while retaining constant-size proofs ([CFQ19, XZZ+19, DRZ20] have instead (poly-)logarithmic-size proofs). We also mention the very recent works of Bunz, Fisch and Szepieniec [BFS20], and Chiesa, Ojha and Spooner [COS20] 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. We highlight another construction of a universal zkSNARK [KPPS20] that, despite its proofs of 4 group elements and comparable proving time, has an SRS which is not updatable.

The current landscape of zkSNARKs with universal SRS. A known modular paradigm to build efficient cryptographic arguments [Ish19] works in two distinct steps. First construct an information-theoretic protocol in some abstract model, e.g., interactive proofs [GMR89], standard or linear PCPs [BCI+13], IOPs [RRR16, BCS16]. Then apply a compiler that, taking an abstract protocol as input, transforms it into an efficient computationally sound argument via a cryptographic primitive. This

[1]One could generate an SRS for a universal circuit for computations up to size T, but this adds a multiplicative overhead of $O(\log T)$ which is often unacceptable.

approach has been successfully adopted to construct zkSNARKs with universal SRS in the recent works [GWC19, CHM+20, BFS20], in which the information theoretic object is an algebraically-flavored variant of Interactive Oracle Proofs (IOPs), while the cryptographic primitive are *polynomial commitments* [KZG10]. Through polynomial commitments, a prover can compress a polynomial p (as a message much shorter than the 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 the IOP abstraction, called *algebraic holographic proofs* (AHP) in [CHM+20] and *polynomial IOPs*[2] in [BFS20], the prover and the verifier interact with the latter sending random challenges (in the public coin version) and the prover providing oracle access to a set of polynomials. At the end of the protocol the verifier asks for evaluations of these polynomials and decides to accept or reject based on the results it receives. The *idealized low-degree protocols* (ILDPs) abstraction of [GWC19] 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). The basic idea to build a zkSNARK with universal SRS starting from AHPs/ILDPs is that the prover commits to the polynomials obtained from the information-theoretic prover, and then uses the opening feature of polynomial commitments to reply 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 [MBKM19, GWC19, CHM+20] in terms of proof size and running time of the prover. Precisely, through our results we obtain a collection of zkSNARKs with different tradeoffs (see Table 4 for the full list).

In Table 1, we present a detailed efficiency comparison between prior work[3] and the best of our schemes, dubbed LunarLite, when using arithmetic circuit satisfiability as common benchmark. As one can see, LunarLite has the smallest proof size 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 R1CS.

In Table 2, we instead show a selection of our solutions that directly support the R1CS constraint system and we give a comparison with Marlin [CHM+20]. Our scheme Lunar1cs (fast & short) ($\Pi^{(2)}_{\mathsf{r1cs2}}$ in Table 4) offers the smallest SRS, the smallest proof and the fastest proving time. This comes at the price of higher constants for the size of the (specialized) verification key and the verification time. Lunar1cs (short vk) ($\Pi^{(1)}_{\mathsf{r1cs3}}$ in Table 4) offers a tradeoff: it has smaller verification key and faster proving time than Lunar1cs (fast & short), but slightly larger proofs, $3\times$ larger SRS, and $5m$ more exponentiations in $\mathbb{G}_1$ for the prover, compared to Lunar1cs (fast & short). Even with this tradeoff, Lunar1cs (short vk) outperforms Marlin in all these measures.

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 that we call *Polynomial Holographic IOPs* (PHPs), and (2) a new compiler that combines our PHPs with *commit-and-prove zkSNARKs* for committed polynomials. Additional contributions include: (3) realizations of these CP-SNARKs for polynomials based on pairings, (4) constructions of PHPs for R1CS and a simplified variant of it that we propose, (5) a variant of the compiler that yields a commit-and-prove universal zkSNARK.

Below we explain our contributions in more detail.

Polynomial Holographic IOPs (PHPs). Our PHPs generalize AHPs[4] as well as ILDPs. Much like an ILDP of [GWC19], 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

[2]AHPs and polynomial IOPs are virtually the same notion; in the rest of our paper we use AHP and polynomial IOPs interchangeably when the minor differences between the two models are not important.

[3]We focus our comparison on solutions with constant-size proofs.

[4]More precisely, Polynomial Holographic Proofs generalize AHPs where the verifier is "algebraic" (see Section 3.2). This encompasses all the AHP constructions in [CHM+20].

zkSNARK		size				time			
		$\|\mathsf{srs}\|$	$\|\mathsf{ek_R}\|$	$\|\mathsf{vk_R}\|$	$\|\pi\|$	KeyGen	Derive	Prove	Verify
Sonic [MBKM19]	$\mathbb{G}_1$	$4N$	$36n$	—	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 [CHM+20]	$\mathbb{G}_1$	$3M$	$3m$	12	13	$3M$	$12m$	$14n+8m$	2 pairings
	$\mathbb{G}_2$	2	—	2	—	—	—	—	
	$\mathbb{F}$	—	—	—	8	—	$O(m \log m)$	$O(m \log m)$	$O(\ell + \log m)$
PLONK (small proof) [GWC19]	$\mathbb{G}_1$	$3N^*$	$3n+3a$	8	7	$3N^*$	$8n+8a$	$11n+11a$	2 pairings
	$\mathbb{G}_2$	1	—	1	—	1	—	—	
	$\mathbb{F}$	—	—	—	7	—	$O((n+a)\log(n+a))$	$O((n+a)\log(n+a))$	$O(\ell + \log(n+a))$
PLONK (fast prover) [GWC19]	$\mathbb{G}_1$	N^*	$n+a$	8	9	N^*	$8n+8a$	$9n+9a$	2 pairings
	$\mathbb{G}_2$	1	—	1	—	1	—	—	
	$\mathbb{F}$	—	—	—	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	—	$O(m \log m)$	$O(m \log m)$	$O(\ell + \log m)$

Table 1: Efficiency comparison of universal zkSNARKs for arithmetic circuit satisfiability with constant-size proofs. n: number of multiplication gates; a: number of addition gates; $m \geq n$: number of nonzero entries of the R1CS/R1CS-lite matrices encoding the arithmetic circuit; N, A and M are the largest supported values for n, a and m respectively. In PLONK N^* is the maximum for the total number of gates in a circuit.

zkSNARK		size				time			
		$\|\mathsf{srs}\|$	$\|\mathsf{ek_R}\|$	$\|\mathsf{vk_R}\|$	$\|\pi\|$	KeyGen	Derive	Prove	Verify
Marlin [CHM+20]	$\mathbb{G}_1$	$3M$	$3m$	12	13	$3M$	$12m$	$14n+8m$	2 pairings
	$\mathbb{G}_2$	2	—	2	—	—	—	—	
	$\mathbb{F}$	—	—	—	8	—	$O(m \log m)$	$O(m \log m)$	$O(\ell + \log m)$
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	—	$O(m \log m)$	$O(m \log m)$	$O(\ell + \log m)$

Table 2: Efficiency comparison of universal zkSNARKs for R1CS with constant-size proofs. n (resp. m) is the dimension (resp. the number of nonzero entries) of the R1CS matrices; N and M are the largest supported values for n and m respectively.

to be checked on the prover's polynomials (such as $a(X)b(X) - z \cdot c(X) \stackrel{?}{=} 0$, where a, b, c are oracle polynomials and z is some scalar). To provide an intuition on the PHP notion, let us compare it with that of ILDP; they differ in the following features. First, a PHP prover sends the verifier not only oracle messages—polynomials—but also actual messages—field elements). Second, a PHP verifier can also carry out another type of test: degree bounds of polynomials, e.g. $\mathsf{deg}(a(X)) < D$ (as done in AHPs with a different syntax). Third, a PHP has a fine-grained notion of zero-knowledge. While the other differences are minor, we see the last one as a substantial contribution because it allows to achieve more efficient zkSNARKs after compilation. The basic zero-knowledge of a PHP notion ensures that the prover's messages—but not the oracles—reveal no information on the witness. This is sufficient whenever one applies variants of well-known compilation approaches, such as that by Ben-Or et al. [BGG+90] or that by Cramer and Damgaard [CD98], in which the prover first commits to the oracle polynomials and then shows in zero-knowledge that the verifier would accept. This approach would require (standard) *hiding* commitments. In this work we propose more efficient compilation strategies (which we describe next) which exploit weaker commitments. In order to support weaker commitments we propose a stronger notion of zero-knowledge that we call $(\mathsf{b}_1, \ldots, \mathsf{b}_n)$-bounded zero-knowledge. The latter guarantees that one learns no information about the witness from an interaction with a PHP prover *even if* given access up to b_i evaluations of the i-th oracle polynomial[5].

From PHPs to zkSNARKs: polynomial commitments through a different lens. We describe how to compile a (public-coin) PHP into a zkSNARK. As usual, we do this in two steps: first compile the PHP into a succinct interactive argument, and then apply Fiat-Shamir to remove interaction. For AHPs and ILDPs [GWC19, CHM+20], compilation works by letting the prover use polynomial commitments to commit to the oracles and then open the commitments to the evaluations asked by the verifier. We use a similar approach with a key difference: *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 [CFQ19, BCFK19], 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 we also need to be able to prove other properties about them, such as checking degree bounds or testing equations over committed polynomials. Since 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 further step 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 this modular abstraction that are idiosyncratic to our context. 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. For example, we show a simple scheme for quadratic equations that can even have an empty proof (see below); our efficient realizations exploit this fact.

[5]Ours is a more general notion than standard zero-knowledge; the latter corresponds to $\mathbf{0}$-bounded zero-knowledge.

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 PHP 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 [GWC19], 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" [Gro10, KZG10] and then show CP-SNARKs for various relations over that same commitment scheme. In particular, by using techniques from previous works [KZG10, GWC19, CHM+20] we show CP-SNARKs for: proof of knowledge in the algebraic group model [FKL18] (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 a single group element; this relies on the PKE assumption [Gro10] 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 the pairing. This technique is reminiscent of the compiler from [BCI+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 [CHM+20]: we adopt their encoding of sparse matrices ; like in Marlin, one of our main building blocks is the sumcheck protocol from [BCR+19]. But in our PHPs we explore a different protocol for proving properties of sparse matrices and we introduce a refined efficient technique for zero-knowledge in a univariate sumcheck. In a nutshell, compared to [BCR+19] we show how to choose the masking polynomial with a specific sparse distribution, which has only a constant-time impact on the prover. The 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 obtain PHPs with different tradeoffs (see Table 3).

Commit-and-prove zkSNARKs from PHPs. We describe how to adapt our compiler in order to turn a PHP into a CP-SNARK, namely a SNARK where the verifier's input is 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. We note that this requirement rules out the committing methods for polynomials used in [GWC19, CHM+20] as these require to know a bound on the number of evaluations to be produced (and thus on the number of usages), which may be unknown at commitment time. For this reason we assume these given commitments to be full-fledged hiding rather than just somewhat-hiding.

The main idea to obtain this commit-and-prove compiler is to prove a linking between the committed witnesses $(\mathsf{u}_j)_{j\in[\ell]}$, that open *hiding* commitments $(\hat{c}_j)_{j\in[\ell]}$, and the PHP polynomials $(p_j)_{j\in[n]}$, that open *somewhat-hiding* commitments $(c_j)_{j\in[n]}$.[6] We delegate this linking task to a specific CP-SNARK, called $\mathsf{CP}_{\mathsf{link}}$, and we design one that works for pairing-based commitments to polynomials and a class of linking relations which cover our PHP constructions.

1.2 Other Related Work

In this work we focus on zkSNARKs with a universal setup and constant-size proofs. Here we briefly survey other works that obtain universality through other approaches at the cost of a larger proof size.

Spartan [Set20] obtains preprocessing arguments with a URS; it trades a transparent setup for larger arguments, $O_\lambda(n^{1/c})$, and less efficient verification, $O_\lambda(n^{1-1/c})$, for a chosen constant $c \geq 2$.

Other works obtain universal SNARGS through a transparent setup and exploiting the structure of the computation for succinctness. They mainly use two classes of techniques, hash-based vector commitments applied to oracle interactive proofs [BBC+17, BBHR19, BCG+19] or multivariate polynomial commitments and doubly-efficient[7] interactive proofs [ZGK+17b, ZGK+17a, ZGK+18, WTs+18, XZZ+19, ZXZS20].

Fractal [COS20] 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 obtains a more restricted notion of succinctness where, not preprocessing the computation, proof size is sublinear, but verification time is not. Works in this line include [Gab19, BBB+18, BCC+16, AHIV17, BCR+19]

1.3 Outline

We split our preliminaries in two different sections: basic preliminaries are in Section 2, while in Section 5 we provide background on commitment schemes, zkSNARK with universal SRS and commit-and-prove zkSNARKs (CP-SNARKs). In Section 3 we introduce Polynomial Holographic IOPs (PHPs) and compare them with AHPs. Our PHP schemes for R1CS-like constraint systems are in Section 4, which also contains a description of the R1CS-lite as well as algebraic preliminaries necessary to constructions. In Section 6 we describe our compiler from PHPs to universal zkSNARKs as well as the required building blocks, including polynomial commitment schemes. Section 7 describes commitment schemes for polynomials and compatible CP-SNARKs that we use to instantiate our compilers. Section 8 presents our compiler from PHPs to universal commit-and-prove zkSNARKs, as well as additional building blocks and their instantiations with pairing based commitments. In Section 9 we describe our concrete universal zkSNARKs and CP-SNARKs. We refer the reader to the appendix for additional preliminaries, details on constraint systems, proofs, as well as our PHP for properties of sparse matrices.

2 Basic Preliminaries

We denote by $\lambda \in \mathbb{N}$ the security parameter, and by $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ the set of polynomial and negligible functions respectively. A function $\varepsilon(\lambda)$ is said *negligible* – denoted $\varepsilon(\lambda) \in \mathsf{negl}(\lambda)$ – if $\varepsilon(\lambda)$ vanishes faster than the inverse of any polynomial in λ. An adversary $\mathcal{A}$ is called *efficient* if $\mathcal{A}$ is a family $\{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ of nonuniform circuits of size $\mathsf{poly}(\lambda)$.

For a positive integer $n \in \mathbb{N}$ we let $[n] := \{1, \ldots, n\}$. For a set S, $|S|$ denotes its cardinality, and $x \leftarrow_\$ S$ denotes the process of selecting x uniformly at random over S. We write vectors and matrices in boldface font, e.g., $\boldsymbol{v}, \boldsymbol{V}$. So for a set S, $\boldsymbol{v} \in S^n$ is a short-hand for the tuple $(v_1, \ldots, v_n)$. Given two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ we denote by $\boldsymbol{u} \circ \boldsymbol{v}$ their entry-wise (aka Hadamard) product.

[6]Note that such linking is implicit in any PHP prover strategy.

[7]i.e. with both an efficient prover and verifier [GKR08].

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

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 $\mathcal{D}_\mathsf{u} \times \mathcal{D}_\omega$. Finally, we sometimes use an even more fine-grained specification of $\mathcal{D}_\mathsf{u}$ assuming we can split it over ℓ arbitrary domains $(\mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell)$ for some arity ℓ.

3 Polynomial Holographic IOPs

In this section we define our notion of Polynomial Holographic IOPs (PHP). 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, in the sense that 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 $\mathbb{F}$, the prover can at each round send to the verifier field elements or oracle polynomials, and the verifier queries are algebraic checks over these prover messages. For example the verifier can directly check polynomial identities such as $p_1(X)p_2(X)p_3(X) + p_4(X) \stackrel{?}{=} 0$.

Compared to the AHP notion of [CHM+20] and the polynomial IOP of [BFS20], PHPs have the following differences: the prover can also send actual messages in addition to oracle polynomials, and the verifier queries are more expressive than polynomial evaluations. This richer syntax—as we shall see in Sections 6 and 9—gives us more flexibility when compiling into a cryptographic argument system. Our model is closer to the idealized polynomial protocols of [GWC19] in terms of verifier's checks, but it adds to it the aforementioned general prover messages and a notion of zero-knowledge.

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

Definition 3.1 (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 $\mathcal{RE}(\mathbb{F}, \mathsf{R})$ is executed on input a field $\mathbb{F} \in \mathcal{F}$ and a relation description* R*, and it returns* $\mathsf{n}(0)$ *polynomials $\{p_{0,j}\}_{j \in [\mathsf{n}(0)]}$ encoding the relation* R*.*
- **Online phase:** *The prover $\mathcal{P}(\mathbb{F}, \mathsf{R}, \mathsf{x}, \mathsf{w})$ and the verifier $\mathcal{V}^{\mathcal{RE}(\mathbb{F},\mathsf{R})}(\mathbb{F}, \mathsf{x})$ are executed for* $\mathsf{r}(|\mathsf{R}|)$ *rounds; the prover has a tuple $(\mathsf{R}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$ and the verifier has an instance x and 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 polynomials identities hold between the oracle polynomials and the prover messages. More in detail, 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* $(p_1, \dots, p_{\mathsf{n}^*})$ *and* $(\pi_1, \dots, \pi_{\mathsf{m}^*})$ *all the oracle polynomials (including the* $\mathsf{n}(0)$ *ones from the encoder) and all the messages sent by the prover. The verifier can specify a list of* $\mathsf{n_e}$ *tuples, each of the form* $(G, v_1, \dots, v_{\mathsf{n}^*})$*, where* $G \in \mathbb{F}[X, X_1, \dots, X_{\mathsf{n}^*}, Y_1, \dots, Y_{\mathsf{m}^*}]$ *and every* $v_k \in \mathbb{F}[X]$.
Then a tuple $(G, v_1, \dots, v_{\mathsf{n}^*})$ *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 below the following shorthands for two more measures of* PHP*, the degree* D *and the 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 *can satisfy completeness, (knowledge) soundness and zero-knowledge, defined as follows.*

Completeness. *A PHP is complete if for all* $\mathbb{F} \in \mathcal{F}$ *and any satisfying triple* $(\mathsf{R}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$*, the checks returned by* $\mathcal{V}^{\mathcal{RE}(\mathbb{F}, \mathsf{R})}(\mathbb{F}, \mathsf{x})$ *after interacting with the honest prover* $\mathcal{P}(\mathbb{F}, \mathsf{R}, \mathsf{x}, \mathsf{w})$*, are satisfied with probability 1.*

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 also rewind it) and obtain all the messages and polynomials returned by it.*

Zero-Knowledge. PHP *is* ϵ-zero-knowledge *if there exists a PPT simulator* $\mathcal{S}$ *such that for every field* $\mathbb{F}$*, 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* $\pi_1, \dots, \pi_{\mathsf{m}^*}$ *(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.*

In our PHP notion the use of prover's messages π_i is not strictly necessary as they could be replaced by (degree-0) polynomial oracles evaluated on 0 during the checks. However, having them explicitly is useful for the zero-knowledge definition: while messages are supposed not to leak information on the witness (i.e., they must be simulated), this does not hold for the oracles. Looking ahead to our compiler, this implies that one does not need to hide these messages from the verifier.

On the class of polynomial checks. In the definition above, the class of polynomial checks of the verifier is stated quite generally. For convenience, 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. A PHP is said *public coin* if each verifier message ρ_i, for $i = 1, \ldots, \mathsf{r}(|\mathsf{R}|)$, is a random element over a prescribed set, and so is an additional value $\rho_{\mathsf{r}(|\mathsf{R}|)+1}$ possibly used by the verifier to generate the final checks. A PHP is *non-adaptive* if all the verifier's checks can be fully determined from its inputs and randomness, and thus are independent of the prover's messages.

Since the PHP verifier's checks are also polynomials evaluated over the prover's messages, one may wonder if these are really independent. However, we note that, once having fixed the verifier's randomness (which is independent of the prover's messages), these checks (i.e., the pairs of polynomials $(G, \boldsymbol{v})$ and degrees $\boldsymbol{d}$) can be fully determined. More formally, this means that the verifier $\mathcal{V}(\mathbb{F}, \mathsf{x})$ can be written as the combination of two prover-independent algorithms: a probabilistic sampler $S_{\mathcal{V}}(\mathbb{F}) \to \boldsymbol{\rho} := (\rho_1, \ldots, \rho_{\mathsf{r}(\mathsf{R})+1})$ and a deterministic algorithm $D_{\mathcal{V}}(\mathbb{F}, \mathsf{x}; \boldsymbol{\rho}) \to (\boldsymbol{d} \cup \{(G_j, \boldsymbol{v}_j)\}_{j \in [\mathsf{n_e}]})$.

In our work, we only consider PHPs that are public coin and non-adaptive.

In the following we define two additional properties that can be satisfied by a PHP.

Bounded Zero-Knowledge. We define a zero-knowledge property for PHPs, which is useful for our compiler of Section 6. Intuitively, this property requires that zero-knowledge holds even if the view includes a bounded number of evaluations of certain oracle polynomials at given points. Since such evaluations may leak information about the witness, this property ensures that this is not the case.

For simplicity, we define this property for our scenario of interest only: for PHPs that are public-coin and with non-adaptive honest verifiers.

The notion below shall guarantee zero-knowledge against verifiers that follow the specification of the protocol (thus, they are honest) but that can also arbitrarily query the polynomials sent by the prover. However, as the polynomials evaluated in some specific points could leak bits of information of the witness we define a notion of "admissible" evaluations.

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 3.2 ((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* $\Pr[\mathsf{C}(i, y) = 0] \in \mathsf{negl}(\lambda)$ *where* y *is uniformly sampled over* $\mathbb{F}$*.*

Straight-line extractability. In our compiler to commit-and-prove zkSNARKs, we consider PHPs where the extractor for the knowledge soundness satisfies a stronger property usually called in literature as *straight-line extractability.* Informally, we consider an extractor that upon input the polynomials returned by the prover during an interaction with the verifier outputs a valid witness. We formalize this property below:

Definition 3.3 (Knowledge Soundness for PHPs with straight-line extractor.). *A PHP is* ϵ*-knowledge-sound with straight-line extractor if there exists an extractor* $\mathsf{WitExtract}$ *such that for any prover* $\mathcal{P}^*$*, every field* $\mathbb{F} \in \mathcal{F}$*, relation* R*, and instance* x*:*

$$\Pr\left[(\mathsf{R}, \mathsf{x}, \mathsf{WitExtract}((p_j)_{j \in [\mathsf{n_p}]})) \in \mathsf{R}\right] \geq \Pr[\langle \mathcal{P}^*, \mathcal{V}^{\mathcal{RE}(\mathbb{F},\mathsf{R})}(\mathbb{F}, \mathsf{x})\rangle = 1] - \epsilon$$

where $(p_j)_{j \in [\mathsf{n_p}]}$ *are the polynomials output by* $\mathcal{P}^*$ *in an execution of* $\langle \mathcal{P}^*, \mathcal{V}^{\mathcal{RE}(\mathbb{F},\mathsf{R})}(\mathbb{F}, \mathsf{x})\rangle$*.*

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 compilers in Sections 6 and 8.

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 $\mathcal{R}$, where D is its maximal degree. We define $\mathcal{R}_{\mathsf{php}}$ as a family of polynomial-time relations that expresses 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 two relations defined below:

$$\mathsf{R_{deg}}((d_k)_{k\in[\mathsf{n_p}]}, (p_k)_{k\in[\mathsf{n_p}]}) := \bigwedge_{k\in[\mathsf{n_p}]} \mathsf{deg}(p_k) \stackrel{?}{\leq} d_k$$

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

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 $(G', \boldsymbol{v})$, $\mathsf{R_{eq}}$ expresses such check if one considers G as the partial evaluation of G' at $(Y_1 = \pi_1, \ldots, Y_{\mathsf{m}^*} = \pi_{\mathsf{m}^*})$. $\mathsf{R_{deg}}$ instead expresses the degree checks of a PHP verifier.

Given two 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 operation $\mathsf{R}_A \times \mathsf{R}_B \subset \mathcal{D}_\mathsf{x} \times \mathcal{D}'_\mathsf{x} \times \mathcal{D}_\mathsf{w}$ 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 an integer $\mathsf{n_e}$, let

$$\mathsf{R}_{\mathsf{n}^*,\mathsf{n_p},\mathsf{n_e}} := \mathsf{R_{deg}} \times \overbrace{\mathsf{R_{eq}} \times \cdots \times \mathsf{R_{eq}}}^{\mathsf{n_e}\text{ times}}$$

Then we can define the family $\mathcal{R}_{\mathsf{php}}$ as

$$\mathcal{R}_{\mathsf{php}} := \left\{\mathsf{R}_{\mathsf{n}^*(|\mathsf{R}|),\mathsf{n_p}(|\mathsf{R}|),\mathsf{n_e}(|\mathsf{R}|)} : \mathsf{R} \in \mathcal{R}\right\}$$

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

3.2 Compiling PHPs and AHPs into One Another

Here we discuss ways in which the formalisms of PHPs and AHPs are similar and how they can be compiled into each other straightforwardly. Recall that the main difference in the semantics of the two models is that a PHP supports more abstract queries that may not involve actual polynomial evaluations but only polynomial equations. One more difference is in the expressivity of verifiers' decision algorithms (see below).

In the remainder of this section we consider only public-coin AHPs and PHPs with non-adaptive queries. For AHPs, this implies that the last steps of verification can be expressed as a pair of algorithms: one outputs a tuple of queries for the polynomial oracles; the other algorithm, that we denote by $\mathcal{V}_{\text{AHP}}$ decides whether to accept or reject and takes the oracle responses and the view of the verifier's randomness as input. We can structure the verifier in a public-coin PHP with non-adaptive queries in an analogous manner.

There is one main difference between the verifiers in the two models: the decision algorithm of a PHP, $\mathcal{V}_{\text{PHP}}$, is completely "algebraic"; $\mathcal{V}_{\text{AHP}}$ is an arbitrary algorithm. While $\mathcal{V}_{\text{PHP}}$ accepts if and only if all the degree-bounds and polynomial checks hold, $\mathcal{V}_{\text{AHP}}$ can (in principle) run any arbitrary subroutine internally. We remark, however, that all the AHP constructions in [CHM+20] and several of the polynomial IOPs[8] described in [BFS20] actually present a very specific structure: they can all be expressed as a set of randomized zero-tests of low-degree polynomials[9]. We finally assume that the verifier accepts if and only if all tests pass and that all polynomials in a test are sampled on the same unique point. When compiling AHPs into PHPs below, we shall assume this restriction on $\mathcal{V}_{\text{AHP}}$.

[8]Polynomial IOP Starks [BBHR19], Spartan [Set20] and Sonic Univariate [MBKM19].

[9]The final step in the constructions in Marlin [CHM+20] can be expressed as a conjunction of checks of the type $p_i(q_i, y_1, \ldots, y_{k_i}) = 0$, where q_i is a point the verifier queried, y_j-s are oracle responses for the query q_i, p_i is some low-degree polynomial.

Some high-level observations about compilation follow. When compiling AHPs into PHPs, or viceversa, the offline stages and the public coins sent by the verifier are the same. In the compilers below we need to slightly modify the provers in the two models (that we denote respectively by $\mathcal{P}_{\mathrm{AHP}}$ and $\mathcal{P}_{\mathrm{PHP}}$) as well as the last steps of the verifiers. We need to take into account that the verifier in an AHP performs point-evaluation queries, whereas a PHP verifier does not. While all communication from $\mathcal{P}_{\mathrm{PHP}}$ consists in providing oracle access to some polynomial, in a PHP the prover can also send "messages", scalars whose distribution we require to simulate for zero-knowledge.

Compiling PHP $\to$ AHP: The AHP prover $\mathcal{P}_{\mathrm{AHP}}$ sends the same oracle polynomials at the same round as $\mathcal{P}_{\mathrm{PHP}}$. It also sends all messages (scalars) from $\mathcal{P}_{\mathrm{PHP}}$ at their respective rounds as degree-0 oracle polynomials. We let $\mathcal{V}_{\mathrm{PHP}}$ sample K random scalars $(r_i)_{i\in[K]}$, where K is the number of polynomial tests of $\mathcal{V}_{\mathrm{PHP}}$. It then queries all the oracle polynomials in test i of $\mathcal{V}_{\mathrm{PHP}}$ on point r_i. Finally, for each of the polynomial checks i in the PHP it evaluates $F(r_i)$ with F as defined in "Polynomial checks" in Definition 3.1 (at this point the verifier has all it needs to perform such a computation). It accepts if and only if all the evaluations equal 0.

Compiling AHP $\to$ PHP: The PHP prover $\mathcal{P}_{\mathrm{PHP}}$ acts exactly as $\mathcal{P}_{\mathrm{AHP}}$ does by sending the same oracle polynomials at their respective rounds (it sends no scalar messages). We let $\mathcal{V}_{\mathrm{PHP}}$ perform the same test as $\mathcal{V}_{\mathrm{AHP}}$ and encode the queries of $\mathcal{V}_{\mathrm{AHP}}$ as constant polynomials v_j-s (see "Polynomial checks" in Definition 3.1) appropriately. More specifically, each of the polynomials v_j-s in test i are such that $v_j(X) = r_i$ where r_i is the polynomial we are sampling in test i. The polynomial G for each test is the one derived from the $\mathcal{V}_{\mathrm{AHP}}$ in the natural way. We also let $\mathcal{V}_{\mathrm{PHP}}$ output an explicit degree check for each of the oracle polynomials.

4 Our PHP Constructions

In this section we present a collection of PHP constructions for two types of constraint systems: the by now standard rank-1 constraint systems [GGPR13] and an equally expressive variant we introduce in Section 4.3 called R1CS-lite. The two differ in the number of matrices used to represent a relation. While any relation for R1CS uses three matrices, instances of R1CS-lite use only two; the R1CS-lite matrices have roughly the same size as the ones in R1CS.

All the PHPs in this section derive from the same (implicit) bare-bone protocols: one for R1CS and another one for R1CS-lite. We then provide variants of these protocols differing in two dimensions: how we encode non-zero entries in matrices—the ones corresponding to the relation—and how low is the degree in the verifier's checks. In $\mathsf{PHP}_{\mathsf{lite1}}$ (resp. $\mathsf{PHP}_{\mathsf{r1cs1}}$), we encode non-zero entries of the matrices using one single mapping, while in $\mathsf{PHP}_{\mathsf{lite2}}$ (resp. $\mathsf{PHP}_{\mathsf{r1cs2}}$), each matrix carries its own mapping. In turn, we describe for each of these four constructions PHP_{*} a slight variant that uses fewer polynomials to represent the relation, that we refer to as $\mathsf{PHP}_{*\mathsf{x}}$ (intuition: *"the fewer polynomials"* $\approx$ *"the higher the degree of the verifer checks"*). Finally we provide one more construction called $\mathsf{PHP}_{\mathsf{r1cs3}}$ that shows an interesting tradeoff between the complexity of the offline phase and the verifier workload.

4.1 Algebraic Preliminaries

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, that is the unique monic polynomial of degree at most $|S|$ that is zero on every point of S. Also, for any $S \subseteq \mathbb{F}$ we denote 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 $s' \in S$

$$\mathcal{L}_s^S(s') = \begin{cases} 1 & \text{if } s = s', \\ 0 & \text{otherwise.} \end{cases}$$

Multiplicative subgroups. In this paper we work with subsets of $\mathbb{F}$ that are *multiplicative subgroups*. These have nice efficiency properties crucial for our results. 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, [IS90, TB04, WZC+18] show that for such specific $\mathbb{H}$ every Lagrange polynomial has the following compact representation $\mathcal{L}^{\mathbb{H}}_\eta(X) = \frac{\eta}{|\mathbb{H}|} \cdot \frac{X^{|\mathbb{H}|}-1}{X-\eta}$. Both $\mathcal{Z}_\mathbb{H}(X)$ and $\mathcal{L}^{\mathbb{H}}_\eta(X)$ can be evaluated in $O(\log n)$ field operations. When $\mathbb{H}$ is clear from the context we just write $\mathcal{Z}(X)$ instead of $\mathcal{Z}_\mathbb{H}(X)$.

We assume that $\mathbb{H}$ comes with a bijection $\phi_\mathbb{H} : \mathbb{H} \to [n]$ (e.g., using a canonical ordering of the elements of $\mathbb{H}$). For more compact notation, we use elements of $\mathbb{H}$ to index the entries of a matrix $\boldsymbol{M} \in \mathbb{F}^{n\times n}$ (resp. vector $\boldsymbol{v} \in \mathbb{F}^n$, namely we use $\boldsymbol{M}_{\eta,\eta'}$ (resp. $\boldsymbol{v}_\eta$) to denote $\boldsymbol{M}_{\phi_\mathbb{H}(\eta),\phi_\mathbb{H}(\eta')}$ (resp. $\boldsymbol{v}_{\phi_\mathbb{H}(\eta)}$).

For a multiplicative subgroup $\mathbb{H} \subseteq \mathbb{F}$ of order n and any vector $\boldsymbol{v} \in \mathbb{F}^n$, we denote by $v(X)$ its *interpolating polynomial* in $\mathbb{H}$, which is the unique polynomial of degree at most $|\mathbb{H}| - 1$ such that, for all $\eta \in \mathbb{H}$, $v(\eta) = \boldsymbol{v}_\eta$. Note that $v(X)$ can be computed from $\boldsymbol{v}$ in time $O(n \log n)$.

Lemma 4.1 (Polynomial Division)**.** *Given a multiplicative subgroup $\mathbb{H} \subset \mathbb{F}$ and polynomial $p \in \mathbb{F}_{\le d}[X]$ where $d \ge n$, there exist unique quotient and remainder polynomials $q \in \mathbb{F}_{\le d-|\mathbb{H}|}(X), r \in \mathbb{F}_{\le |\mathbb{H}|-2}(X)$ and constant $c \in \mathbb{F}$ such that $p(X) = q(X)\cdot \mathcal{Z}_\mathbb{H}(X) + X\cdot r(X) + c$. We denote by $\mathsf{DivPoly}_\mathbb{H}$ the (efficient) procedure that computes these polynomials in $O(d \log |\mathbb{H}|)$ time using polynomial long division.*

We use the following strategy from [BCR+19, CHM+20] as the main tool to define a sumcheck protocol for univariate polyomials over multiplicative subgroups:

Lemma 4.2 (Univariate Sumcheck)**.** *Let $p \in \mathbb{F}_d[X]$ and multiplicative subgroup $\mathbb{H} \subset \mathbb{F}$ of order $|\mathbb{H}| = n$,*

$$\sigma = \sum_{\eta\in\mathbb{H}} p(\eta) \iff p(0) = \frac{\sigma}{n}$$

Proof. The above claim $\sigma = \sum_{\eta\in\mathbb{H}} p(\eta)$ is equivalent to $0 = \sum_{\eta\in\mathbb{H}} p'(\eta)$ with $p'(X) = p(X) - \frac{\sigma}{n}$. Let $q(X), r(X), c$ be the output of $\mathsf{DivPoly}_\mathbb{H}(p'(X))$ and $p'(0) = c = 0$, since $\mathcal{Z}_\mathbb{H}(\eta) = 0$ for all $\eta \in \mathbb{H}$, the sumcheck above reduces to checking $\sum_{\eta\in\mathbb{H}} \eta \cdot r(\eta) = 0$. By the zero sum lemma from the Aurora proof system [BCR+19][Remark 5.6], given any polynomial $f \in \mathbb{F}_{<n}[X]$ and multiplicative subgroup $\mathbb{H}$ of size n it holds that $\sum_{\eta\in\mathbb{H}} f(\eta) = 0$ if and only if $f(0) = 0$. Then, the equation holds because $r'(X) = X \cdot r(X)$ is a polynomial of degree less than n with constant term 0. □

Definition 4.1 (Masking Polynomial)**.** *Given a subgroup $\mathbb{H} \subset \mathbb{F}$ and an integer $\mathsf{b} \ge 1$, we denote by $\mathsf{Mask}^\mathbb{H}_\mathsf{b}(\cdot)$ a method which on input a polynomial $p \in \mathbb{F}_{<|\mathbb{H}|}[X]$ returns a random polynomial $p'(X) \in \mathbb{F}_{<|\mathbb{H}|+\mathsf{b}}[X]$ that agrees with $p(X)$ on the points of the subgroup $\mathbb{H}$. This is essentially a shorthand for $\mathsf{Mask}^\mathbb{H}_\mathsf{b}(p(X)) := p(X) + \mathcal{Z}_\mathbb{H}(X)\rho(X)$ for a randomly sampled $\rho(X) \leftarrow\!\!\$\, \mathbb{F}_{<\mathsf{b}}[X]$.*

Definition 4.2 (Bivariate Lagrange polynomial)**.** *Given a multiplicative subgroup $\mathbb{H} \subseteq \mathbb{F}$, we define the* bivariate Lagrange polynomial $\Lambda_\mathbb{H}(X, Y) := \frac{\mathcal{Z}_\mathbb{H}(X)\cdot Y - X\cdot\mathcal{Z}_\mathbb{H}(Y)}{n\cdot(X-Y)}$.

This polynomial has two properties that are interesting for our work. First, for all $\eta \in \mathbb{H}$ it holds that $\Lambda_\mathbb{H}(X, \eta) = \mathcal{L}^\mathbb{H}_\eta(X)$. Second, its compact representation enables its evaluation in $O(\log n)$ time.

The first property is a direct corollary of the following lemma.

Lemma 4.3. *Let $\mathbb{F}$ be a finite field and $\mathbb{H} \subseteq \mathbb{F}$ a multiplicative subgroup. Then it holds $\Lambda_\mathbb{H}(X, Y) = \sum_{\eta\in\mathbb{H}} \mathcal{L}^\mathbb{H}_\eta(X) \cdot \mathcal{L}^\mathbb{H}_\eta(Y)$.*

Proof. The claim is proven via the following transformations:

$$\begin{aligned}
\sum_{\eta\in\mathbb{H}} \mathcal{L}^\mathbb{H}_\eta(X)\cdot\mathcal{L}^\mathbb{H}_\eta(Y) &= \sum_{\eta\in\mathbb{H}} \frac{\eta^2}{n^2}\frac{\mathcal{Z}_\mathbb{H}(X)\cdot\mathcal{Z}_\mathbb{H}(Y)}{(X-\eta)(Y-\eta)} = \frac{\mathcal{Z}_\mathbb{H}(X)\cdot\mathcal{Z}_\mathbb{H}(Y)}{X-Y}\sum_{\eta\in\mathbb{H}}\frac{\eta^2}{n^2}\frac{X-Y}{(X-\eta)(Y-\eta)} \\
&= \frac{\mathcal{Z}_\mathbb{H}(X)\cdot\mathcal{Z}_\mathbb{H}(Y)}{n\cdot(X-Y)}\sum_{\eta\in\mathbb{H}}\frac{\eta^2}{n}\left(\frac{X-\eta}{(X-\eta)(Y-\eta)} + \frac{X-Y-(X-\eta)}{(X-\eta)(Y-\eta)}\right) \\
&= \frac{\mathcal{Z}_\mathbb{H}(X)\cdot\mathcal{Z}_\mathbb{H}(Y)}{n\cdot(X-Y)}\sum_{\eta\in\mathbb{H}}\frac{\eta^2}{n}\left(\frac{1}{Y-\eta} - \frac{1}{X-\eta}\right) \\
&= \frac{1}{n\cdot(X-Y)}\left(\mathcal{Z}_\mathbb{H}(X)\sum_{\eta\in\mathbb{H}}\eta\cdot\mathcal{L}^\mathbb{H}_\eta(Y) - \mathcal{Z}_\mathbb{H}(Y)\sum_{\eta\in\mathbb{H}}\eta\cdot\mathcal{L}^\mathbb{H}_\eta(X)\right) \\
&= \frac{(\mathcal{Z}_\mathbb{H}(X)\cdot Y - \mathcal{Z}_\mathbb{H}(Y)\cdot X)}{n\cdot(X-Y)}
\end{aligned}$$

In the last step we used the property that any polynomial $p(X)$ of degree $< |\mathbb{H}|$ can be written as $\sum_{\eta\in\mathbb{H}} p(\eta)\cdot\mathcal{L}^{\mathbb{H}}_{\eta}(X)$, which implies that $X = \sum_{\eta\in\mathbb{H}} \eta\cdot\mathcal{L}^{\mathbb{H}}_{\eta}(X)$. □

Sparse Matrix Encodings For a matrix $\boldsymbol{M}$ we denote by $||\boldsymbol{M}||$ the number of its nonzero entries, which we call its *density*. We will occasionally use encodings for sparse matrices inspired to that of [CHM+20]. In brief, a sparse matrix $\boldsymbol{M}$ can be represented with three polynomials $(\mathsf{val_M}, \mathsf{row_M}, \mathsf{col_M})$, where $\mathsf{row_M} : \mathbb{K} \to \mathbb{H}$ (resp. $\mathsf{col_M} : \mathbb{K} \to \mathbb{H}$) is the function such that $\mathsf{row_M}(\kappa)$ (resp. $\mathsf{col_M}(\kappa)$) 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.

Definition 4.3 (Sparse Matrix Encodings)**.** *Let $\mathbb{H}$ be a multiplicative subgroup of order n, $\boldsymbol{M} \in \mathbb{F}^{n\times n}$ be a square matrix with elements in $\mathbb{F}$, and let $\mathbb{K}$ be another multiplicative subgroup of $\mathbb{F}$ whose order is at least[10] the number of nonzero elements of $\boldsymbol{M}$, namely $||\boldsymbol{M}|| \leq |\mathbb{K}|$.*

The sparse encoding *of $\boldsymbol{M}$ is a triple $(\mathsf{val_M}, \mathsf{row_M}, \mathsf{col_M})$ of polynomials in $\mathbb{F}_{<|\mathbb{K}|}[X]$ such that for all $\kappa \in \mathbb{K}$*

$$\mathsf{val_M}(\kappa) = \boldsymbol{M}_{\mathsf{row_M}(\kappa),\mathsf{col_M}(\kappa)}$$

We define the matrix-encoding polynomial *of $\boldsymbol{M}$ as the bivariate polynomial*

$$V_M(X,Y) := \sum_{\kappa\in\mathbb{K}} \mathsf{val_M}(\kappa)\cdot\mathcal{L}^{\mathbb{H}}_{\mathsf{row_M}(\kappa)}(X)\cdot\mathcal{L}^{\mathbb{H}}_{\mathsf{col_M}(\kappa)}(Y).$$

Note that the matrix-encoding polynomial of $\boldsymbol{M}$ is such that, for all $\eta,\eta' \in \mathbb{H}$, $V_M(\eta,\eta') = \boldsymbol{M}_{\eta,\eta'}$.

When the matrix is obvious from the context, we will not explicitly use the subscript $\boldsymbol{M}$ in these polynomials.

In the following lemma we show how a sparse encoding polynomial of a matrix $\boldsymbol{M}$ can be used to express linear transformations by $\boldsymbol{M}$.

Lemma 4.4 (Sparse Linear Encoding)**.** *Let $\boldsymbol{M} \in \mathbb{F}^{n\times n}$ be a matrix with a sparse encoding polynomial $V_M(X,Y)$ as per Definition 4.3. Let $\boldsymbol{v},\boldsymbol{y} \in \mathbb{F}^n$ bet 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 $y(X) = \sum_{\eta\in\mathbb{H}} v(\eta)\cdot V_M(X,\eta)$.*

Proof. This can be seen via the following equality

$$\begin{aligned}
\sum_{\eta,\eta'\in\mathbb{H}} \boldsymbol{M}_{\eta,\eta'}\cdot v(\eta')\cdot\mathcal{L}^{\mathbb{H}}_{\eta}(X) &= \sum_{\kappa\in\mathbb{K}} \mathsf{val}_M(\kappa)\cdot v(\mathsf{col}(\kappa))\cdot\mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(X) \\
&= \sum_{\kappa\in\mathbb{K}} \mathsf{val}_M(\kappa)\cdot\sum_{\eta\in\mathbb{H}} v(\eta)\cdot\mathcal{L}^{\mathbb{H}}_{\mathsf{col}(\kappa)}(\eta)\cdot\mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(X) \\
&= \sum_{\eta\in\mathbb{H}} v(\eta)\cdot V_M(X,\eta)
\end{aligned}$$

If $\boldsymbol{y} = \boldsymbol{M}\cdot\boldsymbol{v}$ then its interpolation $y(X) = \sum_{\eta\in\mathbb{H}} \boldsymbol{y}_\eta\cdot\mathcal{L}^{\mathbb{H}}_{\eta}(X)$ can be written $y(X) = \sum_{\eta,\eta'\in\mathbb{H}} \boldsymbol{M}_{\eta,\eta'}\cdot v(\eta')\cdot\mathcal{L}^{\mathbb{H}}_{\eta}(X)$, and thus the above equality shows the desired result. On other direction, if $y(X) = \sum_{\eta\in\mathbb{H}} v(\eta)\cdot V_M(X,\eta)$ then by the above equality we have that for all $\eta\in\mathbb{H}$ holds $\boldsymbol{y}_\eta = \sum_{\eta'\in\mathbb{H}} \boldsymbol{M}_{\eta,\eta'}\cdot v(\eta')$, i.e., $\boldsymbol{y} = \boldsymbol{M}\cdot\boldsymbol{v}$. □

Joint Sparse Encodings for Multiple Matrices. Finally, 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 that is common to all matrices.

Here we show the case of two matrices $\boldsymbol{L}, \boldsymbol{R}$. This can be easily extended to more matrices. Let $\mathcal{S} = \{(\eta,\eta') \in \mathbb{H}\times\mathbb{H} : \boldsymbol{L}_{\eta,\eta'} \neq 0 \vee \boldsymbol{R}_{\eta,\eta'} \neq 0\}$ be the set of indices where either $\boldsymbol{L}$ or $\boldsymbol{R}$ are nonzero. Let $\mathbb{K}$ be the minimal-size multiplicative subgroup of $\mathbb{F}$ such that $|\mathbb{K}| \geq |\mathcal{S}|$, where $|\mathcal{S}|$ is in the worst case

[10] In the best case, we will have $|\mathbb{K}| = ||\boldsymbol{M}||$. But sometimes a subgroup of this size (being FFT-friendly as well) may not exist and we need to pad with dummy zero entries.

$||\boldsymbol{L}||+||\boldsymbol{R}||$. Then we can encode matrices $\boldsymbol{L}, \boldsymbol{R}$ similarly to definition 4.3 by using the same polynomials $\{\mathsf{row}, \mathsf{col}\}$ to keep track of the indices of their nonzero entries, and the polynomials $\{\mathsf{val}_L, \mathsf{val}_R\}$ for their values. Namely, for any $\kappa \in \mathbb{K}$, the polynomials are defined such that $\mathsf{val}_L(\kappa) = \boldsymbol{L}_{\mathsf{row}(\kappa),\mathsf{col}(\kappa)}$ and $\mathsf{val}_R(\kappa) = \boldsymbol{R}_{\mathsf{row}(\kappa),\mathsf{col}(\kappa)}$.

4.2 Rank-1 Constraint Systems

We recall the definition of the rank-1 constraint systems (R1CS) language.[11]

Definition 4.4 (R1CS)**.** *Let $\mathbb{F}$ be a finite field and $n, m, \ell \in \mathbb{N}$ be positive integers. The universal relation $\mathcal{R}_{R1CS}$ is the set of triples*

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

where $\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O} \in \mathbb{F}^{n\times n}$, $\max\{||\boldsymbol{L}||, ||\boldsymbol{R}||, ||\boldsymbol{O}||\} \le m$, $\boldsymbol{x} \in \mathbb{F}^{\ell-1}$, $\boldsymbol{w} \in \mathbb{F}^{n-\ell}$, and for $\boldsymbol{z} := (1, \boldsymbol{x}, \boldsymbol{w})$ it holds

$$(\boldsymbol{L} \cdot \boldsymbol{z}) \circ (\boldsymbol{R} \cdot \boldsymbol{z}) = \boldsymbol{O} \cdot \boldsymbol{z}$$

We now introduce a new language called *R1CS-lite*, which 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 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 as it can be used to express the language of arithmetic circuit satisfiability with essentially the same complexity as R1CS (see Appendix A.3). At the same time, though, the two-matrix form allows us to obtain PHP constructions (and resulting zkSNARKs) that are simpler and more efficient.

More formally, R1CS-lite is defined as follows.

Definition 4.5 (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 $\boldsymbol{L}, \boldsymbol{R} \in \mathbb{F}^{n\times n}$, $\max\{||\boldsymbol{L}||, ||\boldsymbol{R}||\} \le m$, the first ℓ rows of $\boldsymbol{R}$ are $(-1, 0, \ldots, 0) \in \mathbb{F}^{1\times n}$, $\boldsymbol{x} \in \mathbb{F}^{\ell-1}$, $\boldsymbol{c} \in \mathbb{F}^{n-\ell}$, and for $\boldsymbol{c} := (1, \boldsymbol{x}, \boldsymbol{w})$, it holds

$$(\boldsymbol{L}\boldsymbol{c}) \circ (\boldsymbol{R}\boldsymbol{c}) = \boldsymbol{c}$$

Summary of our PHP constructions. In the following table, we provide a summary of our constructions for R1CS and R1CS-lite that are described in the next sections:

4.3 Our PHPs for R1CS-lite

In this section we describe a collection of PHPs for the R1CS-lite constraint system. Precisely, we give one main protocol and a few variants of it that offer various efficiency tradeoffs.

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

Definition 4.6 (Polynomial R1CS-lite)**.** *Let $\mathbb{F}$ be a finite field and $n, m \in \mathbb{N}$ be positive integers. We define the universal relation $\mathcal{R}_{polyR1CS\text{-}lite}$ as the set of triples*

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

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

$$a(X) + Z \cdot b(X) + \sum_{\eta,\eta'\in\mathbb{H}} \big(\boldsymbol{L}_{\eta,\eta'} + Z \cdot \boldsymbol{R}_{\eta,\eta'}\big) \cdot a(\eta') \cdot b(\eta') \cdot \mathcal{L}^{\mathbb{H}}_{\eta}(X) \quad = \quad 0 \in \mathbb{F}[X, Z] \tag{1}$$

where $\mathbb{L} := \{\phi_{\mathbb{H}}^{-1}(1), \ldots, \phi_{\mathbb{H}}^{-1}(\ell)\}$.

The following lemma shows that the two relations are equivalent. For completeness, we give the proof in Appendix A.1.

Lemma 4.5. $\mathcal{L}(\mathcal{R}_{R1CS-lite}) \equiv \mathcal{L}(\mathcal{R}_{polyR1CS\text{-}lite})$.

[11] For simplicity of presentation, our definition uses square matrices.

PHP		degree	oracles		msgs	proof length	$\mathcal{V}$ checks		
			$\mathcal{RE}$	$\mathcal{P}$			deg	$\mathsf{deg}_{X,\{X_i\}}(G_1)$	$\mathsf{deg}_{X,\{X_i\}}(G_2)$
$\mathsf{PHP}_{\mathsf{lite1}}$	4.3.1	$2m$	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
$\mathsf{PHP}_{\mathsf{lite2}}$	4.3.2	m	24	7	1	$\lvert\pi\rvert$	2	2	2
$\mathsf{PHP}_{\mathsf{lite2x}}$	Rk.3	m	16	7	1	$\lvert\pi\rvert$	2	2	3
$\mathsf{PHP}_{\mathsf{r1cs1}}$	4.4.1	$3m$	9	8	1	$\lvert\pi'\rvert + 4m$	2	2	2
$\mathsf{PHP}_{\mathsf{r1cs1x}}$	Rk.5	$3m$	6	8	1	$\lvert\pi'\rvert + 4m$	2	2	3
$\mathsf{PHP}_{\mathsf{r1cs2}}$	4.4.2	m	57	8	1	$\lvert\pi'\rvert$	2	2	2
$\mathsf{PHP}_{\mathsf{r1cs2x}}$	Rk.6	m	42	8	1	$\lvert\pi'\rvert$	2	2	3
$\mathsf{PHP}_{\mathsf{r1cs3}}$	4.4.3	$3m$	12	8	1	$\lvert\pi'\rvert$	2	2	5

Table 3: Comparison of our PHP constructions, all with relation encoder complexity $O(m \log m)$, prover complexity $O(m \log m + n \log n)$ and verifier complexity $O(\ell + \log m + \log n)$. Here, n is the dimension of the square matrices. For simplicity of the table, we make the assumption that $|\mathbb{K}| = m > 2n$, which is true in many cases. 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 the 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 where in the first one we have all $v_j(X) = y$ and in the second one all $v_j(X) = X$.

4.3.1 Our Main PHP for R1CS-lite

We start by describing the main ideas of this PHP protocol, which we denote $\mathsf{PHP}_{\mathsf{lite1}}$. 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, as discussed in section 4.1. This encoding 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 4.4 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

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

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

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

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

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

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 [BCR+19], 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

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

that are computed as follows. First, it finds the polynomials $\{\mathsf{col}, \mathsf{row}, \mathsf{val}_L, \mathsf{val}_R\}$ described above such that for all $\kappa \in \mathbb{K}$ $\mathsf{val}_L(\kappa) = \boldsymbol{L}_{\mathsf{row}(\kappa),\mathsf{col}(\kappa)}$ and $\mathsf{val}_R(\kappa) = \boldsymbol{R}_{\mathsf{row}(\kappa),\mathsf{col}(\kappa)}$. Second, it computes:

$$\begin{aligned}
\mathsf{cr}(X) &:= \sum_{\kappa \in \mathbb{K}} \mathsf{col}(\kappa) \cdot \mathsf{row}(\kappa) \cdot \mathcal{L}^{\mathbb{K}}_{\kappa}(X) \\
\mathsf{vcr}_L(X) &:= \sum_{\kappa \in \mathbb{K}} \mathsf{val}_L(\kappa) \cdot \mathsf{cr}(\kappa) \cdot \mathcal{L}^{\mathbb{K}}_{\kappa}(X) \\
\mathsf{vcr}_R(X) &:= \sum_{\kappa \in \mathbb{K}} \mathsf{val}_R(\kappa) \cdot \mathsf{cr}(\kappa) \cdot \mathcal{L}^{\mathbb{K}}_{\kappa}(X)
\end{aligned}$$

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

Essentially, the polynomials $\mathsf{cr}(X), \mathsf{vcr}_L(X)$ and $\mathsf{vcr}_R(X)$ are low-degree extensions of the polynomials $\mathsf{col}(X) \cdot \mathsf{row}(X)$, $\mathsf{val}_L(X) \cdot \mathsf{col}(X) \cdot \mathsf{row}(X)$ and $\mathsf{val}_R(X) \cdot \mathsf{col}(X) \cdot \mathsf{row}(X)$ respectively, while $\mathsf{col}', \mathsf{row}'$ and cr' are a shifted version of $\mathsf{col}, \mathsf{row}$ and cr respectively. The intuition behind expanding the sparse encoding of $\boldsymbol{L}$ and $\boldsymbol{R}$ in this way is to keep the polynomial checks G 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 allows interesting instantiations of our compiler. As an example, by adding the polynomial $\mathsf{cr}(X)$ 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 $\langle \mathcal{P}((\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}\}, \ell), \boldsymbol{x}, (a'(X), b'(X))), \mathcal{V}(\mathbb{F}, n, m, \boldsymbol{x}) \rangle$.

Round 1.

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

The prover samples two random polynomials

$$q_s(X) \leftarrow_{\$} \mathbb{F}_{\mathsf{b}_s + \mathsf{b}_q - 1}[X], \quad r_s(X) \leftarrow_{\$} \mathbb{F}_{\mathsf{b}_r + \mathsf{b}_q - 1}[X],$$

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 to $\mathcal{V}$: $s(X)$ and 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.

$\mathcal{P}$ $\xleftarrow{x, \alpha}$ $\mathcal{V}$

$\mathcal{P}$ $\xrightarrow{\{q(X), r(X)\}}$ $\mathcal{V}$

The verifier sends two random points $x, \alpha \leftarrow_{\$} \mathbb{F}$.

The prover uses x, α to "compress" the check of equation (1) over $\mathbb{F}[X, Z]$ into a sumcheck $\sum_{\eta \in \mathbb{H}} p(\eta) = 0$ 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

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

and $\Lambda_{\mathbb{H}}(x, X) \in \mathbb{F}_{n-1}[X]$ is the minimal degree polynomial such that for all $\eta \in \mathbb{H}$: $\Lambda_{\mathbb{H}}(x, \eta) = \mathcal{L}^{\mathbb{H}}_{\eta}(x)$.

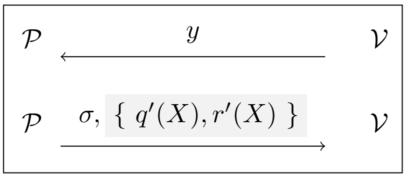
Next, $\mathcal{P}$ computes and sends polynomials $q(X) \in \mathbb{F}_{\leq 2n+\mathsf{b}_a+\mathsf{b}_b+2\mathsf{b}_q-3}[X]$ and $r(X) \in \mathbb{F}_{\leq n-2}[X]$ such that

$$s(X) + p(X) = q(X) \cdot \mathcal{Z}_{\mathbb{H}}(X) + X \cdot r(X)$$

to prove the univariate sumcheck statement $\sum_{\eta \in \mathbb{H}} s(\eta) + p(\eta) = 0$.

Note that by construction $\sum_{\eta \in \mathbb{H}} s(\eta) = 0$, and its role here is to (sufficiently) randomize $q(X), r(X)$ in such a way that their evaluations do not leak information about the witness (see the proof of bounded zero-knowledge in Theorem 4.7).

Round 3.



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

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

The goal of the prover is to convince the verifier that

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

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 the verifier since it depends only on the polynomials provided by the encoder.

For the first statement, since $p'(X)$ is a polynomial with degree smaller than the size of the subgroup $\mathbb{K}$, the univariate sumcheck lemma over $(p'(X) - \frac{\sigma}{|\mathbb{K}|})$ reduces to proving that its constant coefficient is zero. This can be done by computing $r'(X) \in \mathbb{F}_{\leq |\mathbb{K}|-2}[X]$ such that $p'(X) = X \cdot r'(X) + \frac{\sigma}{|\mathbb{K}|}$.

For the second statement, note that by decomposition of the Lagrangians this is equivalent to:

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

that, by using the definition of $p'(X)$, can be written as

$$\forall\ \kappa \in \mathbb{K} : \left(\kappa \cdot r'(\kappa) + \frac{\sigma}{|\mathbb{K}|}\right) \cdot n^2 \cdot (xy + \mathsf{cr}(\kappa) - x \cdot \mathsf{col}(\kappa) - y \cdot \mathsf{row}(\kappa)) - \big(\mathsf{vcr}_L(\kappa) + \alpha \cdot \mathsf{vcr}_R(\kappa)\big) \cdot \mathcal{Z}_{\mathbb{H}}(x) \cdot \mathcal{Z}_{\mathbb{H}}(y) = 0$$

Using the relation polynomials, $\mathcal{P}$ can define the auxiliary polynomial

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

of degree $\leq 2|\mathbb{K}| - 2$, that equals 0 on any $\kappa \in \mathbb{K}$. By the remainder theorem,

$$\forall\ \kappa \in \mathbb{K} : t(X) \equiv t(\kappa) \mod (X - \kappa) \iff t(X) \equiv 0 \mod \mathcal{Z}_{\mathbb{K}}(X)$$

Thus $\mathcal{P}$ can compute the following polynomial:

$$q'(X) := \frac{t(X)}{\mathcal{Z}_{\mathbb{K}}(X)} \in \mathbb{F}_{\leq |\mathbb{K}|-2}[X]$$

and sends $\{q'(X), r'(X)\}$ to $\mathcal{V}$.

Decision phase. The verifier outputs the following degree checks

$$\mathsf{deg}(\hat{a}'), \mathsf{deg}(\hat{b}'), \mathsf{deg}(s), \mathsf{deg}(q), \mathsf{deg}(q') \overset{?}{\leq} \mathsf{D}_{snd} \quad (3)$$

$$\mathsf{deg}(r) \overset{?}{\leq} n-2 \quad (4)$$

$$\mathsf{deg}(r') \overset{?}{\leq} |\mathbb{K}|-2 \quad (5)$$

and the following polynomial checks

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

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

where, we recall, $\Lambda_{\mathbb{H}}(x,y) = \frac{\mathcal{Z}_{\mathbb{H}}(x) \cdot y - x \cdot \mathcal{Z}_{\mathbb{H}}(y)}{n \cdot (x-y)}$. 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 ones described in Definition 3.1.

In the first degree check, D_{snd} is an integer that can be chosen by the verifier and governs the soundness error as shown in Theorem 4.6. While for correctness we need $\mathsf{D}_{snd} \geq \mathsf{D}-1$, where D is the degree of the PHP (shown below), this bound does not need to be tight (i.e., $\mathsf{D}_{snd} = \mathsf{D}-1$) as it is the case for the degree checks on r and r'. This observation has an impact in our compiler where, by choosing D_{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 "$\leq \mathsf{D}_{snd}$".

4.3.1.1 Efficiency Analysis

We analyze the efficiency of the protocol $\mathsf{PHP}_{\mathsf{lite1}}$.

Relation encoder It creates 8 polynomials, five of degree $\leq |\mathbb{K}|-1$ and three of degree $\leq |\mathbb{K}|$; this is doable in time $O(|\mathbb{K}| \log |\mathbb{K}|)$.

Degree By looking at the polynomials of the highest degree sent by relation encoder and prover, one can see that $\mathsf{D} = \max\{2n + \mathsf{b}_a + \mathsf{b}_b + 2\mathsf{b}_q - 3,\ n + \mathsf{b}_s + \mathsf{b}_q - 1,\ |\mathbb{K}|\}$, whose result depends on the difference between $|\mathbb{H}|$ and $|\mathbb{K}|$ and the concrete values of $\mathsf{b}_a, \mathsf{b}_b, \mathsf{b}_q, \mathsf{b}_s$. For example, when all these bounds are small constants (as in our use cases) and $|\mathbb{K}| \geq 3|\mathbb{H}|$, then $\mathsf{D} = |\mathbb{K}|$.

Proof length. The prover sends one element of $\mathbb{F}$ and 7 oracle polynomials. By inspection, the proof length is $\mathsf{l}(|\mathsf{R}|) = 6n + 2|\mathbb{K}| - 2\ell + 2\mathsf{b}_a + 2\mathsf{b}_b + \mathsf{b}_s + 5\mathsf{b}_q - 4$. With a closer look at the shape of $s(X)$, we have that the number of its nonzero coefficients is actually at most $\mathsf{b}_s + 2\mathsf{b}_q + \max\{\mathsf{b}_s, \mathsf{b}_r\}$, which gives us a proof length $\mathsf{l}(|\mathsf{R}|) = 5n + 2|\mathbb{K}| - 2\ell + 2\mathsf{b}_a + 2\mathsf{b}_b + 2\mathsf{b}_s + 6\mathsf{b}_q - 4$.

Prover complexity. The total complexity is $O(|\mathbb{K}| \log |\mathbb{K}| + |\mathbb{H}| \log |\mathbb{H}|)$, which is justified as follows.

The polynomials sent in the first round can be computed in time $O(|\mathbb{H}| \log |\mathbb{H}|)$.

In the second round, the less trivial step is computing $V_{LR}(x, X, \alpha)$ which we claim doable in time $O(|\mathbb{K}| + |\mathbb{H}| \log |\mathbb{H}|)$ as follows. First, one can precompute all $\mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(x)$ in time $O(|\mathbb{H}| \log |\mathbb{H}|)$ since each of them can be computed in $O(\log |\mathbb{H}|)$ time and there are at most $|\mathbb{H}|$ of these terms (recall that row maps into $\mathbb{H}$). Second, one can compute all the terms

$$\{V_{LR}(x, \eta, \alpha)\}_{\eta \in \mathbb{H}} = \left\{ \sum_{\substack{\kappa \in \mathbb{K} \\ \mathsf{col}(\kappa) = \eta}} (\mathsf{val}_L(\kappa) + \alpha \cdot \mathsf{val}_R(\kappa)) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(x) \right\}_{\eta \in \mathbb{H}}$$

$$\mathcal{P}\left((\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}\}, \ell), \boldsymbol{x'}, (a'(X), b'(X))\right) \qquad \mathcal{V}^{\,\mathsf{vcr}_L, \mathsf{vcr}_R, \mathsf{row}, \mathsf{col}, \mathsf{cr}, \mathsf{row}', \mathsf{col}', \mathsf{cr}'}(\mathbb{F}, n, m, \boldsymbol{x'})$$

Sample random q_s, r_s, set $s(X) \leftarrow q_s(X)\mathcal{Z}_{\mathbb{H}}(X) + X r_s(X)$

Sample random $\hat{a}', \hat{b}'$ that agree with a', b' on $\mathbb{H} \setminus \mathbb{L}$ $\quad \xrightarrow{\{\hat{a}', \hat{b}', s\}}$

$$\hat{a}(X) \leftarrow \hat{a}'(X)\mathcal{Z}_{\mathbb{L}}(X) + \sum_{\eta \in \mathbb{L}} \boldsymbol{x'}_{\phi_{\mathbb{H}}(\eta)} \mathcal{L}^{\mathbb{L}}_{\eta}(X)$$

$\hat{b}(X) \leftarrow \hat{b}'(X)\mathcal{Z}_{\mathbb{L}}(X) + 1$ $\quad \xleftarrow{x, \alpha}$ $\quad x, \alpha \leftarrow\!\$\, \mathbb{F}$

. . . . // Sumcheck for "$\sum_{\eta \in \mathbb{H}} s(\eta) + (\hat{a}(\eta) + \alpha \cdot \hat{b}(\eta)) \cdot \Lambda_{\mathbb{H}}(x, \eta) + \hat{a}(\eta) \cdot \hat{b}(\eta) \cdot V_{LR}(x, \eta, \alpha) = 0$" . . . .

Compute $q(X), r(X)$ s.t.

$$s(X) + (\hat{a}(X) + \alpha \cdot \hat{b}(X)) \cdot \Lambda_{\mathbb{H}}(x, X)$$
$$+ \hat{a}(X) \cdot \hat{b}(X) \cdot V_{LR}(x, X, \alpha) = q(X)\mathcal{Z}_{\mathbb{H}}(X) + X r(X) \quad \xrightarrow{\{q, r\}}$$

$\xleftarrow{y}$ $\quad y \leftarrow\!\$\, \mathbb{F} \setminus \mathbb{H}$

. . . . . . // Structured sumcheck for "$\sum_{\kappa \in \mathbb{K}} (\mathsf{val}_L(\kappa) + \alpha \cdot \mathsf{val}_R(\kappa)) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(x) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{col}(\kappa)}(y) = V_{LR}(x, y, \alpha)$" . . . . . .

$\sigma \leftarrow V_{LR}(x, y, \alpha)$

Compute $q'(X), r'(X)$ s.t. $q'(X) \cdot \mathcal{Z}_{\mathbb{K}}(X) =$

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

Verifier's checks

- $\deg(\hat{a}'), \deg(\hat{b}'), \deg(s), \deg(q), \deg(q') \stackrel{?}{\leq} \mathsf{D}_{snd} \ \wedge\ \deg(r) \stackrel{?}{\leq} n - 2 \ \wedge\ \deg(r') \stackrel{?}{\leq} |\mathbb{K}| - 2$
- $$s(y) + \left(\hat{a}'(y) \cdot \mathcal{Z}_{\mathbb{L}}(y) + \sum_{\eta \in \mathbb{L}} \boldsymbol{x'}_{\phi_{\mathbb{H}}(\eta)} \cdot \mathcal{L}^{\mathbb{L}}_{\eta}(y)\right) \cdot \left(\Lambda_{\mathbb{H}}(x, y) + (\hat{b}'(y) \cdot \mathcal{Z}_{\mathbb{L}}(y) + 1) \cdot \sigma\right)$$
$$+ (\hat{b}'(y) \cdot \mathcal{Z}_{\mathbb{L}}(y) + 1) \cdot \alpha \cdot \Lambda_{\mathbb{H}}(x, y) - q(y)\,\mathcal{Z}_{\mathbb{H}}(y) - y\, r(y) \stackrel{?}{=} 0$$

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

Figure 1: Our PHP protocol $\mathsf{PHP}_{\mathsf{lite1}}$ for R1CS-lite.

in time $O(|\mathbb{K}|)$ (with $O(|\mathbb{H}|)$ memory). This is possible by computing, for every $\kappa \in \mathbb{K}$, the term $(\mathsf{val}_L(\kappa) + \alpha \cdot \mathsf{val}_R(\kappa)) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(x)$, which can be accumulated into the relevant variable $V_{LR}(x, \eta, \alpha)$ such that $\eta = \mathsf{col}(\kappa)$. Finally, $V_{LR}(x, X, \alpha)$ is computed by interpolating $\{V_{LR}(x, \eta, \alpha)\}_{\eta \in \mathbb{H}}$ in time

$O(|\mathbb{H}| \log |\mathbb{H}|)$.

Once having computed $V_{LR}(x, X, \alpha)$, the polynomials $q(X)$ and $r(X)$ can be obtained using polynomial long division in time $O(|\mathbb{H}| \log |\mathbb{H}|)$.

In round 3, one can compute $p'(X)$ in time $O(|\mathbb{K}| \log |\mathbb{K}| + |\mathbb{H}| \log |\mathbb{H}|)$ using ideas similar the ones above, while $q'(X), r'(X)$ can be computed in time $O(|\mathbb{K}| \log |\mathbb{K}|)$ using polynomial division.

Verifier complexity. This amounts to $O(\ell + \log |\mathbb{H}| + \log |\mathbb{K}|)$ field operations, which are needed to construct the polynomial checks. In particular, notice that: computing evaluations of the vanishing polynomials in $\mathbb{H}$ costs $O(\log |\mathbb{H}|)$; $\log |\mathbb{K}|$ stems from defining the integer $|\mathbb{K}|$; and ℓ is the cost needed to compute the "shifted polynomial" with the public input.

4.3.1.2 Security Analysis

Theorem 4.6 (Knowledge Soundness). *The PHP protocol* $\mathsf{PHP}_{\mathsf{lite1}}$ *described in section 4.3 is* ϵ*-sound with* $\epsilon = \frac{|\mathbb{H}|}{|\mathbb{F}|} + \frac{2\mathsf{D}_{snd}+|\mathbb{H}|}{|\mathbb{F}\setminus\mathbb{H}|}$*, and* 0*-knowledge-sound. Furthermore,* $\mathsf{PHP}_{\mathsf{lite1}}$ *is straightline extractable (Definition 3.3).*

Proof. We begin by proving the soundness of this PHP, and then show its proof of knowledge property.

Soundness. Assume that for the given polynomial R1CS-lite relation $\mathsf{R} = (\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}\}, \ell)$ and input $\boldsymbol{x}$ there exists no witness $a'(X), b'(X)$ that satisfies the equation (1) of Definition 4.6. Then by correctness of the relation encoder's polynomials, also there is no witness satisfying equation (2).

This means that for the polynomials $\hat{a}'(X), \hat{b}'(X)$ sent by the prover in the first round it must be the case that

$$f(X, Z) = \sum_{\eta \in \mathbb{H}} (\hat{a}(\eta) + Z \cdot \hat{b}(\eta)) \cdot \Lambda_{\mathbb{H}}(X, \eta) + \hat{a}(\eta) \cdot \hat{b}(\eta) \cdot V_{LR}(X, \eta, Z) \neq 0 \text{ over } \mathbb{F}[X, Z].$$

where $\hat{a}(X)$ and $\hat{b}(X)$ are appropriately reconstructed as $\hat{a}'(X) \cdot \mathcal{Z}_{\mathbb{L}}(X) + \sum_{\eta \in \mathbb{L}} \boldsymbol{x'}_{\phi_{\mathbb{H}}(\eta)} \cdot \mathcal{L}^{\mathbb{L}}_{\eta}(X)$ and $\hat{b}'(X) \cdot \mathcal{Z}_{\mathbb{L}}(X) + 1$ respectively.

Let $s(X), \hat{a}'(X), \hat{b}'(X), q(X), r(X), q'(X), r'(X)$ and σ be the polynomials and message sent by the prover $\mathcal{P}^*$, and x, α, y be the verifier's messages. Let us recall that by the order of the messages in the protocol we have: $s(X), \hat{a}'(X), \hat{b}'(X)$ are independent of x, α, and that $\sigma, q(X), r(X)$ are independent of y.

By considering the polynomial check (7) and by the correctness of the relation encoder's polynomials we deduce that the polynomial $p'(X) := \left(X \cdot r'(X) + \frac{\sigma}{|\mathbb{K}|}\right)$ is such that

$$\forall\ \kappa \in \mathbb{K} : p'(\kappa) = \big(\mathsf{vcr}_L(\kappa) + \alpha \cdot \mathsf{vcr}_R(\kappa)\big) \cdot \frac{\mathcal{Z}_{\mathbb{H}}(x) \cdot \mathcal{Z}_{\mathbb{H}}(y)}{n^2(xy + \mathsf{cr}(\kappa) - x \cdot \mathsf{col}(\kappa) - y \cdot \mathsf{row}(\kappa))}$$

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

Then, by considering the degree check (5) we have that $r'(X) \in \mathbb{F}_{\leq |\mathbb{K}|-2}[X]$, and thus $p'(X)$ is a polynomial of degree $\leq |\mathbb{K}|-1$ with constant term $\sigma/|\mathbb{K}|$. Hence by Lemma 4.2, it holds $\sigma = \sum_{\kappa \in \mathbb{K}} p'(\kappa)$.

Putting this together with the definition of $p'(\kappa)$ we obtain that $\sigma = V_{LR}(x, y, \alpha)$.

Next, since the polynomials $s(X), \hat{a}(X), \hat{b}(X), q(X), r(X), V_{LR}(x, X, \alpha)$ are independent of y, by Schwartz-Zippel we obtain that the polynomial check (7) (combined with the first degree check (3) and that $\sigma = V_{LR}(x, y, \alpha)$) implies that

$$s(X) + (\hat{a}(X) + \alpha \cdot \hat{b}(X)) \cdot \Lambda_{\mathbb{H}}(x, X) + \hat{a}(X) \cdot \hat{b}(X) \cdot V_{LR}(x, X, \alpha) = q(X) \cdot \mathcal{Z}_{\mathbb{H}}(X) + X \cdot r(X)$$

holds with probability $\geq 1 - \frac{2\mathsf{D}_{snd}+|\mathbb{H}|}{|\mathbb{F}\setminus\mathbb{H}|}$ over the choice of y.

The degree check (4) gives us that $r(X) \in \mathbb{F}_{\leq n-2}[X]$ and thus by Lemma 4.2 we have that

$$\sum_{\eta \in \mathbb{H}} s(\eta) + f(x, \alpha) = \sum_{\eta \in \mathbb{H}} s(\eta) + (\hat{a}(\eta) + \alpha \cdot \hat{b}(\eta)) \cdot \Lambda_{\mathbb{H}}(x, \eta) + \hat{a}(\eta) \cdot \hat{b}(\eta) \cdot V_{LR}(x, \eta, \alpha) = 0$$

Let $s^* = \sum_{\eta\in\mathbb{H}} s(\eta)$. Since s^* and $f(X,Z)$ are independent of x, α, by the Schwartz-Zippel lemma, we have that, over the random choice of $x, \alpha \leftarrow\!\!\$\, \mathbb{F}$, $\Pr[f(x,\alpha) + s^* = 0] \le \frac{|\mathbb{H}|}{|\mathbb{F}|}$.

Knowledge-Soundness. We define the extractor $\mathcal{E}$, which is simply the algorithm that runs the prover $\mathcal{P}^*$ for the first round, obtains $\hat{a}'(X), \hat{b}'(X)$, and then reconstructs the non randomized witness polynomials $a'(X) = \sum_{\eta\in\mathbb{H}} \hat{a}'(\eta)\mathcal{L}^{\mathbb{H}}_\eta(X)$ and $b'(X) = \sum_{\eta\in\mathbb{H}} \hat{b}'(\eta)\mathcal{L}^{\mathbb{H}}_\eta(X)$.

If the verifier accepts with probability greater than the soundness error ϵ given above, then the polynomials returned by $\mathcal{E}$ must encode a valid witness.

Finally, it is straightforward to see the straightline extractability. The algorithm $\mathsf{WitExtract}$ is the one that takes the polynomials $\hat{a}'(X), \hat{b}'(X)$, and reconstructs the R1CS-lite witness by taking the product of their evaluations on the points of $\mathbb{H} \setminus \mathbb{L}$ (see Appendix A.1). □

Theorem 4.7 (Zero-Knowledge)**.** *The PHP $\mathsf{PHP}_{\mathsf{lite1}}$ described in section 4.3.1 is perfect zero-knowledge. Furthermore, it is perfect honest-verifier zero-knowledge with query bound* $\mathsf{b} = (\mathsf{b}_a, \mathsf{b}_b, \mathsf{b}_s, \mathsf{b}_q, \mathsf{b}_r, \infty, \infty)$.

Proof. We begin by showing the perfect zero-knowledge. This turns out rather easily. In fact, in the PHP model we do not need to worry about the oracle polynomials, the prover in Section 4.3 sends only one (non-oracle) message, σ. This message, moreover, does not depend on the witness. More formally, we describe a simulator $\mathcal{S}$ that on input the relation $\mathsf{R} = (\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}\}, \ell)$ and the input $\boldsymbol{x}$, and given oracle access to the verifier $\mathcal{V}^*$, proceeds as follows. It runs $\mathcal{V}^*$ to obtain its random messages x, y, α and its checks. Next, it computes $\sigma = V_{LR}(x, y, \alpha)$, and outputs σ followed by checks obtained from $\mathcal{V}^*$. It is easy to see that $\mathsf{View}\big(\mathcal{S}^{\mathcal{V}^*}(\mathbb{F}, \mathsf{R}, \boldsymbol{x})\big)$ is identically distributed to $\mathsf{View}\big(\mathcal{P}(\mathbb{F}, \mathsf{R}, \boldsymbol{x}, a'(X), b'(X))\ , \mathcal{V}^*\big)$.

Next, we prove b-HVZK for bounds $\mathsf{b}_a, \mathsf{b}_b, \mathsf{b}_s, \mathsf{b}_q, \mathsf{b}_r$ on the polynomials $\hat{a}'(X), \hat{b}'(X), s(X), q(X), r(X)$ respectively, whereas for the polynomials $q'(X), r'(X)$ we tolerate unbounded number of evaluations (this is trivial as these polynomials depend on public information only).

Let $\mathsf{C}(i,\gamma)$ be the algorithm that on any pair (i,γ) outputs 1 if and only if $i \in \{1, \ldots, 7\}$ and $\gamma \notin \mathbb{H}$. For a $\gamma \leftarrow\!\!\$\, \mathbb{F}$, it holds $\Pr[\mathsf{C}(i,\gamma) = 0] = |\mathbb{H}|/|\mathbb{F}|$, which is negligible for the choices of $\mathbb{F}$ considered in this paper.

The simulator samples a random tape $\boldsymbol{\rho}$ for the honest verifier and runs its query sampler $(x, y, \alpha) \leftarrow Q_{\mathcal{V}}(\boldsymbol{\rho})$ and its decision algorithm $\{\boldsymbol{d}, \{(G, \boldsymbol{v})\} \leftarrow D_{\mathcal{V}}(\mathbb{F}, \boldsymbol{x}; \boldsymbol{\rho})$ to obtain its checks. Then, it simulates answers to polynomial evaluations as follows.

For every pair (i,γ) with $i \in \{6,7\}$ (i.e., for every query on q', r'), the simulator computes $t_{i,\gamma} \leftarrow p_i(\gamma)$ honestly, which is trivial as these polynomials depend only on public information.

For every pair $(i,\gamma) \in \mathcal{L}$ such that $i \in [5] \setminus \{4\}$ (i.e., every query on $\hat{a}', \hat{b}', s, r$), the simulator samples a random value $t_{i,\gamma} \leftarrow\!\!\$\, \mathbb{F}$ and stores a tuple $(i, \gamma, t_{i,\gamma})$ in a table T.

For every query $(4,\gamma)$ it simulates the answer with the value $t_{4,\gamma}$ computed as follows:

$$\begin{aligned}
t_{a,\gamma} &\leftarrow t_{1,\gamma} \cdot \mathcal{Z}_{\mathbb{L}}(\gamma) + \sum_{\eta\in\mathbb{L}} \boldsymbol{x'}_{\phi_{\mathbb{H}}(\eta)} \cdot \mathcal{L}^{\mathbb{L}}_\eta(\gamma) \\
t_{b,\gamma} &\leftarrow t_{2,\gamma} \cdot \mathcal{Z}_{\mathbb{L}}(\gamma) + 1 \\
t_{p,\gamma} &\leftarrow (t_{a,\gamma} + \alpha \cdot t_{b,\gamma}) \cdot \Lambda_{\mathbb{H}}(x,\gamma) + t_{a,\gamma} \cdot t_{b,} \ \cdot V_{LR}(x,\gamma,\alpha) \\
t_{4,\gamma} &\leftarrow \frac{t_{p,\gamma} + t_{3,\gamma} - \gamma \cdot t_{5,\gamma}}{\mathcal{Z}_{\mathbb{H}}(\gamma)}
\end{aligned}$$

While doing the computations above, for $j = 1, 2, 3, 5$, if an entry $(j, \gamma, t_{j,\gamma})$ already exists in T, then the corresponding value $t_{j,\gamma}$ is used; otherwise a random $t_{j,\gamma} \leftarrow\!\!\$\, \mathbb{F}$ is sampled and a new entry $(j, \gamma, t_{j,\gamma})$ is added to T.

$\mathcal{S}$ returns $\big(\boldsymbol{\rho}, V_{LR}(x, y, \alpha), (\boldsymbol{d}, \{(G, \boldsymbol{v})\}), \{t_{i,\gamma}\}_{(i,\gamma)\in\mathcal{L}}\big)$.

To conclude the proof, we argue that the distribution of $\mathcal{S}$'s output is identical to that of

$$\big(\mathsf{View}\big(\mathcal{P}(\mathbb{F}, \mathsf{R}, \boldsymbol{x}, a', b')\ , \mathcal{V}\big), (p_i(\gamma))_{(i,\gamma)\in\mathcal{L}}\big)\,.$$

By the $(\mathsf{b}_a + \mathsf{b}_q)$-wise (resp. $(\mathsf{b}_b + \mathsf{b}_q)$-wise) independence of the polynomial $\hat{a}'(X)$ (resp. $\hat{b}'(X)$) sampled by the honest prover (and using the fact that they are evaluated on $\mathbb{F}\setminus\mathbb{H}$), we have that the set of simulated answers $\{t_{1,\gamma}\}_{(1,\gamma)\in\mathcal{L}}$ (resp. $\{t_{2,\gamma}\}_{(2,\gamma)\in\mathcal{L}}$) are identically distributed (we recall that these sets are of size b_a and b_b respectively) to those of the real prover.

For the remaining polynomials, let us recall that for the honest prover we have

$$p(X) = (\hat{a}(X) + \alpha \cdot \hat{b}(X)) \cdot \Lambda_{\mathbb{H}}(x, X) + \hat{a}(X) \cdot \hat{b}(X) \cdot V_{LR}(x, X, \alpha)$$
$$s(X) = q_s(X) \mathcal{Z}_{\mathbb{H}}(X) + X r_s(X)$$

where $\boldsymbol{x}' = (1, \boldsymbol{x})$, $\hat{a}(X) = \hat{a}'(X) \cdot \mathcal{Z}_{\mathbb{L}}(X) + \sum_{\eta \in \mathbb{L}} \boldsymbol{x'}_{\phi_{\mathbb{H}}(\eta)} \cdot \mathcal{L}^{\mathbb{L}}_{\eta}(X)$, $\hat{b}(X) = \hat{b}'(X) \cdot \mathcal{Z}_{\mathbb{L}}(X) + 1$, and $q_s(X) \leftarrow\$ \mathbb{F}_{\mathsf{b}_s+\mathsf{b}_q}[X]$ and $r_s(X) \leftarrow\$ \mathbb{F}_{\mathsf{b}_r+\mathsf{b}_q}[X]$. Also, let us write $p(X) = q_p(X)\mathcal{Z}_{\mathbb{H}}(X) + X r_p(X)$ for the unique $q_p(X), r_p(X)$ by polynomial division.

By the uniqueness of polynomials $q(X)$ and $r(X) \in \mathbb{F}_{\leq n-2}[X]$ such that $s(X) + p(X) = q(X) \cdot \mathcal{Z}_{\mathbb{H}}(X) + X \cdot r(X)$, we have that $q(X) = q_p(X) + q_s(X)$ and $r(X) = r_p(X) + r_s(X)$.

By the $(\mathsf{b}_r + \mathsf{b}_q)$-wise independence of $r_s(X)$ (and thus of $r(X)$) we obtain that the set of simulated answers $\{t_{5,\gamma}\}_{(5,\gamma)\in\mathcal{L}}$ (whose cardinality is at most b_r) are identically distributed to those, $\{r(\gamma)\}_{(5,\gamma)\in\mathcal{L}}$, of the real prover. Furthermore, by the $(\mathsf{b}_s + \mathsf{b}_q)$-wise independence of $q_s(X)$ we obtain that the set of simulated answers $\{t_{3,\gamma}\}_{(3,\gamma)\in\mathcal{L}}$ (whose cardinality is at most b_s) are identically distributed to those, $\{s(\gamma)\}_{(5,\gamma)\in\mathcal{L}}$, of the real prover. In particular, for this we use that for $\gamma \in \mathbb{F} \setminus \mathbb{H}$, $s(X)$ is $(\mathsf{b}_s + \mathsf{b}_q)$-wise independent even conditioned on $r_s(X)$.

To argue the correct distribution of the set of simulated answers $\{t_{4,\gamma}\}_{(4,\gamma)\in\mathcal{L}}$, we observe that the honest $q(X)$ is determined by $(p(X) + s(X) - Xr(X))/\mathcal{Z}_{\mathbb{H}}(X)$, where $p(X)$ is defined as above. In particular, an evaluation of $q(\gamma)$ on $\gamma \in \mathbb{F} \setminus \mathbb{H}$ can be obtained as $(p(\gamma) + s(\gamma) - \gamma r(\gamma))/\mathcal{Z}_{\mathbb{H}}(\gamma)$, thus using evaluations of $\hat{a}'(\gamma)$, $\hat{b}'(\gamma)$, $s(\gamma)$, $r(\gamma)$, and evaluations of publicly available polynomials. This explains the simulation strategy of $t_{4,\gamma}$ by $\mathcal{S}$, and these values are identically distributed to $q(\gamma)$ as the polynomials $\hat{a}'(X)$, $\hat{b}'(X)$, $s(X)$, and $r(X)$, each allows b_q more evaluations whose outputs are uniformly distributed. □

Remark 1 (On degree optimizations)**.** *From the proof of the above theorem it turns out that increasing the degrees of polynomials $\hat{a}'$, $\hat{b}'$, s, r by b_q may be a too conservative choice. Indeed, additional information about these four polynomials is leaked* only *if an evaluation $q(X)$ is revealed on a point γ on which these polynomials were not already evaluated. More precisely, if the list $\mathcal{L}$ is such that the simulation of $t_{4,\gamma}$ does not require sampling new values $t_{j,\gamma}$, $j \in \{1, 2, 3, 5\}$, then it is sufficient to have $\hat{a}' \in \mathbb{F}_{\leq n+\mathsf{b}_a}$, $\hat{b}' \in \mathbb{F}_{\leq n+\mathsf{b}_b}$, $q_s \in \mathbb{F}_{\leq \mathsf{b}_s}$, $r_s \in \mathbb{F}_{\leq \mathsf{b}_r}$.*

Remark 2 ($\mathsf{PHP}_{\mathsf{lite1x}}$: a variant with fewer relation polynomials)**.** *We present a variant of $\mathsf{PHP}_{\mathsf{lite1}}$, that we call $\mathsf{PHP}_{\mathsf{lite1x}}$, whose difference with the former is a reduced number of relation polynomials. In particular, the offline phase of $\mathsf{PHP}_{\mathsf{lite1x}}$ outputs three less polynomials $\mathsf{col}'(X), \mathsf{row}'(X)$ and $\mathsf{cr}'(X)$. Here the second polynomial check has degree 3, with a publicly computable term X:*

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

4.3.2 A Variant with Separate Sparse Matrix Encodings

We propose a variant of the PHP for R1CS-lite $\mathsf{PHP}_{\mathsf{lite1}}$ described in the previous section. We call this protocol $\mathsf{PHP}_{\mathsf{lite2}}$. In $\mathsf{PHP}_{\mathsf{lite2}}$, the matrices $\{\boldsymbol{L}, \boldsymbol{R}\}$ are encoded in sparse form separately, namely without keeping track of common nonzero entries (see Definition 4.3). The main benefit of this choice is that in this case we can work with a subgroup $\mathbb{K} \subset \mathbb{F}$ of minimal size such that $|\mathbb{K}| \geq m$, which is half the size of the one needed in $\mathsf{PHP}_{\mathsf{lite1}}$.

Namely, $\boldsymbol{L}, \boldsymbol{R}$ can be represented with the functions $\{\mathsf{val}_M, \mathsf{row}_M, \mathsf{col}_M\}_{M\in\{L,R\}}$. Here, for $M = \{L, R\}$ and any $\kappa \in \mathbb{K}$, $\mathsf{val}_M(\kappa) = \boldsymbol{M}_{\mathsf{row}_M(\kappa),\mathsf{col}_M(\kappa)}$. We can use such sparse encoding of $\boldsymbol{L}$ and $\boldsymbol{R}$ to change the $V_{LR}(X, Y, Z)$ polynomial in equation (2) into the following one

$$\begin{aligned} V_{LR}(X, Y, Z) &= V_L(X, Y) + Y \cdot V_R(X, Y) \\ &= \sum_{\kappa\in\mathbb{K}} \Big(\mathsf{val}_L(\kappa) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}_L(\kappa)}(X) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{col}_L(\kappa)}(Y) + Z \cdot \mathsf{val}_R(\kappa) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}_R(\kappa)}(X) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{col}_R(\kappa)}(Y) \Big) \end{aligned}$$

Then in this variant the prover's goal is to show that the polynomials sent in the first round satisfy the equation above. This variant proceeds almost identically to the one of section 4.3.1; the only differences are in the relation polynomials and the third round.

Offline phase $\mathcal{RE}(\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}\}, \ell)$. The holographic relation encoder outputs 24 polynomials

$$\left\{\{\mathsf{v}_{M,i,j}(X)\}_{M\in\{L,R\},\{i,j\}\in\{0,1\}}, \{\mathsf{cr}_{i,j}(X), \mathsf{cr}'_{i,j}(X)\}_{i,j\in\{0,1,2\}\wedge i\neq 2\neq j}\right\} \in \mathbb{F}_{\leq|\mathbb{K}|}[X]$$

that are computed as follows. First, it finds the polynomials $\{\mathsf{val}_L, \mathsf{col}_L, \mathsf{row}_L, \mathsf{val}_R, \mathsf{col}_R, \mathsf{row}_R\}$ such that for all $\kappa \in \mathbb{K}$ $\mathsf{val}_L(\kappa) = \boldsymbol{L}_{\mathsf{row}_L(\kappa),\mathsf{col}_L(\kappa)}$ and $\mathsf{val}_R(\kappa) = \boldsymbol{R}_{\mathsf{row}_R(\kappa),\mathsf{col}_R(\kappa)}$. Second, it computes:

$$\begin{aligned}
\mathsf{v}_{L,0,0}(X) &:= \sum_{\kappa\in\mathbb{K}} \mathsf{val}_L(\kappa)\cdot\mathsf{col}_L(\kappa)\cdot\mathsf{row}_L(\kappa)\cdot\mathsf{col}_R(\kappa)\cdot\mathsf{row}_R(\kappa)\cdot\mathcal{L}^{\mathbb{K}}_\kappa(X)\\
\mathsf{v}_{L,0,1}(X) &:= \sum_{\kappa\in\mathbb{K}} \mathsf{val}_L(\kappa)\cdot\mathsf{col}_L(\kappa)\cdot\mathsf{row}_L(\kappa)\cdot\mathsf{row}_R(\kappa)\cdot\mathcal{L}^{\mathbb{K}}_\kappa(X)\\
\mathsf{v}_{L,1,0}(X) &:= \sum_{\kappa\in\mathbb{K}} \mathsf{val}_L(\kappa)\cdot\mathsf{col}_L(\kappa)\cdot\mathsf{row}_L(\kappa)\cdot\mathsf{col}_R(\kappa)\cdot\mathcal{L}^{\mathbb{K}}_\kappa(X)\\
\mathsf{v}_{L,1,1}(X) &:= \sum_{\kappa\in\mathbb{K}} \mathsf{val}_L(\kappa)\cdot\mathsf{col}_L(\kappa)\cdot\mathsf{row}_L(\kappa)\cdot\mathcal{L}^{\mathbb{K}}_\kappa(X)
\end{aligned}$$

and analogously $\{\mathsf{v}_{R,0,0}, \mathsf{v}_{R,0,1}, \mathsf{v}_{R,1,0}, \mathsf{v}_{R,1,1}\}$. Third, it computes

$$\begin{aligned}
\mathsf{cr}_{0,0}(X) &:= \sum_{\kappa\in\mathbb{K}} \mathsf{col}_L(\kappa)\cdot\mathsf{row}_L(\kappa)\cdot\mathsf{col}_R(\kappa)\cdot\mathsf{row}_R(\kappa)\cdot\mathcal{L}^{\mathbb{K}}_\kappa(X)\\
\mathsf{cr}_{0,1}(X) &:= -\sum_{\kappa\in\mathbb{K}} \mathsf{row}_L(\kappa)\cdot(\mathsf{col}_L(\kappa)\cdot\mathsf{row}_R(\kappa)+\mathsf{col}_R(\kappa)\cdot\mathsf{row}_R(\kappa))\cdot\mathcal{L}^{\mathbb{K}}_\kappa(X)\\
\mathsf{cr}_{1,0}(X) &:= -\sum_{\kappa\in\mathbb{K}} \mathsf{col}_L(\kappa)\cdot(\mathsf{col}_R(\kappa)\cdot\mathsf{row}_R(\kappa)+\mathsf{col}_R(\kappa)\cdot\mathsf{row}_L(\kappa))\cdot\mathcal{L}^{\mathbb{K}}_\kappa(X)\\
\mathsf{cr}_{1,1}(X) &:= \sum_{\kappa\in\mathbb{K}} (\mathsf{col}_L(\kappa)+\mathsf{col}_R(\kappa))\cdot(\mathsf{row}_R(\kappa)+\mathsf{row}_L(\kappa))\cdot\mathcal{L}^{\mathbb{K}}_\kappa(X)\\
\mathsf{cr}_{2,0}(X) &:= \sum_{\kappa\in\mathbb{K}} \mathsf{col}_L(\kappa)\cdot\mathsf{col}_R(\kappa)\cdot\mathcal{L}^{\mathbb{K}}_\kappa(X)\\
\mathsf{cr}_{0,2}(X) &:= \sum_{\kappa\in\mathbb{K}} \mathsf{row}_L(\kappa)\cdot\mathsf{row}_R(\kappa)\cdot\mathcal{L}^{\mathbb{K}}_\kappa(X)\\
\mathsf{cr}_{1,2}(X) &:= -\sum_{\kappa\in\mathbb{K}} (\mathsf{row}_L(\kappa)+\mathsf{row}_R(\kappa))\cdot\mathcal{L}^{\mathbb{K}}_\kappa(X)\\
\mathsf{cr}_{2,1}(X) &:= -\sum_{\kappa\in\mathbb{K}} (\mathsf{col}_L(\kappa)+\mathsf{col}_R(\kappa))\cdot\mathcal{L}^{\mathbb{K}}_\kappa(X)
\end{aligned}$$

as well as $\{\mathsf{cr}'_{i,j}(X) := X\cdot\mathsf{cr}_{i,j}(X)\}$

Online phase $\langle\mathcal{P}\left((\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}\}, \ell), \boldsymbol{x}, (a'(X), b'(X))\right), \mathcal{V}(\mathbb{F}, n, m, \mathsf{x})\rangle$. Round 1 and 2 proceed identically to the PHP of section 4.3.1 except for the different definition of the polynomial V_{LR}.

Round 3 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

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

The goal of the prover is to convince the verifier that

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

and for this it computes

$$
\begin{aligned}
r'(X) &:= \frac{p'(X) - \sigma/|\mathbb{K}|}{X} \in \mathbb{F}_{\leq |\mathbb{K}|-2}[X] \\
t(X) &= \frac{\sigma}{|\mathbb{K}|} \cdot n^2 \cdot \sum_{i,j \in [0,2]} x^i y^j \cdot \mathsf{cr}_{i,j}(X) + r'(X) \cdot n^2 \cdot \sum_{i,j \in [0,2]} x^i y^j \cdot \mathsf{cr}'_{i,j}(X) \\
&\qquad - \mathcal{Z}_{\mathbb{H}}(x) \cdot \mathcal{Z}_{\mathbb{H}}(y) \cdot \sum_{i,j \in [0,1]} x^i y^j \left(\mathsf{vcr}_{L,i,j}(X) + \alpha \cdot \mathsf{vcr}_{R,i,j}(X)\right)
\end{aligned}
$$

where $\mathsf{cr}_{2,2}(X) := 1$ and $\mathsf{cr}'_{2,2}(X) := X$, defines polynomial

$$
q'(X) := \frac{t(X)}{\mathcal{Z}_{\mathbb{K}}(X)} \in \mathbb{F}_{\leq |\mathbb{K}|-2}[X]
$$

and sends $\{q'(X), r'(X)\}$ to $\mathcal{V}$.

Decision phase. The degree checks and first polynomial check stay the same, while the second polynomial check is as follows

$$
\begin{aligned}
&\frac{\sigma}{|\mathbb{K}|} \cdot n^2 \cdot \sum_{i,j \in [0,2]} x^i y^j \cdot \mathsf{cr}_{i,j}(X) + r'(X) \cdot n^2 \cdot \sum_{i,j \in [0,2]} x^i y^j \cdot \mathsf{cr}'_{i,j}(X) \\
&\quad - \mathcal{Z}_{\mathbb{H}}(x) \cdot \mathcal{Z}_{\mathbb{H}}(y) \cdot \sum_{i,j \in [0,1]} x^i y^j \left(\mathsf{vcr}_{L,i,j}(X) + \alpha \cdot \mathsf{vcr}_{R,i,j}(X) \right) - q'(X) \cdot \mathcal{Z}_{\mathbb{K}}(X) \stackrel{?}{=} 0 \qquad (9)
\end{aligned}
$$

where $\mathsf{cr}_{2,2}(X) := 1$ and $\mathsf{cr}'_{2,2}(X) := X$.

By construction of the relation polynomials, observe that the check of equation (9) is equivalent to checking

$$
\begin{aligned}
&\left(X \cdot r'(X) + \frac{\sigma}{|\mathbb{K}|}\right) \cdot n^2 \cdot \prod_{M \in \{L,R\}} (x - \mathsf{row}_M(X)) \cdot (y - \mathsf{col}_M(X)) \\
&\; - \mathcal{Z}_{\mathbb{H}}(x) \cdot \mathcal{Z}_{\mathbb{H}}(y) \cdot \big(\mathsf{val}_L(X) \mathsf{col}_L(X) \mathsf{row}_L(X)(x - \mathsf{row}_R(X))(y - \mathsf{col}_R(X)) \\
&\qquad + \alpha \cdot \mathsf{val}_R(X) \mathsf{col}_R(X) \mathsf{row}_R(X)(x - \mathsf{row}_L(X))(y - \mathsf{col}_L(R)) \big) \stackrel{?}{=} 0 \bmod \mathcal{Z}_{\mathbb{K}}(X)
\end{aligned}
$$

Knowledge soundness and zero-knowledge of $\mathsf{PHP}_{\mathsf{lite2}}$ are essentially identical to those of $\mathsf{PHP}_{\mathsf{lite1}}$. The only differences concern polynomials that are produced by the relation encoder and thus are correct by definition.

Efficiency analysis. The relation encoder creates 24 polynomials of degree $\leq |\mathbb{K}|$, doable in time $O(|\mathbb{K}| \log |\mathbb{K}|)$. If expressed as functions of $|\mathbb{K}|$, the degree, proof length, prover complexity and verifier complexity are the same as in section 4.3.1. The only notable difference is that in this construction, in which we use separate sparse encodings for the matrices $\boldsymbol{L}, \boldsymbol{R}$, we have $|\mathbb{K}| \geq m$, unlike in the previous construction where it was $|\mathbb{K}| \geq 2m$.

Remark 3 ($\mathsf{PHP}_{\mathsf{lite2x}}$: a variant with fewer relation polynomials)**.** *We present a variant of* $\mathsf{PHP}_{\mathsf{lite2}}$*, that we call* $\mathsf{PHP}_{\mathsf{lite2x}}$*, whose difference with the former is a reduced number of relation polynomials. In particular, the offline phase of* $\mathsf{PHP}_{\mathsf{lite2x}}$ *outputs 8 less polynomials* $\mathsf{cr}'_{i,j}(X)$*. Here the second polynomial check has degree 3, with a publicly computable term* X*:*

$$
\begin{aligned}
&n^2 \cdot \left(X \cdot r'(X) + \frac{\sigma}{|\mathbb{K}|}\right) \cdot \sum_{i,j \in [0,2]} x^i y^j \cdot \mathsf{cr}_{i,j}(X) \\
&\quad - \mathcal{Z}_{\mathbb{H}}(x) \cdot \mathcal{Z}_{\mathbb{H}}(y) \cdot \sum_{i,j \in [0,1]} x^i y^j \left(\mathsf{vcr}_{L,i,j}(X) + \alpha \cdot \mathsf{vcr}_{R,i,j}(X) \right) - q'(X) \cdot \mathcal{Z}_{\mathbb{K}}(X) \stackrel{?}{=} 0 \qquad (10)
\end{aligned}
$$

4.4 Our PHP for R1CS

In this section we present our constructions of PHPs for R1CS. We give three constructions and two more variants that achieve different tradeoffs.

Recall that in R1CS we have a claim of the form $(\boldsymbol{L}\cdot\boldsymbol{z})\circ(\boldsymbol{R}\cdot\boldsymbol{z}) = \boldsymbol{O}\cdot\boldsymbol{z}$. In all our constructions we consider an equivalence of the R1CS relation in which we express all the checks merged into polynomial format as follows.

Definition 4.7 (Polynomial R1CS)**.** *Let $\mathbb{F}$ be a finite field and $n, m \in \mathbb{N}$ be positive integers. We define the universal relation $\mathcal{R}_{polyR1CS}$ as the set of triples*

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

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

$$\sum_{\eta\in\mathbb{H}} \big(Z_L \cdot a(\eta) + Z_R \cdot b(\eta) - Z_O \cdot a(\eta)b(\eta)\big)\cdot \mathcal{L}_{\eta}^{\mathbb{H}}(X) + \sum_{\substack{\eta,\eta'\in\mathbb{H}\\ M\in\{L,R,O\}}} Z_M \cdot \boldsymbol{M}_{\eta,\eta'} \cdot z(\eta')\cdot \mathcal{L}_{\eta}^{\mathbb{H}}(X) \;=\; 0 \quad (11)$$

where $\mathbb{L} := \{\phi_{\mathbb{H}}^{-1}(1), \ldots, \phi_{\mathbb{H}}^{-1}(\ell)\}$ and the above is a polynomial over $\mathbb{F}[X, Z_L, Z_R, Z_O]$.

The following simple lemma shows that the two relations are equivalent. For completeness, we give the proof in Appendix A.2.

Lemma 4.8. $\mathcal{L}(\mathcal{R}_{R1CS}) \equiv \mathcal{L}(\mathcal{R}_{polyR1CS})$.

4.4.1 Our Main PHP for R1CS

Here we present our first PHP for R1CS that we call $\mathsf{PHP}_{\mathsf{r1cs1}}$ and that uses a joint sparse encoding as stated in definition 4.4. The differences with $\mathsf{PHP}_{\mathsf{lite1}}$ are very subtle, and for this reason we only highlight the main keypoints and then show the full PHP in Figure 2.

Because $\mathcal{R}_{polyR1CS}$ requires one more matrix than $\mathcal{R}_{polyR1CS\text{-}lite}$, we must modify the main equation accordingly. In particular, we define a new matrix encoding polynomial V_{LRO}. The holographic relation encoder of this PHP requires more polynomials than in $\mathsf{PHP}_{\mathsf{lite1}}$, for the same reason. The protocol follows directly from these modifications, and the fact that the prover sends one more oracle, $\hat{w}(X)$, in the first round.

In this setting, we will need a multiplicative the subgroup be such that $|\mathbb{K}| \geq ||\boldsymbol{M}|| \leq 3m$ for any $\boldsymbol{M} \in \{\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O}\}$. The prover's goal is to convince the verifier that the polynomials $a(X), b(X), z(X)$ satisfy equation (11), which can be expressed as

$$0 \;=\; \sum_{\eta\in\mathbb{H}} \big(Z_L \cdot a(\eta) + Z_R\cdot b(\eta) - Z_O \cdot a(\eta)b(\eta)\big)\Lambda_{\mathbb{H}}(X,\eta) + z(\eta)\cdot V_{LRO}(X, \eta, Z_L, Z_R, Z_O) \quad (12)$$

where $V_{LRO} \in \mathbb{F}[X, Y, Z_L, Z_R, Z_O]$ is the following polynomial,

$$\begin{aligned} V_{LRO}(X, Y, Z_L, Z_R, Z_O) &= Z_L \cdot V_L(X,Y) + Z_R \cdot V_R(X,Y) + Z_O \cdot V_O(X,Y) \\ &= \sum_{\substack{\kappa\in\mathbb{K}\\ M\in\{L,R,O\}}} Z_M \cdot \mathsf{val_M}(\kappa)\cdot \mathcal{L}_{\mathsf{row}(\kappa)}^{\mathbb{H}}(X)\cdot \mathcal{L}_{\mathsf{col}(\kappa)}^{\mathbb{H}}(Y) \end{aligned}$$

Offline phase $\mathcal{RE}(\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O}\}, \ell)$. The holographic relation encoder takes as input a description of the specific relation and outputs 9 polynomials

$$\big\{\mathsf{row}(X), \mathsf{col}(X), \mathsf{cr}(X), \mathsf{row}'(X), \mathsf{col}'(X), \mathsf{cr}'(X), \{\mathsf{vcr}_M(X)\}_{M\in\{L,R,O\}}\big\} \in \mathbb{F}_{\leq|\mathbb{K}|-1}[X]$$

Online phase $\langle \mathcal{P}\left((\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O}\}, \ell), \boldsymbol{x}, (a(X), b(X), w(X))\right), \mathcal{V}(\mathbb{F}, n, m, \boldsymbol{x})\rangle$. The online phase of $\mathsf{PHP}_{\mathsf{r1cs1}}$ proceeds with the same round structure as in $\mathsf{PHP}_{\mathsf{lite1}}$. We refer the reader to Figure 2 for the full description of the protocol.

$\mathcal{P}((\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O}\}, \ell), \boldsymbol{x'}, (a(X), b(X), w(X)))$ $\mathcal{V}^{\mathsf{row},\mathsf{col},\mathsf{cr},\mathsf{row}',\mathsf{col}',\mathsf{cr}',\{\mathsf{vcr}_M\}}(\mathbb{F}, n, m, \boldsymbol{x'})$

Sample random $q_s \leftarrow\$ \mathbb{F}_{\mathsf{b}_s+\mathsf{b}_q-1}[X], r_s \leftarrow\$ \mathbb{F}_{\mathsf{b}_r+\mathsf{b}_q-1}[X]$

Set $s(X) \leftarrow q_s(X)\mathcal{Z}_{\mathbb{H}}(X) + Xr_s(X) \in \mathbb{F}_{\leq n+\mathsf{b}_s+\mathsf{b}_q-1}[X]$

Sample random $\hat{a}, \hat{b}, \hat{w}$ that agree with a, b, w on $\mathbb{H}$ → $\{\hat{a}, \hat{b}, \hat{w}, s\}$

$\hat{a}(X) \leftarrow\$ \mathsf{Mask}^{\mathbb{H}}_{\mathsf{b}_a+\mathsf{b}_q}(a(X)) \in \mathbb{F}_{\leq n+\mathsf{b}_a+\mathsf{b}_q-1}[X]$

$\hat{b}(X) \leftarrow\$ \mathsf{Mask}^{\mathbb{H}}_{\mathsf{b}_b+\mathsf{b}_q}(b(X)) \in \mathbb{F}_{\leq n+\mathsf{b}_b+\mathsf{b}_q-1}[X]$

$\hat{w}(X) \leftarrow\$ \mathsf{Mask}^{\mathbb{H}}_{\mathsf{b}_w+\mathsf{b}_q}(w(X)) \in \mathbb{F}_{\leq n-\ell+\mathsf{b}_w+\mathsf{b}_q-1}[X]$

$\hat{z}(X) := x'(X) + \hat{w}(X) \cdot \mathcal{Z}_{\mathbb{L}}(X) \in \mathbb{F}_{\leq n+\mathsf{b}_w+\mathsf{b}_q-1}[X]$ ← $x, \{\alpha_M\}_{\{L,R,O\}}$ — $x, \{\alpha_M\} \leftarrow\$ \mathbb{F}$

....// Sumcheck for "$\sum_{\eta\in\mathbb{H}} s(\eta) + p(\eta) = 0$" where

// $p(X) := (\alpha_L\hat{a}(X) + \alpha_R\hat{b}(X) - \alpha_O\hat{a}(X)\hat{b}(X))\Lambda_{\mathbb{H}}(x, X) + \hat{z}(X) \cdot V_{LRO}(x, X, \alpha_L, \alpha_R, \alpha_O) \in \mathbb{F}_{\leq 3n+\mathsf{b}_a+\mathsf{b}_b+2\mathsf{b}_q-3}[X]$

Compute $q \in \mathbb{F}_{\leq 2n+\mathsf{b}_a+\mathsf{b}_b+2\mathsf{b}_q-3}[X], r \in \mathbb{F}_{\leq n-2}[X]$ s.t. → $\{q, r\}$

$s(X) + p(X) = q(X) \cdot \mathcal{Z}_{\mathbb{H}}(X) + X \cdot r(X)$ ← y — $y \leftarrow\$ \mathbb{F} \setminus \mathbb{H}$

.......// Structured sumcheck for "$\sum_{\substack{\kappa\in\mathbb{K} \\ M\in\{L,R,O\}}} \alpha_M \cdot \mathsf{val_M}(\kappa) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(x) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{col}(\kappa)}(y) = V_{LRO}(x, y, \alpha_L, \alpha_R, \alpha_O)$".......

Let $\sigma \leftarrow V_{LRO}(x, y, \alpha_L, \alpha_R, \alpha_O)$

Compute $q'(X), r'(X) \in \mathbb{F}_{\leq|\mathbb{K}-2|}[X] : q'(X)\mathcal{Z}_{\mathbb{K}}(X) =$

$$n^2 \cdot \left(X \cdot r'(X) + \frac{\sigma}{|\mathbb{K}|}\right) \cdot (x - \mathsf{row}(X)) \cdot (y - \mathsf{col}(X))$$

$$- \mathcal{Z}_{\mathbb{H}}(x) \cdot \mathcal{Z}_{\mathbb{H}}(y) \cdot \mathsf{row}(x) \cdot \mathsf{col}(y) \cdot \sum_{M\in\{L,R,O\}} \alpha_M \mathsf{val_M}(X)$$ → $\sigma, \{q', r'\}$

Verifier's checks

- $\mathsf{deg}(\hat{a}), \mathsf{deg}(\hat{b}), \mathsf{deg}(\hat{w}), \mathsf{deg}(s), \mathsf{deg}(q), \mathsf{deg}(q') \overset{?}{\leq} \mathsf{D}_{snd} \wedge \mathsf{deg}(r) \overset{?}{\leq} n-2 \wedge \mathsf{deg}(r') \overset{?}{\leq} |\mathbb{K}| - 2$
- $$s(y) + \left(\alpha_L\, \hat{a}(y) + \alpha_R\, \hat{b}(y) - \alpha_O\, \hat{a}(y)\hat{b}(y)\right) \Lambda_{\mathbb{H}}(x, y)$$ $$+\sigma\Big(\sum_{\eta\in\mathbb{L}} \boldsymbol{x'}_{\phi_{\mathbb{H}}(\eta)}\mathcal{L}^{\mathbb{L}}_{\eta}(y) + \hat{w}(y)\, \mathcal{Z}_{\mathbb{L}}(y)\Big) - q(y)\, \mathcal{Z}_{\mathbb{H}}(y) - y\, r(y) \overset{?}{=} 0$$

$$\wedge \quad n^2 \cdot \frac{\sigma}{|\mathbb{K}|} \cdot (xy - x \cdot \mathsf{col}(X) - y \cdot \mathsf{row}(X) + \mathsf{cr}(X))$$
$$+n^2 \cdot r'(X) \cdot (xy \cdot X - x \cdot \mathsf{col}'(X) - y \cdot \mathsf{row}'(X) + \mathsf{cr}'(X))$$
$$-\mathcal{Z}_{\mathbb{H}}(x) \cdot \mathcal{Z}_{\mathbb{H}}(y) \cdot \sum_{M\in\{L,R,O\}} \alpha_M \cdot \mathsf{vcr}_M(X) - q'(X) \cdot \mathcal{Z}_{\mathbb{K}}(X) \overset{?}{=} 0$$

Figure 2: Our PHP protocol $\mathsf{PHP}_{\mathsf{r1cs1}}$ for R1CS.

4.4.1.1 Efficiency Analysis

We analyze the efficiency of our PHP protocol for R1CS with joint sparse encoding and $|\mathbb{K}| \geq 3m$.

Relation encoder It creates 9 polynomials, six of degree $\leq |\mathbb{K}|-1$ and the other three of degree $\leq |\mathbb{K}|$, doable in time $O(|\mathbb{K}| \log |\mathbb{K}|)$.

Degree By looking at the polynomials of the highest degree sent by indexer and prover, one can see that $D = \max\{2n + \mathsf{b}_a + \mathsf{b}_b + 2\mathsf{b}_q - 3, n + \mathsf{b}_s + \mathsf{b}_q - 1, |\mathbb{K}|\}$, which depends on the difference between $|\mathbb{H}|$ and $|\mathbb{K}|$, and the concrete values of $\mathsf{b}_a, \mathsf{b}_b, \mathsf{b}_q, \mathsf{b}_s$, which are small constants in our use cases. For example, when $m \geq n$ (which holds for matrices that encode arithmetic circuits), then $D = |\mathbb{K}|$.

Proof length. The prover sends one element of $\mathbb{F}$ and 8 oracle polynomials. By inspection, the proof length is $\mathsf{l}(\mathsf{R}) = 7n + 2|\mathbb{K}| - \ell + 2\mathsf{b}_a + 2\mathsf{b}_b + \mathsf{b}_w + \mathsf{b}_s + 6\mathsf{b}_q - 4$. By the structure of $s(X)$, we have that its number of nonzero coefficients is upperbounded by $\mathsf{b}_s + 2\mathsf{b}_q + \max\{\mathsf{b}_s, \mathsf{b}_r\}$, what gives us a proof length $\mathsf{l}(|\mathsf{R}|) = 6n + 2|\mathbb{K}| - \ell + 2\mathsf{b}_a + 2\mathsf{b}_b + \mathsf{b}_w + 2\mathsf{b}_s + 7\mathsf{b}_q - 4$.

Prover complexity. Using ideas similar to the ones for R1CS-lite, the total complexity is $O(|\mathbb{K}| \log |\mathbb{K}| + |\mathbb{H}| \log |\mathbb{H}|)$.

Verifier complexity. Similarly to the PHP for R1CS-lite, this this amounts to $O(\ell + \log |\mathbb{H}| + \log |\mathbb{K}|)$ field operations, which are needed to construct the polynomial checks.

4.4.1.2 Security Analysis

Theorem 4.9 (Knowledge Soundness)**.** *The PHP protocol* $\mathsf{PHP}_{\mathsf{r1cs1}}$ *described in section 4.4 is ϵ-sound with $\epsilon = \frac{2\mathsf{D}_{snd} + |\mathbb{H}|}{|\mathbb{F} \setminus \mathbb{H}|} + \frac{|\mathbb{H}|}{\mathbb{F}}$, and 0-knowledge sound.*

Proof. First we prove the soundness of this PHP, and then show its proof of knowledge property.

Soundness. Given the polynomial R1CS relation $\mathsf{R} = (\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O}\}, \ell)$ and input $\boldsymbol{x}$, assume there exists no witness $a(X), b(X), w(X)$ that satisfies the equation (11) of Definition 4.7. Since the relation encoder's polynomials are generated honestly (and thus are correct), there is no witness satisfying the equivalent equation (12) either. Then, for whatever polynomials $\hat{a}(X), \hat{b}(X), \hat{w}(X)$ sent by the prover $\mathcal{P}^*$ in the first round, it must be the case that

$$f(X, Z_L, Z_R, Z_O) := \sum_{\eta \in \mathbb{H}} \left(Z_L \cdot \hat{a}(\eta) + Z_R \cdot \hat{b}(\eta) - Z_O \cdot \hat{a}(\eta)\hat{b}(\eta)\right)\Lambda_{\mathbb{H}}(X, \eta) + \hat{z}(\eta) \cdot V_{LRO}(X, \eta, Z_L, Z_R, Z_O) \neq 0$$

for properly reconstructed $\hat{z}(X) := \sum_{\eta \in \mathbb{L}} \boldsymbol{x'}_{\phi_{\mathbb{H}}(\eta)} \cdot \mathcal{L}_\eta^{\mathbb{L}}(X) + \hat{w}(X) \cdot \mathcal{Z}_{\mathbb{L}}(X)$.

Let the protocol run as usual, then $\hat{a}(X), \hat{b}(X), \hat{w}(X), s(X), q(X), r(X), q'(X), r'(X)$ and σ are the polynomials and message sent by $\mathcal{P}^*$, and $x, \alpha_L, \alpha_R, \alpha_O, y$ are the messages from $\mathcal{V}$. Due to the order of the messages, we know that $\hat{a}(X), \hat{b}(X), \hat{w}(X), s(X)$ are independent of answers $x, \{\alpha_M\}$, and $\sigma, q(X), r(X)$ are independent of y.

Conditioned on the verifier accepting the proof, meaning that all degree and both polynomial checks are satisfied, we denote with $\mathtt{bad_1}$ and $\mathtt{bad_2}$ the events that the first and second polynomial checks hold when there exists no satisfying witness for the R1CS relation.

Given that the verifier accepted and the second polynomial check is deterministic, $\Pr(\mathtt{bad_2}) = 0$. This means that for all $\kappa \in \mathbb{K}$, the prover will find a polynomial $p'(X)$ such that $p'(\kappa) = \sum_{M \in \{L,R,O\}} \alpha_M \cdot \mathsf{val}_M(\kappa) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(x) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{col}(\kappa)}(y)$, as it does not depend on the witness. Considering the degree check on $r'(X)$, we have that $p'(X) := (X \cdot r'(X) + \frac{\sigma}{|\mathbb{K}|})$ is a polynomial of degree $\leq |\mathbb{K}| - 1$ that sums to σ on $\mathbb{K}$. Putting all of this together and considering the definition of $p'(X)$, we have that $\sigma = V_{LRO}(x, y, \alpha_L, \alpha_R, \alpha_O)$.

Next, since the polynomials $\hat{a}(X), \hat{b}(X), \hat{w}(X), s(X), q(X), r(X), V_{LRO}(x, X, \alpha_L, \alpha_R, \alpha_O)$ are independent of y, by the Schwartz-Zippel lemma we obtain that the first polynomial and degree checks imply that $q(X)\mathcal{Z}_{\mathbb{H}}(X) + Xr(X) = s(X) + p(X)$ holds with probability $\geq 1 - \frac{2\mathsf{D}_{snd} + |\mathbb{H}|}{|\mathbb{F} \setminus \mathbb{H}|}$ over the choice of $y \in \mathbb{F} \setminus \mathbb{H}$.

By the assumption on the nonexistence of a satisfying witness, the above equality can only hold when y happens to be a root

$$s(y)+\left(\alpha_L\hat{a}(y)+\alpha_R\hat{b}(y)-\alpha_O\hat{a}(y)\hat{b}(y)\right)\Lambda_{\mathbb{H}}(x,y)+\hat{z}(y)V_{LRO}(x,y,\alpha_L,\alpha_R,\alpha_O)-q(y)\mathcal{Z}_{\mathbb{H}}(y)-yr(y)=0,$$

which occurs with probability at most $\Pr(\mathsf{bad}_1)\le\frac{2\mathsf{D}_{snd}+|\mathbb{H}|}{|\mathbb{F}\setminus\mathbb{H}|}$.

The remaining degree check gives us that $r(X)\in\mathbb{F}_{\le n-2}[X]$, and thus by Lemma 4.2, we have that

$$\sum_{\eta\in\mathbb{H}}s(\eta)+f(x,\alpha_L,\alpha_R,\alpha_O)=0$$

Let $\varsigma=\sum_{\eta\in\mathbb{H}}s(\eta)$, since ς and $f(X,Z_L,Z_R,Z_O)$ are independent of $x,\{\alpha_M\}$, by Schwartz-Zippel we have that $\Pr[f(x,\alpha_L,\alpha_R,\alpha_O)+\varsigma=0]\le\frac{\mathbb{H}}{\mathbb{F}}$ over the choice of $x,\{\alpha_M\}\leftarrow_\$\mathbb{F}$. □

Knowledge Soundness. We define the extractor algorithm $\mathcal{E}$ that runs the prover $\mathcal{P}^*$ for the first round, obtains $\hat{a}(X),\hat{b}(X),\hat{w}(X)$, and reconstructs the nonrandomized witness polynomials by interpolation as $a(X)=\sum_{\eta\in\mathbb{H}}\hat{a}(\eta)\mathcal{L}^{\mathbb{H}}_\eta(X), b(X)=\sum_{\eta\in\mathbb{H}}\hat{b}(\eta)\mathcal{L}^{\mathbb{H}}_\eta(X), w(X)=\sum_{\eta\in\mathbb{H}}\hat{w}(\eta)\mathcal{L}^{\mathbb{H}}_\eta(X)$.

If the verifier accepts with probability greater than the soundness error ϵ given above, then the polynomials returned by $\mathcal{E}$ must encode a valid witness.

Finally, it is easy to see the straightline extractability. The algorithm WitExtract is the one that takes the polynomial $\hat{w}(X)$, and reconstructs the R1CS witness $\boldsymbol{w}$ by taking its evaluations on the points of $\mathbb{H}\setminus\mathbb{L}$—recall $w(X):=\sum_{\eta\in\mathbb{H}\setminus\mathbb{L}}\boldsymbol{w}_{\phi_{\mathbb{H}}(\eta)}\cdot\mathcal{L}^{\mathbb{H}\setminus\mathbb{L}}_\eta(X)$.

□

Theorem 4.10 (Zero-Knowledge)**.** *The PHP $\mathsf{PHP}_{\mathsf{r1cs1}}$ described in section 4.4.1 is perfect zero-knowledge. Furthermore, it is perfect honest-verifier zero-knowledge with query bound $\mathsf{b}=(\mathsf{b}_a,\mathsf{b}_b,\mathsf{b}_w,\mathsf{b}_s,\mathsf{b}_q,\mathsf{b}_r,\infty,\infty)$.*

Proof. We begin by showing the perfect zero-knowledge. As this scheme lies on the PHP model, there is no need to worry about oracle polynomials. Thus, we set our focus on the single non-oracle message σ that the prover sends throughout the rounds, which by the way does not depend on the witness. More formally, we describe a simulator $\mathcal{S}$ that on input the relation $\mathsf{R}=(\mathbb{F},n,m,\{\boldsymbol{L},\boldsymbol{R}\},\ell)$ and the input $\boldsymbol{x}$, and given oracle access to the verifier $\mathcal{V}^*$, proceeds as follows. It runs $\mathcal{V}^*$ to obtain its random messages x,y,α and its checks. Next, it computes $\sigma=V_{LR}(x,y,\alpha_L,\alpha_R,\alpha_O)$, and outputs σ followed by checks obtained from $\mathcal{V}^*$. Note that $\mathsf{View}\big(\mathcal{S}^{\mathcal{V}^*}(\mathbb{F},\mathsf{R},\boldsymbol{x})\big)$ is identically distributed to $\mathsf{View}\big(\mathcal{P}(\mathbb{F},\mathsf{R},\boldsymbol{x},a(X),b(X),w(X))\ ,\mathcal{V}^*\big)$.

Next, we prove b-HVZK for bounds $\mathsf{b}_a,\mathsf{b}_b,\mathsf{b}_w,\mathsf{b}_s,\mathsf{b}_q,\mathsf{b}_r$ on the polynomials $\hat{a}(X),\hat{b}(X),\hat{w}(X),s(X),q(X),r(X)$ respectively, whereas for the polynomials $q'(X),r'(X)$ we tolerate unbounded number of evaluations (this is trivial as these polynomials depend on public information only).

Let $\mathsf{C}(i,\gamma)$ be the algorithm that on any pair (i,γ) outputs 1 if and only if $i\in\{1,\dots,8\}$ and $\gamma\notin\mathbb{H}$. For a $\gamma\leftarrow_\$\mathbb{F}$, it holds $\Pr[\mathsf{C}(i,\gamma)=0]=|\mathbb{H}|/|\mathbb{F}|$, which is negligible for the choices of $\mathbb{F}$ considered in this paper.

The simulator samples a random tape $\boldsymbol{\rho}$ for the honest verifier and runs $Q_{\mathcal{V}}(\boldsymbol{\rho})$ to sample queries $(x,y,\{\alpha_M\}_{M\in\{L,R,O\}})$, and its decision algorithm $\{\boldsymbol{d},\{(G,\boldsymbol{v})\}\leftarrow D_{\mathcal{V}}(\mathbb{F},\boldsymbol{x};\boldsymbol{\rho})$ to obtain its checks. Then, it simulates answers to polynomial evaluations as follows.

For every pair (i,γ) with $i\in\{7,8\}$ (i.e., for every query on q',r'), the simulator computes $t_{i,\gamma}\leftarrow p_i(\gamma)$ honestly, which is trivial as these polynomials depend only on public information.

For every pair $(i,\gamma)\in\mathcal{L}$ such that $i\in[6]\setminus\{5\}$ (i.e., every query on $\hat{a},\hat{b},\hat{w},s,r$), the simulator samples a random value $t_{i,\gamma}\leftarrow_\$\mathbb{F}$ and stores a tuple $(i,\gamma,t_{i,\gamma})$ in a table T.

For every query $(5,\gamma)$ it simulates the answer with the value $t_{5,\gamma}$ computed as follows:

$$t_{z,\gamma}\leftarrow t_{3,\gamma}\cdot\mathcal{Z}_{\mathbb{L}}(\gamma)+\sum_{\eta\in\mathbb{L}}\boldsymbol{x'}_{\phi_{\mathbb{H}}(\eta)}\cdot\mathcal{L}^{\mathbb{L}}_\eta(\gamma)$$

$$t_{p,\gamma}\leftarrow(\alpha_L\cdot t_{1,\gamma}+\alpha_R\cdot t_{2,\gamma}-\alpha_O\cdot t_{1,\gamma}\cdot t_{2,\gamma})\cdot\Lambda_{\mathbb{H}}(x,\gamma)+t_{z,\gamma}\cdot V_{\{L,R,O\}}(x,\gamma,\alpha_L,\alpha_R,\alpha_O)$$

$$t_{5,\gamma}\leftarrow\frac{t_{p,\gamma}+t_{4,\gamma}-\gamma\cdot t_{6,\gamma}}{\mathcal{Z}_{\mathbb{H}}(\gamma)}$$

While doing the computations above, for $j = 1, 2, 3, 4, 6$, if an entry $(j, \gamma, t_{j,\gamma})$ already exists in T, then the corresponding value $t_{j,\gamma}$ is used; otherwise a random $t_{j,\gamma} \leftarrow\$ \mathbb{F}$ is sampled and a new entry $(j, \gamma, t_{j,\gamma})$ is added to T.

$\mathcal{S}$ returns $\big(\boldsymbol{\rho}, V_{LR}(x, y, \alpha_L, \alpha_R, \alpha_O), (\boldsymbol{d}, \{(G, \boldsymbol{v})\}), \{t_{i,\gamma}\}_{(i,\gamma)\in\mathcal{L}}\big)$.

To conclude the proof, we argue that the distribution of $\mathcal{S}$'s output is identical to that of

$$\big(\mathsf{View}\big(\mathcal{P}(\mathbb{F}, \mathsf{R}, \boldsymbol{x}, a, b, w)\ , \mathcal{V}\big), (p_i(\gamma))_{(i,\gamma)\in\mathcal{L}}\big)\,.$$

By the $(\mathsf{b}_a + \mathsf{b}_q)$-wise (resp. $(\mathsf{b}_b + \mathsf{b}_q)$ and $(\mathsf{b}_w + \mathsf{b}_q)$-wise) independence of the polynomial $\hat{a}(X)$ (resp. $\hat{b}(X)$ and $\hat{w}(X)$) sampled by the honest prover (and using the fact that they are evaluated on $\mathbb{F} \setminus \mathbb{H}$), we have that the set of simulated answers $\{t_{i,\gamma}\}_{i\in[3]:(i,\gamma)\in\mathcal{L}}$ are identically distributed (we recall that these sets are of size b_a, b_b and b_w respectively) to those of the real prover.

For the remaining polynomials, let us recall that for the honest prover we have

$$\begin{aligned} p(X) &= \big(\alpha_L \cdot \hat{a}(X) + \alpha_R \cdot \hat{b}(X) - \alpha_O \cdot \hat{a}(X) \cdot \hat{b}(X)\big) \cdot \Lambda_{\mathbb{H}}(x, X) + \hat{z}(X) \cdot V_{LRO}(x, X, \alpha_L, \alpha_R, \alpha_O) \\ s(X) &= q_s(X)\mathcal{Z}_{\mathbb{H}}(X) + X r_s(X) \end{aligned}$$

where $\hat{z}(X) = \hat{w}(X) \cdot \mathcal{Z}_{\mathbb{L}}(X) + \sum_{\eta\in\mathbb{L}} \boldsymbol{x'}_{\phi_{\mathbb{H}}(\eta)} \cdot \mathcal{L}^{\mathbb{L}}_{\eta}(X)$, $q_s(X) \leftarrow\$ \mathbb{F}_{\mathsf{b}_s+\mathsf{b}_q}[X]$ and $r_s(X) \leftarrow\$ \mathbb{F}_{\mathsf{b}_r+\mathsf{b}_q}[X]$. Also, let us write $p(X) = q_p(X)\mathcal{Z}_{\mathbb{H}}(X) + X r_p(X)$ for the unique $q_p(X), r_p(X)$ by polynomial division.

By the uniqueness of polynomials $q(X)$ and $r(X) \in \mathbb{F}_{\le n-2}[X]$ such that $s(X) + p(X) = q(X) \cdot \mathcal{Z}_{\mathbb{H}}(X) + X \cdot r(X)$, we have that $q(X) = q_p(X) + q_s(X)$ and $r(X) = r_p(X) + r_s(X)$.

By the $(\mathsf{b}_r + \mathsf{b}_q)$-wise independence of $r_s(X)$ (and thus of $r(X)$) we obtain that the set of simulated answers $\{t_{6,\gamma}\}_{(6,\gamma)\in\mathcal{L}}$ (whose cardinality is at most b_r) are identically distributed to those, $\{r(\gamma)\}_{(6,\gamma)\in\mathcal{L}}$, of the real prover. Furthermore, by the $(\mathsf{b}_s + \mathsf{b}_q)$-wise independence of $q_s(X)$ we obtain that the set of simulated answers $\{t_{4,\gamma}\}_{(4,\gamma)\in\mathcal{L}}$ (whose cardinality is at most b_s) are identically distributed to those, $\{s(\gamma)\}_{(4,\gamma)\in\mathcal{L}}$, of the real prover. In particular, for this we use that for $\gamma \in \mathbb{F} \setminus \mathbb{H}$, $s(X)$ is $(\mathsf{b}_s + \mathsf{b}_q)$-wise independent even conditioned on $r_s(X)$.

To argue the correct distribution of the set of simulated answers $\{t_{5,\gamma}\}_{(5,\gamma)\in\mathcal{L}}$, we observe that the honest $q(X)$ is determined by $(p(X) + s(X) - Xr(X))/\mathcal{Z}_{\mathbb{H}}(X)$, where $p(X)$ is defined as above. In particular, an evaluation of $q(\gamma)$ on $\gamma \in \mathbb{F} \setminus \mathbb{H}$ can be obtained as $(p(\gamma) + s(\gamma) - \gamma r(\gamma))/\mathcal{Z}_{\mathbb{H}}(\gamma)$, thus using evaluations of $\hat{a}(\gamma)$, $\hat{b}(\gamma)$, $\hat{w}(\gamma)$, $s(\gamma)$, $r(\gamma)$, and evaluations of publicly available polynomials. This explains the simulation strategy of $t_{5,\gamma}$ by $\mathcal{S}$, and these values are identically distributed to $q(\gamma)$ as the polynomials $\hat{a}(X)$, $\hat{b}(X)$, $\hat{w}(X)$, $s(X)$, and $r(X)$, each allows b_q more evaluations whose outputs are uniformly distributed.

□

Remark 4 (On degree optimizations)**.** *From the proof of the above theorem it turns out that increasing the degrees of polynomials $\hat{a}$, $\hat{b}$, $\hat{w}$, s, r by b_q may be a too conservative choice. Indeed, additional information about these four polynomials is leaked* only *if an evaluation $q(X)$ is revealed on a point γ on which these polynomials were not already evaluated. More precisely, if the list $\mathcal{L}$ is such that the simulation of $t_{5,\gamma}$ does not require sampling new values $t_{j,\gamma}$, $j \in \{1, 2, 3, 4, 6\}$, then it is sufficient to have $\hat{a} \in \mathbb{F}_{\le n+\mathsf{b}_a}$, $\hat{b} \in \mathbb{F}_{\le n+\mathsf{b}_b}$, $\hat{w} \in \mathbb{F}_{\le n+\mathsf{b}_w}$, $q_s \in \mathbb{F}_{\le \mathsf{b}_s}$, $r_s \in \mathbb{F}_{\le \mathsf{b}_r}$.*

Remark 5 (On the number of relation polynomials)**.** *We present a variant of $\mathsf{PHP}_{\mathsf{r1cs1}}$, that we call $\mathsf{PHP}_{\mathsf{r1cs1x}}$, whose difference with the former is a reduced number of relation polynomials. In particular, the offline phase of $\mathsf{PHP}_{\mathsf{r1cs1x}}$ outputs three less polynomials $\mathsf{col}'(X), \mathsf{row}'(X)$ and $\mathsf{cr}'(X)$. Here the second polynomial check has degree 3, with a publicly computable term X:*

$$n^2 \cdot \left(X \cdot r'(X) + \frac{\sigma}{|\mathbb{K}|}\right) \cdot \Big(xy - x \cdot \mathsf{col}(X) - y \cdot \mathsf{row}(X) + \mathsf{cr}(X)\Big) \\ - \mathcal{Z}_{\mathbb{H}}(x) \cdot \mathcal{Z}_{\mathbb{H}}(y) \cdot \sum_{M\in\{L,R,O\}} \alpha_M \cdot \mathsf{vcr}_M(X) - q'(X) \cdot \mathcal{Z}_{\mathbb{K}}(X) \stackrel{?}{=} 0 \qquad (13)$$

4.4.2 A Variant with Separate Sparse Matrix Encodings

Here we show a variant of our PHP for R1CS, in which the matrices $\{\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O}\}$ are encoded separately as in definition 4.3. We call this scheme $\mathsf{PHP}_{\mathsf{r1cs2}}$.

We can use such sparse encoding of $\boldsymbol{L}$, $\boldsymbol{R}$ and $\boldsymbol{O}$ to change the $V_{LRO}(X, Y, Z_L, Z_R, Z_O)$ polynomial in equation (12) into the following one:

$$V_{LRO}(X, Y, Z_L, Z_R, Z_O) \;=\; \sum_{\substack{\kappa\in\mathbb{K}\\ M\in\{L,R,O\}}} Z_M \cdot \mathsf{val}_{\mathsf{M}}(\kappa) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}_{\mathsf{M}}(\kappa)}(X) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{col}_{\mathsf{M}}(\kappa)}(Y)$$

Then in this variant the prover's goal is to show that the polynomials sent in the first round satisfy the equation above. This variant proceeds almost identically to the one of section 4.4.1; the main differences are in the relation polynomials and the third round.

Offline phase $\mathcal{RE}(\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O}\}, \ell)$. The holographic relation encoder outputs 57 polynomials,

$$\left\{\{\mathsf{cr}_{i,j}(X)\}_{i,j\in[0,3]\wedge i\neq 3\neq j}, \{\mathsf{v}_{M,i,j}(X)\}_{M\in\{L,R,O\}\wedge i,j\in[0,2]}\right\} \in \mathbb{F}_{\leq|\mathbb{K}|-1}[X]$$
$$\left\{\mathsf{cr}'_{i,j}(X) := X\cdot \mathsf{cr}_{i,j}(X)\right\}_{i,j\in[0,3]\wedge i\neq 3\neq j} \in \mathbb{F}_{\leq|K|}[X]$$

where $\mathsf{cr}_{i,j}(X)$ and $\mathsf{v}_{M,i,j}(X)$ are obtained by computing low-degree extensions of the polynomials that represent the coefficients accompanying the $x^i \cdot y^j$ terms of the following polynomials, respectively:

$$\prod_{M\in\{L,R,O\}} (x - \mathsf{row}_{\mathsf{M}}(X)) \cdot (y - \mathsf{col}_{\mathsf{M}}(X))$$
$$\sum_{M\in\{L,R,O\}} \mathsf{val}_{\mathsf{M}}(X) \cdot \mathsf{row}_{\mathsf{M}}(X) \cdot \mathsf{col}_{\mathsf{M}}(X) \cdot \prod_{M'\neq M} (x - \mathsf{row}_{\mathsf{M'}}(X)) \cdot (y - \mathsf{col}_{\mathsf{M'}}(X))$$

Similarly to $\mathsf{PHP}_{\mathsf{lite2}}$, the goal of all these polynomials is to obtain a verifier polynomial check that has at most degree 2 in the oracle polynomials.

Online phase $\langle \mathcal{P}\left((\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O}\}, \ell), \boldsymbol{x}, (a(X), b(X), w(X))\right), \mathcal{V}(\mathbb{F}, n, m, \mathsf{x})\rangle$. Round 1 and 2 proceed identically to the PHP of section 4.4.1 except for the different definition of the polynomial V_{LRO}.

Round 3 The verifier sends a random point $y \leftarrow_{\$} \mathbb{F}\setminus\mathbb{H}$. The prover uses y to compute $\sigma \leftarrow V_{LRO}(x, y, \alpha_L, \alpha_R, \alpha_O)$ and then defines the degree-$(|\mathbb{K}|-1)$ polynomial

$$V_{LRO}(X, Y, Z_L, Z_R, Z_O) \;=\; \sum_{\substack{\kappa\in\mathbb{K}\\ M\in\{L,R,O\}}} Z_M \cdot \mathsf{val}_{\mathsf{M}}(\kappa) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{row}_{\mathsf{M}}(\kappa)}(X) \cdot \mathcal{L}^{\mathbb{H}}_{\mathsf{col}_{\mathsf{M}}(\kappa)}(Y)$$

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

$$p'(\kappa) = \sum_{M\in\{L,R,O\}} \alpha_M \mathsf{val}_{\mathsf{M}}(\kappa) \mathcal{L}^{\mathbb{H}}_{\mathsf{row}_{\mathsf{M}}(\kappa)}(x) \mathcal{L}^{\mathbb{H}}_{\mathsf{col}_{\mathsf{M}}(\kappa)}(y)$$

Note that by decomposition of the Lagrangians this is equivalent to

$$\forall\, \kappa\in\mathbb{K}: \quad n^2 p'(\kappa) \prod_{M\in\{L,R,O\}} (x - \mathsf{row}_{\mathsf{M}}(\kappa))(y - \mathsf{col}_{\mathsf{M}}(\kappa)) -$$
$$\mathcal{Z}_{\mathbb{H}}(x)\mathcal{Z}_{\mathbb{H}}(y) \sum_{M\in\{L,R,O\}} \alpha_M \mathsf{val}_{\mathsf{M}}(\kappa)\mathsf{row}_{\mathsf{M}}(\kappa)\mathsf{col}_{\mathsf{M}}(\kappa) \prod_{M'\neq M} (x - \mathsf{row}_{\mathsf{M'}}(\kappa))(y - \mathsf{col}_{\mathsf{M'}}(\kappa)) = 0$$

that by using 42 of the relation polynomials and $\mathsf{cr}_{3,3}(X) = 1$ can be rewritten as

$$\forall\, \kappa\in\mathbb{K}: n^2 p'(\kappa) \sum_{i,j\in[0,3]} x^i \cdot y^j \cdot \mathsf{cr}_{i,j}(\kappa) - \mathcal{Z}_{\mathbb{H}}(x) \cdot \mathcal{Z}_{\mathbb{H}}(y) \sum_{\substack{i,j\in[0,2]\\ M\in\{L,R,O\}}} \alpha_M \cdot x^i \cdot y^j \cdot \mathsf{v}_{M,i,j}(\kappa) = 0$$

Then, $\mathcal{P}$ computes $r'(X) = (p'(X) - \frac{\sigma}{|\mathbb{K}|})/X \in \mathbb{F}_{\leq|\mathbb{K}|-2}[X]$ and $q'(X) := \frac{t(X)}{\mathcal{Z}_{\mathbb{K}}(X)} \in \mathbb{F}_{\leq|\mathbb{K}|-2}[X]$ with

$$t(X) = n^2 p'(X) \sum_{i,j\in[0,3]} x^i y^j \mathsf{cr}_{i,j}(X) - \mathcal{Z}_{\mathbb{H}}(x)\mathcal{Z}_{\mathbb{H}}(y) \sum_{\substack{i,j\in[0,2]\\ M\in\{L,R,O\}}} \alpha_M x^i y^j \mathsf{v}_{M,i,j}(X) \in \mathbb{F}_{\leq 2|\mathbb{K}|-2}[X]$$

and sends $\{q'(X), r'(X)\}$ to $\mathcal{V}$.

Decision phase. The degree checks and first polynomial check stay the same, while the second polynomial check using the 57 relation polynomials becomes the following

$$n^2 \frac{\sigma}{|\mathbb{K}|} \sum_{i,j\in[0,3]} x^i y^j \, \mathsf{cr}_{i,j}(X) + n^2 \, \mathsf{r}'(X) \sum_{i,j\in[0,3]} x^i y^j \, \mathsf{cr}'_{i,j}(X)$$
$$- \mathcal{Z}_{\mathbb{H}}(x) \mathcal{Z}_{\mathbb{H}}(y) \sum_{\substack{i,j\in[0,2] \\ M\in\{L,R,O\}}} \alpha_M x^i y^j \, \mathsf{v}_{M,i,j}(X) - q'(X) \, \mathcal{Z}_{\mathbb{K}}(X) \stackrel{?}{=} 0 \quad (14)$$

with $\mathsf{cr}_{3,3}(X) = 1$ and $\mathsf{cr}'_{3,3}(X) = X$.

Efficiency analysis. In this variant where nonzero entries are treated separately $|\mathbb{K}| \geq m$, unlike in the previous construction where it was $|\mathbb{K}| \geq 3m$. The relation encoder creates 42 polynomials of degree $\leq |\mathbb{K}| - 1$ and 15 of degree $\leq |\mathbb{K}|$, doable in time $O(|\mathbb{K}| \log |\mathbb{K}|)$. The degree, proof length, prover complexity and verifier complexity are the the same as in section 4.4.1. To summarize, the degree is $D = \max\{2n + \mathsf{b}_a + \mathsf{b}_b + 2\mathsf{b}_q - 3, n + \mathsf{b}_s + \mathsf{b}_q - 1, |\mathbb{K}|\}$, proof length is $\mathsf{l}(\mathsf{R}) \leq 6n + 2|\mathbb{K}| - \ell + 2\mathsf{b}_a + 2\mathsf{b}_b + \mathsf{b}_w + 2\mathsf{b}_s + 7\mathsf{b}_q - 4$, prover complexity is $O(|\mathbb{K}| \log |\mathbb{K}| + |\mathbb{H}| \log |\mathbb{H}|)$, while verifier's is $O(\ell + \log |\mathbb{H}| + \log |\mathbb{K}|)$.

Remark 6 (On the number of relation polynomials)**.** *We present a variant of* $\mathsf{PHP}_{\mathsf{r1cs2}}$*, that we call* $\mathsf{PHP}_{\mathsf{r1cs2x}}$*, whose difference with the former is a reduced number of relation polynomials. In particular, the offline phase of* $\mathsf{PHP}_{\mathsf{r1cs2x}}$ *outputs* 15 *less polynomials* $\mathsf{cr}'_{i,j}(X)$*. Here the second polynomial check has degree 3, with a publicly computable term* X*:*

$$n^2 \big(X \, \mathsf{r}'(X) + \frac{\sigma}{|\mathbb{K}|}\big) \sum_{i,j\in[0,3]} x^i y^j \, \mathsf{cr}_{i,j}(X) - \mathcal{Z}_{\mathbb{H}}(x) \mathcal{Z}_{\mathbb{H}}(y) \sum_{\substack{i,j\in[0,2] \\ M\in\{L,R,O\}}} \alpha_M x^i y^j \, \mathsf{v}_{M,i,j}(X) - q'(X) \, \mathcal{Z}_{\mathbb{K}}(X) \stackrel{?}{=} 0 \quad (15)$$

4.4.3 A Variant with Better Tradeoffs

Here we show another variant of a PHP for R1CS that presents a tradeoff between the number of relation polynomials and the degree of the second polynomial check. We call it $\mathsf{PHP}_{\mathsf{r1cs3}}$, and it will follow the separate sparse encoding of definition 4.3.

We will proceed as in $\mathsf{PHP}_{\mathsf{r1cs2}}$, which also follows this encoding. The main differences we highlight are in the relation polynomials and the final round.

Offline phase $\mathcal{RE}(\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O}\}, \ell)$. The holographic relation encoder outputs 12 polynomials describing the matrices of the R1CS,

$$\{\mathsf{row_M}, \mathsf{col_M}, \mathsf{cr_M}, \mathsf{vcr_M}, \}_{M\in\{L,R,O\}} \in \mathbb{F}_{\leq|\mathbb{K}|-1}[X]$$

where $\mathsf{cr_M}(X) := \sum_{\eta\in\mathbb{H}} \mathsf{col_M}(\eta) \cdot \mathsf{row_M}(\eta) \cdot \mathcal{L}^{\mathbb{H}}_\eta(X)$ and $\mathsf{vcr_M}(X) := \sum_{\eta\in\mathbb{H}} \mathsf{val_M}(\eta) \cdot \mathsf{col_M}(\eta) \cdot \mathsf{row_M}(\eta) \cdot \mathcal{L}^{\mathbb{H}}_\eta(X)$.

Online phase $\langle \mathcal{P}\left((\mathbb{F}, n, m, \{\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O}\}, \ell), \boldsymbol{x}, (a(X), b(X), w(X))\right), \mathcal{V}(\mathbb{F}, n, m, \mathsf{x})\rangle$. Round 1 and 2 proceed identically to the PHP of section 4.4.2, using the same definition of polynomial V_{LRO}.

Round 3 Here, the only difference comes when redefining the polynomial $t(X)$ which is now defined over $\mathbb{F}_{\leq 7|\mathbb{K}|-7}[X]$. This can be done using 9 relation polynomials as:

$$t(X) := n^2 \cdot p'(X) \prod_{M\in\{L,R,O\}} \big(x - \mathsf{row_M}(X)\big)\big(y - \mathsf{col_M}(X)\big)$$
$$- \mathcal{Z}_{\mathbb{H}}(x)\mathcal{Z}_{\mathbb{H}}(y) \sum_{M\in\{L,R,O\}} \alpha_M \cdot \mathsf{val_M}(X) \, \mathsf{col_M}(X) \, \mathsf{row_M}(X) \prod_{M'\neq M} \big(x - \mathsf{row_{M'}}(X)\big)\big(y - \mathsf{col_{M'}}(X)\big)$$

Nonetheless, this option will lead to a polynomial degree check of degree 8, which is undesirable for the verifier. Instead, we can make use of the other relation polynomials to obtain an equivalent definition of $t(X)$ with at most degree-5 checks:

$$t(X) := n^2 \cdot p'(X) \prod_{M\in\{L,R,O\}} \left(xy - y\mathsf{row_M}(X) - x\mathsf{col_M}(X) + \mathsf{cr_M}(X)\right)$$
$$- z_{\mathbb{H}}(x) z_{\mathbb{H}}(y) \sum_{M\in\{L,R,O\}} \alpha_M \cdot \mathsf{vcr_M}(X) \prod_{M'\neq M} \left(xy - y\mathsf{row_{M'}}(X) - x\mathsf{col_{M'}}(X) + \mathsf{cr_{M'}}(X)\right) \in \mathbb{F}_{\leq 4|\mathbb{K}|-4}[X]$$

As usual, the prover will send $\{q', r'\}$ to the verifier, such that $r'(X) = (p'(X) - \frac{\sigma}{|\mathbb{K}|})/X \in \mathbb{F}_{\leq |\mathbb{K}|-2}[X]$ and $q'(X) := \frac{t(X)}{z_{\mathbb{K}}(X)} \in \mathbb{F}_{\leq 3|\mathbb{K}|-4}[X]$.

Decision phase. The degree checks and first polynomial check stay the same, while the second one becomes the following check using the 9 relation polynomials

$$- q'(X)\, z_{\mathbb{K}}(X) + n^2 \left(X \cdot r'(X) + \frac{\sigma}{|\mathbb{K}|}\right) \prod_{M\in\{L,R,O\}} \left(xy - y\, \mathsf{row_M}(X) - x\, \mathsf{col_M}(X) + \mathsf{cr_M}(X)\right)$$
$$- z_{\mathbb{H}}(x) z_{\mathbb{H}}(y) \sum_{M\in\{L,R,O\}} \alpha_M\, \mathsf{vcr_M}(X) \prod_{M'\neq M} \left(xy - y\, \mathsf{row_{M'}}(X) - x\, \mathsf{col_{M'}}(X) + \mathsf{cr_{M'}}(X)\right) \stackrel{?}{=} 0 \quad (16)$$

Efficiency analysis. In this variant where nonzero entries are treated separately, $|\mathbb{K}| \geq m$ as well. The relation encoder creates 12 polynomials of degree $\leq |\mathbb{K}|-1$, doable in time $O(|\mathbb{K}| \log |\mathbb{K}|)$. Note that the quotient polynomial sent in the third round has much larger degree now, which becomes $D = \max(2n + \mathsf{b}_a + \mathsf{b}_b + 2\mathsf{b}_q - 3, 3|\mathbb{K}| - 4)$. The proof length, prover complexity and verifier complexity are the the same as in section 4.4.1. To summarize, the proof length is $\mathsf{l(R)} \leq 6n + 2|\mathbb{K}| - \ell + 2\mathsf{b}_a + 2\mathsf{b}_b + \mathsf{b}_w + 2\mathsf{b}_s + 7\mathsf{b}_q - 4$, prover complexity is $O(|\mathbb{K}| \log |\mathbb{K}| + |\mathbb{H}| \log |\mathbb{H}|)$, while verifier's is $O(\ell + \log |\mathbb{H}| + \log |\mathbb{K}|)$.

5 Preliminaries on Commitments and zkSNARKs

5.1 Commitment Schemes

In our work we use the notion of *type-based commitments*. Type-based commitments, introduced by Escala and Groth [EG14], are a generalization of regular commitments that unify several committing methods into the same scheme. This capability can be useful when committing to values from different domains (e.g., elements from one of the bilinear groups $\mathbb{G}_1, \mathbb{G}_2$, as in the original motivation of [EG14]), or when creating commitments with different security properties (e.g., some that are hiding and some that are not). As done in [BCFK19], in this work we will exploit the formalism of type-based commitments to describe commit-and-prove zero-knowledge proofs that work with commitments of different types[12].

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 represent the type of c. All the possible types are included in the type space $\mathcal{T}$.

Definition 5.1 (Typed-Based Commitment Schemes)**.** *A typed-based commitment scheme for a set of types $\mathcal{T}$ and with message space $\mathcal{M}$ is a tuple of algorithms $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCom})$ that work as follows:*

$\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{ck}$ *takes the security parameter and outputs a commitment key* ck.

$\mathsf{Commit}(\mathsf{ck}, \mathsf{type}, m) \rightarrow (c, o)$: *takes the commitment key* ck*, a type* $\mathsf{type} \in \mathcal{T}$ *and a message* $m \in \mathcal{M}$*, and outputs a commitment c and an opening o. We assume c contains information about its type, which we denote by* $\mathsf{type}(c)$.

[12] Our notion of type-based commitments is analogous to that in [BCFK19] with one exception: we allow the same message space, e.g. the set of polynomials, to be associated with different types; we see a type as a device different sets of properties from a commitment scheme in a fine-grained manner.

$\mathsf{VerCom}(\mathsf{ck}, \mathsf{type}, c, m, o) \to b:$ *takes as input the commitment key* ck*, a type* $\mathsf{type} \in \mathcal{T}$*, a commitment* c*, a message* $m \in \mathcal{M}$ *and an opening* o*, and it accepts* ($b=1$) *or rejects* ($b=0$)*. By default it outputs* 0 *if* $\mathsf{type}(c) \neq \mathsf{type}$*. Additionally we define* $\mathsf{VerCom}(\mathsf{ck}, c, f, o)$ *that runs* $\mathsf{VerCom}(\mathsf{ck}, \mathsf{type}(c), c, f, o)$*.*

CS *satisfies correctness,* type*-typed binding and* type*-typed trapdoor-hiding properties defined below:*

Correctness. *For any* $\lambda \in \mathbb{N}$*, any commitment key* $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$*, type* $\mathsf{type} \in \mathcal{T}$*, message* $m \in \mathcal{M}$*, and for any honestly generated commitment-opening* $(c, o) \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathsf{type}, m)$*, we have that* $\mathsf{VerCom}(\mathsf{ck}, \mathsf{type}, c, m, o) = 1$*;*

type**-typed Binding.** *Let* $\mathsf{type} \in \mathcal{T}$*,* CS *is* type*-typed (computationally) binding if for every (non-uniform) efficient adversary* $\mathcal{A}$ *we have* $\Pr[\mathsf{Game}^{\mathsf{bind}}_{\mathcal{A}}(\lambda) = 1] = \mathsf{negl}(\lambda)$ *where:*

$\mathsf{Game}^{\mathsf{bind}}_{\mathcal{A}}(\lambda)$

$\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$

$c, m, o, m', o' \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{type}, \mathsf{aux}_Z)$

return $\mathsf{VerCom}(\mathsf{ck}, \mathsf{type}, c, m, o) \stackrel{?}{=} 1 \wedge \mathsf{VerCom}(\mathsf{ck}, \mathsf{type}, c, m', o') \stackrel{?}{=} 1 \wedge m \neq m'$

We simply say that CS *is binding if it is* type*-typed binding for any* $\mathsf{type} \in \mathcal{T}$*.*

type**-typed Trapdoor-Hiding.** *There exist three algorithms* $(\mathsf{ck}, \mathsf{td}) \leftarrow \mathcal{S}_{\mathsf{ck}}(1^\lambda)$*,* $(c, st) \leftarrow \mathsf{TdCom}(\mathsf{td}, \mathsf{type})$ *and* $o \leftarrow \mathsf{TdOpen}(\mathsf{td}, st, \mathsf{type}, c, m)$ *such that: the distribution of the commitment key returned by* $\mathcal{S}_{\mathsf{ck}}$ *is perfectly/statistically close to the one of the key returned by* Setup*; for any* $m \in \mathcal{M}$ *,* $(c, o) \approx (c', o')$ *where* $(c, o) \leftarrow \mathsf{Commit}(\mathsf{ck}, m)$*,* $(c', st) \leftarrow \mathsf{TdCom}(\mathsf{td}, \mathsf{type})$ *and* $o' \leftarrow \mathsf{TdOpen}(\mathsf{td}, st, \mathsf{type}c', m)$*.*

Definition 5.2 (Succinct Commitments)**.** *A commitment scheme* CS *is said* succinct *if there is a fixed polynomial that bounds the size of every commitment* c *returned by* Commit*; in particular* $|c|$ *may be independent of the size of the message.*

5.2 Preprocessing zkSNARKs with Universal and Specializable SRS

In a recent work, Groth et al. [GKM+18] introduced the notion of (preprocessing) zkSNARKs with specializable universal structured reference string (SRS). In a nutshell, this notion formalizes the idea that key generation for $\mathsf{R} \in \mathcal{R}$ can be seen as the sequential combination of two steps: a first probabilistic algorithm that generates an SRS for the universal relation $\mathcal{R}$ and a second deterministic algorithm that specializes this universal SRS into one for a specific R. We remark that by considering "universal relations" $\mathcal{R}$ that contain a single R, and by having Derive as the identity function, one recovers the usual zkSNARK notion.

We consider families of relations parametrized by the output of a probabilistic algorithm $\mathsf{ParGen}(1^\lambda) \to \mathsf{pp}$ that takes as input the security parameter and outputs a set of relation parameters pp. The families also depend on a size bound N; we denote them as a tuple $\left(\mathsf{ParGen}, \{\mathcal{R}_{\mathsf{pp},\mathsf{N}}\}_{\mathsf{pp}\in\{0,1\}^*,\mathsf{N}\in\mathbb{N}}\right)$. Occasionally, as in the definition of CP-SNARK, we will consider "simple" relation families $\mathcal{R}$ parametrized only by a bound $\mathsf{N} \in \mathbb{N}$.

Definition 5.3 (Universal zkSNARK)**.** *A SNARK with specializable universal SRS for a family of relations* $\left(\mathsf{ParGen}, \{\mathcal{R}_{\mathsf{pp},\mathsf{N}}\}_{\mathsf{pp}\in\{0,1\}^*,\mathsf{N}\in\mathbb{N}}\right)$ *is a tuple of algorithms* $\Pi = (\mathsf{KeyGen}, \mathsf{Derive}, \mathsf{Prove}, \mathsf{Verify})$ *that work as described below and that satisfy the notions of* completeness*,* succinctness *and* knowledge-soundness *defined below. If* Π *also satisfies* zero-knowledge *we call it a universal zkSNARK.*

- $\mathsf{KeyGen}(\mathsf{pp}, \mathsf{N}) \to (\mathsf{srs}, \mathsf{td}_\mathsf{k})$ *is a* probabilistic *algorithm that takes as input the public parameters for the relation family and it outputs a* $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk})$*. We assume without loss of generality that* srs *contains* pp *output of* ParGen*.*
- $\mathsf{Derive}(\mathsf{srs}, \mathsf{R}) \to (\mathsf{ek}_\mathsf{R}, \mathsf{vk}_\mathsf{R})$ *is a* deterministic *algorithm that takes as input an* srs *produced by* $\mathsf{KeyGen}(\mathsf{pp}, \mathsf{N})$*, and a relation* $\mathsf{R} \in \mathcal{R}_\mathsf{N}$*, and outputs specialized keys* $\mathsf{srs}_\mathsf{R} := (\mathsf{ek}_\mathsf{R}, \mathsf{vk}_\mathsf{R})$*.*

- $\mathsf{Prove}(\mathsf{ek_R}, \mathsf{x}, \mathsf{w}) \to \pi$ *takes a proving key* $\mathsf{ek_R}$ *for a relation* R*, a statement* x*, and a witness* w *such that* $\mathsf{R}(\mathsf{x}, \mathsf{w})$ *holds, and returns a proof* π*.*
- $\mathsf{Verify}(\mathsf{vk_R}, \mathsf{x}, \pi) \to b$ *takes a verification key for a relation* R*, a statement* x*, and either accepts* $(b = 1)$ *or rejects* $(b = 0)$ *the proof* π*.*

Completeness. *For all* $\mathsf{pp} \in \mathsf{Range}(\mathsf{ParGen}), \mathsf{N} \in \mathbb{N}$*,* $\mathsf{R} \in \mathcal{R}_{\mathsf{pp},\mathsf{N}}$ *and* (x, w) *such that* $\mathsf{R}(\mathsf{x}, \mathsf{w}) = 1$*, it holds:*

$$\Pr\left[\begin{array}{c}(\mathsf{srs}, \mathsf{td_k}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{N}) \\ (\mathsf{ek_R}, \mathsf{vk_R}) \leftarrow \mathsf{Derive}(\mathsf{srs}, \mathsf{R}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{ek_R}, \mathsf{x}, \mathsf{w})\end{array} : \mathsf{Verify}(\mathsf{vk_R}, \mathsf{x}, \pi) = 1\right] = 1$$

Succinctness. Π *is said* succinct *if the running time of* Verify *is* $\mathsf{poly}(\lambda + |\mathsf{x}| + \log |\mathsf{w}|)$ *and the proof size is* $\mathsf{poly}(\lambda + \log |\mathsf{w}|)$*.*

Knowledge Soundness. *Let* $\mathsf{N} = \mathsf{poly}(\lambda)$*, we say* Π *has knowledge soundness for an auxiliary input distribution* $\mathcal{Z}$*, denoted* $\mathsf{KSND}(\mathcal{Z})$ *for brevity, if for every (non-uniform) efficient adversary* $\mathcal{A}$ *there exists a (non-uniform) efficient extractor* $\mathcal{E}$ *such that* $\Pr\left[\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{Z},\mathcal{A},\mathcal{E}}(\lambda) = 1\right] = \mathsf{negl}(\lambda)$*. We say that* Π *is knowledge-sound if there exists benign* $\mathcal{Z}$ *such that* Π *is* $\mathsf{KSND}(\mathcal{Z})$*.*

$$\underline{\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}}(\lambda) \to b}$$
$$\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{ParGen}(1^\lambda) \\ (\mathsf{srs}, \mathsf{td_k}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{N}) \\ \mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathsf{srs}) \\ (\mathsf{R}, \mathsf{x}, \pi) \leftarrow \mathcal{A}(\mathsf{srs}, \mathsf{aux}_Z) \\ \mathsf{w} \leftarrow \mathcal{E}(\mathsf{srs}, \mathsf{aux}_Z) \\ \mathsf{vk_R} \leftarrow \mathsf{Derive}(\mathsf{srs}, \mathsf{R}) \\ b = \mathsf{Verify}(\mathsf{vk_R}, \mathsf{x}, \pi) \ \wedge \neg \mathsf{R}(\mathsf{x}, \mathsf{w})\end{array}$$

Zero-Knowledge in SRS Model. *We say* Π *is zero-knowledge if there exists a simulator* $\mathcal{S}$ *such that for all adversaries* $\mathcal{A}$*, for all* $\mathsf{pp} \in \mathsf{Range}(\mathsf{ParGen}), \mathsf{N} \in \mathbb{N}$*, for all* $\mathsf{R} \in \mathcal{R}_{\mathsf{pp},\mathsf{N}}$*, and for all* (x, w) *such that* $\mathsf{R}(\mathsf{x}, \mathsf{w}) = 1$*,*

$$\Pr\left[\begin{array}{r}\mathsf{pp} \leftarrow \mathsf{ParGen}(1^\lambda) \\ (\mathsf{srs}, \mathsf{td_k}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{N}) \\ \mathsf{srs_R} \leftarrow \mathsf{Derive}(\mathsf{srs}, \mathsf{R}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{srs_R}, \mathsf{x}, \mathsf{w})\end{array} : \mathcal{A}(\mathsf{srs}, \mathsf{td_k}, \mathsf{R}, \mathsf{x}, \mathsf{w}, \pi) = 1\right] \approx$$
$$\Pr\left[\begin{array}{r}\mathsf{pp} \leftarrow \mathsf{ParGen}(1^\lambda) \\ (\mathsf{srs}, \mathsf{td_k}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{N}) \\ \pi \leftarrow \mathcal{S}(\mathsf{td_k}, \mathsf{R}, \mathsf{x})\end{array} : \mathcal{A}(\mathsf{srs}, \mathsf{td_k}, \mathsf{R}, \mathsf{x}, \mathsf{w}, \pi) = 1\right]$$

5.3 Commit-and-Prove Universal SNARKs

Here we adapt the notion of commit-and-prove SNARKs of [CFQ19] to universal relations.

Definition 5.4 (Universal CP-SNARKs). *Let* $\{\mathcal{R}_\mathsf{N}\}_{\mathsf{N} \in \mathbb{N}}$ *be a simple family of relations* R *over* $\mathcal{D}_\mathsf{x} \times \mathcal{D}_\mathsf{u} \times \mathcal{D}_\omega$ *such that* $\mathcal{D}_\mathsf{u}$ *splits over* ℓ *arbitrary domains* $(\mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell)$ *for some arity parameter* $\ell \geq 1$*. Let* $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCom})$ *be a commitment scheme (as per Definition 5.1) whose input space* $\mathcal{D}$ *is such that* $\mathcal{D}_i \subset \mathcal{D}$ *for all* $i \in [\ell]$*. A universal commit and prove zkSNARK for* CS *and* $\{\mathcal{R}_\mathsf{N}\}_{\mathsf{N} \in \mathbb{N}}$ *is a zkSNARK for a family of relations* $\left(\mathsf{ParGen} = \mathsf{CS.Setup}, \{\mathcal{R}^{\mathsf{Com}}_{\mathsf{ck},\mathsf{N}}\}_{\mathsf{ck} \in \{0,1\}^*, \mathsf{N} \in \mathbb{N}}\right)$ *such that:*

- *every* $\mathsf{R}^{\mathsf{Com}}_{\mathsf{ck},\mathsf{N}} \in \mathcal{R}^{\mathsf{Com}}$ *is represented by a pair* $(\mathsf{ck}, \mathsf{R})$ *where* $\mathsf{N} = \mathsf{poly}(\lambda)$*,* $\mathsf{ck} \in \mathsf{Setup}(1^\lambda)$ *and* $\mathsf{R} \in \mathcal{R}_\mathsf{N}$*;*

- $\mathsf{R}^{\mathsf{Com}}_{\mathsf{ck},\mathsf{N}}$ *is over pairs* $(\hat{\mathsf{x}}, \hat{\mathsf{w}})$ *where the statement is* $\hat{\mathsf{x}} := (\mathsf{x}, (c_j)_{j\in[\ell]}) \in \mathcal{D}_{\mathsf{x}} \times \mathcal{C}^{\ell}$*, the witness is* $\hat{\mathsf{w}} := ((\mathsf{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega) \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_{\ell} \times \mathcal{O}^{\ell} \times \mathcal{D}_{\omega}$*, and the relation* $\mathsf{R}^{\mathsf{Com}}_{\mathsf{ck},\mathsf{N}}$ *holds if and only if*

$$\bigwedge\nolimits_{j\in[\ell]} \mathsf{VerCom}(\mathsf{ck}, c_j, \mathsf{u}_j, o_j) = 1 \wedge \mathsf{R}(\mathsf{x}, (\mathsf{u}_j)_{j\in[\ell]}, \omega) = 1$$

We denote a Universal CP-SNARK as a tuple of algorithms $\mathsf{CP} = (\mathsf{KeyGen}, \mathsf{Derive}, \mathsf{Prove}, \mathsf{Verify})$. For ease of exposition, in our constructions we adopt the syntax for CP's algorithms defined below.

- $\mathsf{KeyGen}(\mathsf{ck}, \mathsf{N}) \to \mathsf{srs} := (\mathsf{ek}, \mathsf{vk})$ generates the structured reference string.
- $\mathsf{Derive}(\mathsf{srs}, \mathsf{R}) \to \mathsf{vk}_{\mathsf{R}}$ is a deterministic algorithm that takes as input a srs produced by $\mathsf{KeyGen}(1^{\lambda}, \mathsf{N})$, and a relation $\mathsf{R} \in \mathcal{R}_{\mathsf{N}}$.
- $\mathsf{Prove}(\mathsf{ek}, \mathsf{x}, (c_j)_{j\in[\ell]}, (\mathsf{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega) \to \pi$ outputs the proof.
- $\mathsf{Verify}(\mathsf{vk}_{\mathsf{R}}, \mathsf{x}, (c_j)_{j\in[\ell]}, \pi) \to b \in \{0, 1\}$ rejects or accepts the proof.

Type-restricted completeness. In the CP-SNARK notion presented above, the CP-SNARK is required to work on commitments of any type. Here we define a weaker notion of completeness in which the CP-SNARK works only when certain witnesses are committed with a specific type. This is useful if we want to use a CP-SNARK that supports only a subset 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. The following definition formalizes this completeness notion of CP-SNARKs:

Definition 5.5. *Let* $\{\mathcal{R}_{\mathsf{N}}\}_{\mathsf{N}\in\mathbb{N}}$ *be a family of relations* R *over* $\mathcal{D}_{\mathsf{x}} \times \mathcal{D}_{\mathsf{u}} \times \mathcal{D}_{\omega}$ *such that* $\mathcal{D}_{\mathsf{u}} = \mathcal{D}^{\ell}$ *for* $\ell \in \mathbb{N}$*. Let* CS *be a commitment scheme with types set* $\mathcal{T}$ *and message space* $\mathcal{D} \subseteq \mathcal{M}$ *and let* $T \in \mathcal{T}^{\ell}$*.*

A CP-SNARK scheme CP *is* T*-restricted complete if for every* $\mathsf{N} \in \mathbb{N}$*,* $\mathsf{R} \in \mathcal{R}_{\mathsf{N}}$ *and* $((\mathsf{x}, (c_j)_{j\in[\ell]}), \hat{\mathsf{w}})$ *such that* $\mathsf{R}^{\mathsf{Com}}_{\mathsf{ck},\mathsf{N}}((\mathsf{x}, (c_j)_{j\in[\ell]}), \hat{\mathsf{w}}) = 1$*,* *and for all* $j \in [\ell] : \mathsf{type}(c_j) = T_j$ *it holds:*

$$\Pr\left[\ (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{ParGen}(1^{\lambda}), \mathsf{N})\ , \pi \leftarrow \mathsf{Prove}(\mathsf{ek}, \hat{\mathsf{x}}, \hat{\mathsf{w}}) : \mathsf{Verify}(\mathsf{Derive}(\mathsf{srs}, \mathsf{R}), \hat{\mathsf{x}}, \pi) = 1\ \right] = 1$$

For $\mathcal{T}' \subset \mathcal{T}^{\ell}$ *we say that CP-SNARK scheme* CP *is* $\mathcal{T}'$*-restricted complete if for all* $T \in \mathcal{T}'$ *it is* T*-restricted complete.*

Commitment-only SRS. The following definition formalizes a property common to several schemes.

Definition 5.6 (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 defined in Def. 5.3 is not sufficient. In fact, formally speaking, the commitment key ck is part of the description of relation, thus the actual SRS of the CP-SNARK would be the empty string. However, the classical result of [GO94] showed that NIZK in the plain model exists only for trivial languages. Therefore we consider a weaker notion of zero-knowledge where the trapdoor necessary for simulation comes from the commitment key of CS.

Definition 5.7. *A universal CP-SNARK* CP *is trapdoor-commitment zero-knowledge in the SRS model for a family of universal relations* $\{\mathcal{R}_{\mathsf{N}}\}_{\mathsf{N}\in\mathbb{N}}$ *if there exists a simulator* $\mathcal{S}$ *such that for all adversaries* $\mathcal{A}$*,* $\mathsf{N} \in \mathbb{N}$*,* $\mathsf{R} \in \mathcal{R}_{\mathsf{N}}$*,* $(\mathsf{ck}, \mathsf{td}) \in \mathcal{S}_{\mathsf{ck}}(1^{\lambda})$*, and* $\hat{\mathsf{x}}, \hat{\mathsf{w}}$ *such that* $\mathsf{R}^{\mathsf{Com}}_{\mathsf{ck},\mathsf{N}}(\hat{\mathsf{x}}, \hat{\mathsf{w}}) = 1$*:*

$$\Pr\begin{bmatrix} (\mathsf{srs}, \mathsf{td}_{\mathsf{k}}) \leftarrow \mathsf{KeyGen}(\mathsf{ck}, \mathsf{N}) \\ \mathsf{srs}_{\mathsf{R}} \leftarrow \mathsf{Derive}(\mathsf{srs}, \mathsf{R}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{srs}_{\mathsf{R}}, \mathsf{x}, \mathsf{w}) \end{bmatrix} : \mathcal{A}(\mathsf{srs}, \mathsf{td}_{\mathsf{k}}, \mathsf{td}, \mathsf{R}, \mathsf{x}, \mathsf{w}, \pi) = 1 \Bigg] \approx$$

$$\Pr\begin{bmatrix} (\mathsf{srs}, \mathsf{td}_{\mathsf{k}}) \leftarrow \mathsf{KeyGen}(\mathsf{ck}, \mathsf{N}) \\ \pi \leftarrow \mathcal{S}(\mathsf{td}_{\mathsf{k}}, \mathsf{td}, \mathsf{R}, \mathsf{x}) \end{bmatrix} : \mathcal{A}(\mathsf{srs}, \mathsf{td}_{\mathsf{k}}, \mathsf{td}, \mathsf{R}, \mathsf{x}, \mathsf{w}, \pi) = 1 \Bigg]$$

where $\hat{\mathsf{x}} = (\mathsf{x}, (c_j)_{j\in[\ell]})$ *and* $\hat{\mathsf{w}} = ((\mathsf{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega)$*.*

When the CP-SNARK CP is for a family of relations $\{\mathcal{R}_{\mathsf{N}}\}_{\mathsf{N}\in\mathbb{N}}$ and $|\mathcal{R}_{\mathsf{N}}| = 1$ for all N then we omit the algorithm Derive and drop the adjective universal.

Knowledge Soundness with Partial Opening. Finally we can consider a more general notion of knowledge-soundness for CP-SNARKs introduced in [BCFK19]. The intuition is to consider adversaries that explicitly return a valid opening for a subset of the commitments that they return. This models scenarios in which these commitments are not extractable and trusted by the verifier.

Definition 5.8 (Knowledge Soundness with Partial Opening). *We say that* Π *has knowledge soundness with partial opening for a commitment scheme* CS *and an auxiliary input distribution* $\mathcal{Z}$*, denoted* $\mathsf{poKSND}(\mathsf{CS}, \mathcal{Z})$ *for brevity, if for every (non-uniform) efficient adversary* $\mathcal{A}$ *there exists a (non-uniform) efficient extractor* $\mathcal{E}$ *such that* $\Pr\left[\mathsf{Game}^{\mathsf{poKSND}}_{\mathcal{Z},\mathcal{A},\mathcal{E}}(\lambda) = 1\right] = \mathsf{negl}(\lambda)$*, where the experiment is defined as follows.*

$$\mathsf{Game}^{\mathsf{poKSND}}_{\mathsf{CS},\mathcal{Z},\mathcal{A},\mathcal{E}} \to b$$

$$\mathsf{ck} \leftarrow \mathsf{CS.Setup}(1^\lambda); \quad (\mathsf{srs}, \mathsf{td_k}) \leftarrow \mathsf{KeyGen}(\mathsf{ck}, \mathsf{N}); \quad \mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathsf{srs})$$
$$\left(\mathsf{R}, \mathsf{x}, (c_j)_{j\in[\ell]}, (\mathsf{u}_j)_{j\in[\ell']}, (o_j)_{j\in[\ell']}, \pi\right) \leftarrow \mathcal{A}(\mathsf{srs}, \mathsf{aux}_Z)$$
$$\left((\mathsf{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega\right) \leftarrow \mathcal{E}(\mathsf{srs}, \mathsf{aux}_Z)$$
$$\mathsf{vk_R} \leftarrow \mathsf{Derive}(\mathsf{srs}, \mathsf{R})$$
$$b = \mathsf{Verify}(\mathsf{vk_R}, \mathsf{x}, \pi) \ \wedge \neg\Big(\bigwedge_{j\in[\ell]} \mathsf{VerCom}(\mathsf{ck}, c_j, \mathsf{u}_j, o_j) \stackrel{?}{=} \wedge \mathsf{R}(\mathsf{x}, (\mathsf{u}_j)_{j\in[\ell]}, \omega)\Big)$$

6 Our Compiler from PHPs to zkSNARKs with Universal SRS

In this section we show how to compile PHPs into zkSNARKs. At a high level, we follow the known paradigm stemming from Kilian's work in which the prover commits to the oracles, answers the verifier's queries and proves correctness of these answers [Kil92]. More specifically, our approach refines this paradigm for our case of interest (see the introduction for a high-level description).

We first introduce some required building blocks in Section 6.1 and then describe our compiler in two steps: in Section 6.2 we convert a PHP into a public coin interactive argument system in the structured reference string model (SRS)[13], and then remove interaction through the Fiat-Shamir transform. The proofs of the theorems in this section can be found in Appendix C.

6.1 Building Blocks

In our compiler we shall make use of the following:

- a PHP protocol PHP over a finite field $\mathbb{F}$;
- a commitment scheme CS for polynomials in $\mathbb{F}[X]$;
- a CP-SNARK $\mathsf{CP_{opn}}$ proving knowledge of the committed polynomials;
- a CP-SNARK $\mathsf{CP_{php}}$ proving that the PHP verifier accepts, namely for the family of relations $\mathcal{R}_{\mathsf{php}}$ defined in Section 3.1, which corresponds to the PHP verifier's degree and polynomial checks.

We now describe some of the properties we require from our commitment scheme for polynomials and from CP-SNARKs for them.

6.1.1 Commitments to Polynomials

Recall that a PHP verifier has access to two sets of oracle polynomials: those from the relation encoder (which roughly describe the relation) and those from the prover (which should supposedly persuade the verifier to accept a public input x). During compilation, we shall commit to polynomials in both sets; we will require all these commitments to be binding, but not to fully hide any of these polynomials.

[13] A straightforward extension of interactive arguments; see Section C.1 for a definition.

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).

Above, we ignored leakage when committing to relation encoding polynomials; we cannot do the same when committing to the polynomials from the PHP prover: 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 (see Definition 5.1) 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 6.1 (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 Definition 5.1, but where we allow Commit to be deterministic.*

We say that CS is somewhat-hiding for type type if it satisfies the following property.

type**-typed Somewhat Hiding.** *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: the distribution of the commitment key returned by $\mathcal{S}_{\mathsf{ck}}$ with a uniformly random $s \leftarrow\!\!\$\, \mathbb{F}$ as input is perfectly/statistically close to the one of the key returned by Setup; for any $f \in \mathbb{F}_{<d}[X]$, $(c, o) \approx (c', o')$ where $(c, o) \leftarrow \mathsf{Commit}(\mathsf{ck}, f)$, $(c', st) \leftarrow \mathsf{TdCom}(\mathsf{td}, f(s))$ and $o' \leftarrow \mathsf{TdOpen}(\mathsf{td}, st, c', f)$.*

For our first compiler (Section 6.2) we assume CS to be a type-based commitment scheme with type set $\mathcal{T} = \{\mathtt{rel}, \mathtt{swh}\}$ that is binding for all types and somewhat-hiding for type `swh`. We summarize this requirement in the following definition.

Definition 6.2 (Compiling Commitment Scheme). *Let $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCom})$ be a type-based commitment scheme for a class of polynomials $\mathbb{F}_{<d}[X]$ and a class of types $\mathcal{T} = \{\mathtt{rel}, \mathtt{swh}\}$. We say CS is a compiling commitment scheme if it is $\mathcal{T}$-binding and `swh`-somewhat-hiding.*

6.1.2 CP-SNARKs for CS

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 $\mathcal{R}' \supseteq \mathcal{R}_{\mathsf{php}}$, 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.

Additionally 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.

Leaky Zero-Knowledge. 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. This is formalized by letting the zero-knowledge simulator have access to a list $\{\mathsf{u}_i(y)\}_i$ as a hint for the simulation of proofs. The formal definition follows.

Definition 6.3. *A CP-SNARK CP is (b, C)-leaky zero-knowledge for a family of relations $\{\mathcal{R}_{\mathsf{N}}\}_{\mathsf{N} \in \mathbb{N}}$ if there exists a simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{leak}}, \mathcal{S}_{\mathsf{prv}})$ such that for all adversaries $\mathcal{A}$, for all $\mathsf{N} \in \mathbb{N}$, for all $\mathsf{R} \in \mathcal{R}_{\mathsf{N}}$, the following two properties hold.*

Proof Indistinguishability. *For all* $(\mathsf{ck}, \mathsf{td}) \in \mathcal{S}_{\mathsf{ck}}(1^\lambda)$*, for all* $\hat{\mathsf{x}}, \hat{\mathsf{w}}$ *where* $\hat{\mathsf{x}} = (\mathsf{x}', (c_j)_{j\in[\ell]})$ *and* $\hat{\mathsf{w}} = ((\mathsf{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega)$ *and such that* $\mathsf{R}^{\mathsf{Com}}_{\mathsf{ck},\mathsf{N}}(\hat{\mathsf{x}}, \hat{\mathsf{w}}) = 1$*, for any* $\mathcal{L} \leftarrow \mathcal{S}_{\mathsf{leak}}(1^\lambda, \mathsf{x})$ *let* $\mathit{Leak} := \{(j, \mathsf{u}_j(x))\}_{(j,x)\in\mathcal{L}}$*:*

$$\Pr\left[\begin{array}{r}(\mathsf{srs}, \mathsf{td_k}) \leftarrow \mathsf{KeyGen}(\mathsf{ck}, \mathsf{N})\\ \mathsf{srs_R} \leftarrow \mathsf{Derive}(\mathsf{srs}, \mathsf{R})\\ \pi \leftarrow \mathsf{Prove}(\mathsf{srs_R}, \hat{\mathsf{x}}, \hat{\mathsf{w}})\end{array} : \mathcal{A}(\mathsf{srs}, \mathsf{td_k}, \mathsf{td}, \mathsf{R}, \mathsf{x}, \mathsf{w}, \pi) = 1\right] \approx$$
$$\Pr\left[\begin{array}{r}(\mathsf{srs}, \mathsf{td_k}) \leftarrow \mathsf{KeyGen}(\mathsf{ck}, \mathsf{N})\\ \pi \leftarrow \mathcal{S}(\mathsf{td_k}, \mathsf{td}, \mathsf{R}, \hat{\mathsf{x}}, \mathit{Leak})\end{array} : \mathcal{A}(\mathsf{srs}, \mathsf{td_k}, \mathsf{td}, \mathsf{R}, \mathsf{x}, \mathsf{w}, \pi) = 1\right]$$

Bounded Leakage. *For any* x*, and any* $\mathcal{L} \leftarrow_{\$} \mathcal{S}_{\mathsf{leak}}(1^\lambda, \mathsf{x})$*, the list* $\mathcal{L}$ *is* (b, C)*-bounded with overwhelming probability over the security parameter.*

6.2 Compiling to Universal Interactive Arguments

We describe our compiled universal succinct interactive argument (UIA) system in the SRS model in Figure 3. A high-level description of UIA follows.

- At key-generation time we run the setup of the commitment scheme CS and generate keys for the auxiliary CP-SNARKs.
- When deriving a specialized SRS for a specific relation R we commit to all the polynomials returned by the relation encoder $\mathcal{RE}(\mathsf{R})$.
- The prover acts the same at every round except for the last. If we are not at the last round then it commits to the polynomials from the PHP prover $\mathcal{P}$, proves it knows their openings and propagates the rest of the messages from $\mathcal{P}$. At the last round it proves that the *PHP* verifier $\mathcal{V}$ would accept. In order to do that it first runs the decision stage of $\mathcal{V}$, thus obtaining a vector of degree checks $(d_j)_j$ and descriptions of polynomial equations $(G_j, \boldsymbol{v}_j)_j$. It then partially evaluates the polynomials G_j-s on the prover's message and uses them—together with the other checks—to prove $\mathcal{V}$ would accept.
- At every round that is not the last, the verifier simply propagates the messages from $\mathcal{V}$. At the last round, it obtains the checks from the decision stage of the PHP verifier. It then checks the prover's final PHP proof as well as all the opening proofs received throughout the interaction.

Theorem 6.1. *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 compiling commitment scheme (Definition 6.2) equipped with CP-SNARKs* $\mathsf{CP_{opn}}$ *for* $\mathcal{R}_{\mathsf{opn}}$ *and* $\mathsf{CP_{php}}$ *for* $\mathcal{R}_{\mathsf{php}}$*.*

- *The scheme* $\mathsf{UIA} = (\mathsf{KeyGen}, \mathsf{Derive}, \mathbb{P}, \mathbb{V})$ *defined in Figure 3 is a universal succinct interactive argument in the SRS model for* $\mathcal{R}$*.*
- *If* $\mathsf{CP_{opn}}$ *is TP-ZK, and, for a checker* C*,* PHP *(resp.* $\mathsf{CP_{php}}$*) is* $(\mathsf{b} + \mathbf{1}, \mathsf{C})$*-bounded honest-verifier zero-knowledge (resp.* (b, C)*-leaky zero-knowledge) then* UIA *is trapdoor-commitment honest-verifier zero-knowledge.*

Remark 7. *Under the hypothesis of Theorem 6.1 above, it is sufficient for* $\mathsf{CP_{php}}$ *to be* T*-restricted complete, with* $T = \left((\mathtt{rel})^{\mathsf{n}(0)} \| (\mathtt{swh})^{\mathsf{n_p}}\right) \in \mathcal{T}^{\mathsf{n}^*}$*, in order to obtain the completeness of* UIA*.*

Remark 8 (On updatable SRS)**.** *If the commitment key generated by* Setup *is updatable [GKM+18], and* $\mathsf{CP_{opn}}$ *and* $\mathsf{CP_{php}}$ *have commitment-only SRS (see Definition 5.6) then the SRS of* UIA *is updatable.*

Remark 9 (Efficiency of the resulting UIA)**.** *From the construction one can see that in* UIA*:*

- *prover and verifier interact for* $\mathsf{r} + 1$ *rounds;*
- Derive *outputs a specialized verification key that consists of* $\mathsf{n}(0)$ *commitments;*
- *the prover sends:* m^* *field elements,* $\mathsf{n_p}$ *commitments,* r *proofs of* $\mathsf{CP_{opn}}$*, and one proof of* $\mathsf{CP_{php}}$*;*
- *the verifier's running time is that of the PHP verifier, plus the sum of running* $\mathsf{Verify_{opn}}$ *and* $\mathsf{Verify_{php}}$*.*

Combining the above observations with the succinctness of the commitment scheme CS *(Definition 5.2) and of the CP-SNARKs* $\mathsf{CP_{opn}}$ *and* $\mathsf{CP_{php}}$*, we obtain the succinctness of* UIA*.*

$\mathsf{KeyGen}(1^\lambda, \mathsf{N}) \rightarrow (\mathsf{ek}, \mathsf{vk})$

// Let D be the max degree of PHP (Definition 3.1)

$\mathsf{ck} \leftarrow \mathsf{CS.Setup}(1^\lambda, \mathsf{D})$

$(\mathsf{ek}_{\mathsf{opn}}, \mathsf{vk}_{\mathsf{opn}}) \leftarrow \mathsf{KeyGen}_{\mathsf{opn}}(\mathsf{ck})$

$(\mathsf{ek}_{\mathsf{php}}, \mathsf{vk}_{\mathsf{php}}) \leftarrow \mathsf{KeyGen}_{\mathsf{php}}(\mathsf{ck})$

$\mathsf{ek} := (\mathsf{ck}, \mathsf{ek}_{\mathsf{php}}, \mathsf{ek}_{\mathsf{opn}}); \mathsf{vk} := (\mathsf{vk}_{\mathsf{php}}, \mathsf{vk}_{\mathsf{opn}})$

$\mathsf{Derive}(\mathsf{vk}, \mathsf{R}) \rightarrow (\mathsf{ek}_\mathsf{R}, \mathsf{vk}_\mathsf{R})$

$\boldsymbol{p_0} \leftarrow \mathcal{RE}(\mathbb{F}, \mathsf{R})$

$(\boldsymbol{c_0}, \boldsymbol{o_0}) \leftarrow\!\!\$\ \mathsf{Commit}(\mathsf{ck}, \mathtt{rel}, \boldsymbol{p}_0)$

$\mathsf{vk}_\mathsf{R} := \big(\mathsf{vk}, \boldsymbol{c}_0\big)$

$\mathsf{ek}_\mathsf{R} := \big(\mathsf{ek}, \mathsf{R}, \boldsymbol{p}_0, \boldsymbol{o}_0\big)$

$\mathbb{P}(\mathsf{ek}_\mathsf{R}, \mathsf{x}, \mathsf{w}, \bar{\rho}_1, \ldots, \bar{\rho}_i) \rightarrow \bar{\pi}_i$

Let $\mathsf{r} := \mathsf{r}(|\mathsf{R}|)$

if $i \leq \mathsf{r}(|\mathsf{R}|)$ **then**

// Get polynomials and messages from PHP prover

$\big(\boldsymbol{p}_i, \boldsymbol{\pi}_i\big) \leftarrow \mathcal{P}(\mathbb{F}, \mathsf{R}, \mathsf{x}, \mathsf{w}, \bar{\rho}_1, \ldots, \bar{\rho}_i)$

$(\boldsymbol{c}_i, \boldsymbol{o}_i) \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathtt{swh}, \boldsymbol{p}_i)$

$\boldsymbol{\pi}_{\mathsf{opn},i} \leftarrow \mathsf{Prove}_{\mathsf{opn}}(\mathsf{ek}_{\mathsf{opn}}, \boldsymbol{c}_i, \boldsymbol{o}_i)$

$\bar{\pi}_i := (\boldsymbol{c_i}, \pi_i, \boldsymbol{\pi}_{\mathsf{opn},i})$

else

// Get checks from PHP verifier

$((d_j)_{j\in[\mathsf{n_p}]}, (G_j, \boldsymbol{v}_j)_{j\in[\mathsf{n_e}]}) \leftarrow D_{\mathcal{V}}(\mathbb{F}, \mathsf{x}, \rho_1, \ldots, \rho_{\mathsf{r}+1})$

$\hat{\mathsf{x}}_{\mathsf{php}} := \big((d_j)_{j\in[\mathsf{n_p}]}, (G'_j, \boldsymbol{v}_j)_{j\in[\mathsf{n_e}]}\big)$,

where G'_k partially evaluates G_k, i.e., for $k \in [\mathsf{n_e}]$:

$G'_k(X, (X_j)_{j\in[\mathsf{n}^*]}) := G_k(X, (X_j)_{j\in[\mathsf{n}^*]}, (\boldsymbol{\pi}_1||\ldots||\boldsymbol{\pi}_\mathsf{r}))$

$\hat{\mathsf{w}}_{\mathsf{php}} := ((\boldsymbol{p}_0||\ldots||\boldsymbol{p}_\mathsf{r}), (\boldsymbol{o}_0||\ldots||\boldsymbol{o}_\mathsf{r}))$

$\boldsymbol{\pi}_{\mathsf{php}} \leftarrow \mathsf{Prove}_{\mathsf{php}}(\mathsf{ek}_{\mathsf{php}}, \hat{\mathsf{x}}_{\mathsf{php}}, (\boldsymbol{c}_0||\ldots||\boldsymbol{c}_\mathsf{r}), \hat{\mathsf{w}}_{\mathsf{php}})$

$\bar{\pi}_{\mathsf{r}+1} := \boldsymbol{\pi}_{\mathsf{php}}$

$\mathbb{V}(\mathsf{srs}, \mathsf{vk}, \hat{\mathsf{x}}, \bar{\pi}_1, \ldots, \bar{\pi}_i) \rightarrow \bar{\rho}_i$

if $i \leq \mathsf{r}$ **then**

$\rho_i \leftarrow \mathcal{V}(\mathbb{F}, \mathsf{x}, \pi_1, \ldots, \pi_{i-1})$

$\bar{\rho}_i := \rho_i$

else

$\hat{\mathsf{x}}_{\mathsf{php}} := \big((d_j)_{j\in[\mathsf{n_p}]}, (G'_j, \boldsymbol{v}_j)_{j\in[\mathsf{n_e}]}\big)$,

where G'_k partially evaluates G_k, i.e., for $k \in [\mathsf{n_e}]$:

$G'_k(X, (X_j)_{j\in[\mathsf{n}^*]}) := G_k(X, (X_j)_{j\in[\mathsf{n}^*]}, (\boldsymbol{\pi}_1||\ldots||\boldsymbol{\pi}_\mathsf{r}))$

$b \leftarrow \mathsf{Verify}_{\mathsf{php}}(\mathsf{vk}_{\mathsf{php}}, \hat{\mathsf{x}}_{\mathsf{php}}, (\boldsymbol{c}_0||\ldots||\boldsymbol{c}_\mathsf{r}), \boldsymbol{\pi}_{\mathsf{php}})$

$b^i \leftarrow \mathsf{Verify}_{\mathsf{opn}}(\mathsf{vk}_{\mathsf{opn}}, (c_{i,j})_{j\in\mathsf{n}(i)}, \boldsymbol{\pi}_i)$ for $i \in [\mathsf{r}]$

Accept iff $\big(\wedge_{i\in[\mathsf{r}]} b^i \wedge b\big)$

Above we use a shortcut notation for committing to whole vectors of polynomials in one go. That is, given a commitment type t and a vector of polynomials $\boldsymbol{p}$ of size m, above we write $(\boldsymbol{c}, \boldsymbol{o}) \leftarrow \mathsf{Commit}(\mathsf{ck}, t, \boldsymbol{p})$ to mean that for each $j \in [m]$ $(c_j, o_j) \leftarrow \mathsf{Commit}(\mathsf{ck}, t, p_j)$, $\boldsymbol{c} = (c_1, \ldots, c_m)$ and $\boldsymbol{o} = (o_1, \ldots, o_m)$.

Figure 3: Compiler from PHP to UIA.

Intuition on Security Proof We refer the reader to Appendix C for a formal proof of Theorem 6.1; here we provide an intuition. By the knowledge-soundness property of PHPs we know we can extract a valid witness from the interaction with a PHP prover. Let's call this extractor $\mathcal{E}_{\text{PHP}}$ and let us assume the verifier accepts. The high-level idea is to simulate the interaction between $\mathcal{E}_{\text{PHP}}$ and a PHP prover as follows: whenever $\mathcal{E}_{\text{PHP}}$ queries a polynomial we run the extractor for $\mathsf{CP}_{\mathsf{opn}}$ and respond with the corresponding polynomial (we are ignoring messages in this proof intuition). Call $\tilde{\mathsf{w}}$ the output of $\mathcal{E}_{\text{PHP}}$ at the end of the interaction. If this is not a valid witness with high probability then we broke the assumption on knowledge-soundness of the PHP. To see why: consider the extractor for $\mathsf{CP}_{\mathsf{php}}$, if we run it on π_{php} then we can obtain polynomials that make the PHP verifier accept (if given oracle access to them). These polynomials must be identical to the ones we can extract through $\mathsf{CP}_{\mathsf{opn}}$, otherwise we could break binding. If $\tilde{\mathsf{w}}$ were not a valid witness then we could construct a PHP prover that makes the verifier accept but without being able to extract a valid witness from it breaking knowledge-soundness of the PHP.

We now provide an intuition about zero-knowledge; for simplicity we shall 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 6.1), 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 compuler—b-leaky ZK— and this allows us to still simulate an interaction with an honest prover that is indistinguishable after further b leaked evaluations[14].

7 CP-SNARKs for Pairing-Based Polynomial Commitments

In this section we present constructions of (type-based) commitment schemes for polynomials that work in bilinear groups, and a collection of CP-SNARKs for various relations over such committed 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 [Gro10] and Kate et al. [KZG10].
This section includes two commitment schemes CS_1 and CS_2 (Section 7.2) as well as CP-SNARKs working over them; more details on the CP-SNARKs follow. Our CP-SNARKs work over both commitment schemes unless explicitly stated otherwise.

- *"I know p : c opens to p"*: two CP-SNARKs $\mathsf{CP}_{\mathsf{opn}}$ for proof of knowledge of opening, secure respectively in the algebraic group model and under the mPKE assumption (Section 7.3);
- "$p(x) = y$": a CP-SNARK for polynomial evaluation, $\mathsf{CP}_{\mathsf{eval},1}$, secure under the d-SDH assumption (Section 7.4). We then extend this CP-SNARK as $\mathsf{CP}_{\mathsf{eval}}$ to support batching—"$\left(p_i(x_i) = y_i\right)_{i\in[\ell]}$"— in Section 7.5;
- a very general construction for a CP-SNARK for polynomial equations[15], $\mathsf{CP}_{\mathsf{eq}}$, relying mainly on $\mathsf{CP}_{\mathsf{opn}}$ and $\mathsf{CP}_{\mathsf{eval}}$ (Section 7.6);
- a CP-SNARK, $\mathsf{CP}_{\mathsf{qeq}}$, for quadratic polynomial equations[16] specific to commitment scheme CS_2 (Section 7.7); 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.
- "$\mathsf{deg}(p) \le d$": two CP-SNARKs for degree bounds, $\mathsf{CP}^{(\star)}_{\mathsf{deg}}$ and $\mathsf{CP}^{(2)}_{\mathsf{deg}}$, both secure if $\mathsf{CP}_{\mathsf{opn}}$ and $\mathsf{CP}_{\mathsf{eq}}$ are secure; while $\mathsf{CP}^{(\star)}_{\mathsf{deg}}$ works over both commitment schemes, $\mathsf{CP}^{(2)}_{\mathsf{deg}}$ works only over CS_2;
- a CP-SNARK $\mathsf{CP}_{\mathsf{link}}$, a key ingredient in our compiler to universal CP-SNARKs, to link polynomial commitments of different types; see "Additional building blocks" 8.1 for further motivation.

[14] We observe that, for polynomials that allow for unbounded ZK, we can even use leakier forms of commitments than somewhat-hiding ones.
[15] An example of polynomial equations is $a(X)b(X) - 2c(X)d(X)e(X) = 0$).
[16] Here "quadratic" means it supports products of at most two polynomials.

7.1 Bilinear Groups and Assumptions

A *bilinear group generator* $\mathsf{GenG}(1^\lambda)$ outputs $\mathsf{bgp} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, 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 Type-3 groups where it is assumed there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. We use the bracket notation of [EHK+13], 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_s is a fixed generator of $\mathbb{G}_g$. From an element $[a]_g \in \mathbb{G}_g$ and a scalar b it is possible to efficiently compute $[ab] \in \mathbb{G}_g$. Also, given elements $[a]_1 \in \mathbb{G}_1$ and $[b]_2 \in \mathbb{G}_2$, one can efficiently compute $[a \cdot b]_T$ by using the pairing $e([a]_1, [b]_2)$, that we compactly denote with $[a]_1 \cdot [b]_2$.

In our constructions we make use of the following assumptions over a group generator GenG.

Assumption 1 (d-Power Discrete Logarithm [Lip12])**.** *Given a degree bound $d \in \mathbb{N}$, the d-Power Discrete Logarithm (d-DLOG) assumption holds for a bilinear group generator* GenG *if for every efficient non-uniform adversary $\mathcal{A}$ the following probability is negligible in λ:*

$$\Pr\left[s' = s : \begin{array}{c} \mathsf{bgp} \leftarrow_\$ \mathsf{GenG}(1^\lambda);\ s \leftarrow_\$ \mathbb{Z}_q; \\ s' \leftarrow \mathcal{A}(\mathsf{bgp}, \{[s^j]_1, [s^j]_2\}_{j \in [0,d]}) \end{array} \right].$$

Assumption 2 (d-Strong Diffie-Hellman [BB04])**.** *Given a degree bound $d \in \mathbb{N}$, the d-Strong Diffie-Hellman (d-SDH) assumption holds for a bilinear group generator* GenG *if for every efficient non-uniform adversary $\mathcal{A}$ the following probability is negligible in λ:*

$$\Pr\left[C = [(s+c)^{-1}]_1 : \begin{array}{c} \mathsf{bgp} \leftarrow_\$ \mathsf{GenG}(1^\lambda);\ s \leftarrow_\$ \mathbb{Z}_q; \\ (c, C) \leftarrow \mathcal{A}(\mathsf{bgp}, \{[s^j]_1, [s^j]_2\}_{j \in [0,d]}) \end{array} \right].$$

We consider a slight variant of the Power Knowledge of the Exponent (PKE) Assumption of Groth [Gro10]. This variant, also used in [CHM+20], considers an adversary (resp. an extractor) that outputs a vector of group elements (resp. of tuples of field elements), and is implied by the PKE assumption.

Assumption 3 (mPKE)**.** *The (multi-instance) Power Knowledge of Exponent (mPKE) assumption holds for a bilinear group generator* GenG *if for every efficient non-uniform adversary $\mathcal{A}$ and a degree bound $d \in \mathbb{N}$ there exists an efficient extractor $\mathcal{E}$ such that for any benign distribution $\mathcal{Z}$ the following probability is negligible in λ:*

$$\Pr\left[\exists j : \begin{array}{c} d_j = \gamma \cdot c_j \wedge \\ c_j \neq \sum_k a_k^{(j)} s^j \end{array} : \begin{array}{c} \mathsf{bgp} \leftarrow_\$ \mathsf{GenG}(1^\lambda); \\ \mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathsf{bgp}); \\ s, \gamma \leftarrow_\$ \mathbb{Z}_q; \\ \Sigma = \big(\ [s^j]_1, [s^j]_2, [\gamma s^j]_1, [\gamma]_2\ \big)_{j \in [0,d]}; \\ (c_j)_{j \in [\ell']}, (d_j)_{j \in [\ell']} \leftarrow \mathcal{A}(\mathsf{bgp}, \Sigma, \mathsf{aux}_Z); \\ (\boldsymbol{a}^{(j)})_{j \in [\ell']} \leftarrow \mathcal{E}(\mathsf{bgp}, \Sigma, \mathsf{aux}_Z) \end{array} \right].$$

7.2 The Commitment Schemes

We show two type-based commitment schemes, CS_1 and CS_2, with type set $\{\mathtt{rel}, \mathtt{swh}\}$ and for degree-d polynomials. We begin with an informal explanation of them.

In both schemes, ck contains encodings of powers of a secret point s, a commitment of type $\mathtt{swh}$ to a polynomial $p(X)$ is a group element $[p(s)]_1$. The only difference between the two schemes are the commitments of type $\mathtt{rel}$, which in CS_1 are $[p(s)]_1$ whereas in CS_2 are $[p(s)]_2$. As we shall see in Section 7.7, the advantage of having some polynomials committed in $\mathbb{G}_2$ is that one immediately gets a way to test quadratic equations over polynomials where each quadratic term involves exactly one polynomial of type $\mathtt{rel}$. Both types of commitments are computationally binding under the power-discrete logarithm assumption [Lip12]. For commitments of type $\mathtt{swh}$ we show somewhat hiding.

Below we describe the commitment schemes in more detail. To keep the presentation compact, we describe them as a single scheme CS_g parametrized by the following function, for $g \in \{1, 2\}$,

$$\mu_g(\mathsf{type}) = \begin{cases} g & \text{if } \mathsf{type} = \mathtt{rel} \\ 1 & \text{if } \mathsf{type} = \mathtt{swh} \end{cases}$$

The function essentially dictates in which group is a type-rel commitment.
The algorithms ($\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCom}$) of CS_g are defined as follows:

$\mathsf{CS}_g.\mathsf{Setup}(1^\lambda, d)$**:** run $\mathsf{bgp} \leftarrow\$ \mathsf{GenG}(1^\lambda)$ to generate the bilinear groups description, set the message space to be $\mathbb{F}_{\leq d}[X]$ where $\mathbb{F} := \mathbb{Z}_q$. Next, sample $s \leftarrow\$ \mathbb{Z}_q$ uniformly at random, compute and output:

$$\mathsf{ck} = \begin{cases} \left(([s^j]_1)_{j\in[0,d]}, [s]_2\right) & \text{if } g = 1, \\ ([s^j]_1, [s^j]_2)_{j\in[0,d]} & \text{if } g = 2. \end{cases}$$

$\mathsf{CS}_g.\mathsf{Commit}(\mathsf{ck}, \mathsf{type}, p) \to (c, o)$**:** Let $\hat{g} \leftarrow \mu_g(\mathsf{type})$, and output the commitment $c := [p(s)]_{\hat{g}}$ (the opening o is empty).[17]

$\mathsf{CS}_g.\mathsf{VerCom}(\mathsf{ck}, \mathsf{type}, c, p, o)$**:** set $\hat{g} \leftarrow \mu_g(\mathsf{type})$, and check if $c \stackrel{?}{=} p([s]_{\hat{g}})$.

Remark 10. *We note that in* CS_1*, the elements* $[1, s]_2$ *are not needed to commit and verify openings, but they are useful to verify the correctness of* ck *(which is useful when generating* ck *in an updatable way).*

In the following theorem we state the security of the scheme.

Theorem 7.1. CS_g *is binding under the d-DLOG assumption for* GenG*, and perfectly somewhat-hiding.*

Proof. Binding is essentially the same as in [Gro10]. Assume the adversary produces two polynomials p and p' that evaluate to the same value on the point s. Then by finding the 0's of the polynomial $p(X) - p'(X)$ we can find s and break the d-DLOG assumption.

For somewhat-hiding, we notice that the polynomial commitment scheme does not need any trapdoor opening information, thus the TdCom algorithm we define next sets st to be the empty string and there is no need for the TdOpen algorithm. We define algorithms $\mathcal{S}_{\mathsf{ck}}$ and TdCom and shows that the distributions produced by the algorithms are indistinguishable from the distributions produced by Setup and Commit:

$\mathcal{S}_{\mathsf{ck}}(s) \to (\mathsf{ck}, \mathsf{td})$**:** use s to compute ck as in Setup and output ck and $\mathsf{td} = s$.

$\mathsf{TdCom}(\mathsf{ck}, \mathsf{type}, p(s)) \to (c, st)$**:** let $\mu_g(\mathsf{type}) = g$ and output $[p(s)]_g$.

Clearly for an uniformly random s the distributions of the outputs of $\mathcal{S}_{\mathsf{ck}}$ and Setup are identical. □

7.3 CP-SNARKs for $\mathcal{R}_{\mathsf{opn}}$

Here we present two CP-SNARKs for the commitment schemes $\mathsf{CS}_1, \mathsf{CS}_2$ and the relation $\mathcal{R}_{\mathsf{opn}}$ (which essentially provides a proof of knowledge of the committed polynomials). For our results, we are interested in proving this relation only over commitments of type swh.

A CP-SNARK in the algebraic group model. The first CP-SNARK, $\mathsf{CP}^{\mathsf{AGM}}_{\mathsf{opn}}$, is actually a trivial scheme in which the proof is the empty string. Its knowledge-soundness, can be shown in the algebraic group model [FKL18] where any adversary that returns a commitment is assumed to know coefficients which explain it as a linear combination of the public parameters, the ck. This is an observation already done in previous work, e.g., [GWC19, CHM+20]), and thus we omit the details of the analysis.

Theorem 7.2. $\mathsf{CP}^{\mathsf{AGM}}_{\mathsf{opn}}$ *is a CP-SNARK for* $\mathcal{R}_{\mathsf{opn}}$ *over* CS_1 *(resp.* CS_2*) that is* $\mathtt{swh}^\ell$*-restricted complete, perfectly zero-knowledge and knowledge-sound in the algebraic group model.*

[17] For this reason, all the CP-SNARKs given for this commitment scheme will omit o from the prover's inputs.

A CP-SNARK under the mPKE assumption. The second CP-SNARK, $\mathsf{CP}^{\mathsf{PKE}}_{\mathsf{opn}}$, 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 π_{opn} such that $\pi_{\mathsf{opn}} = \gamma \cdot c$ for some secret $\gamma \in \mathbb{F}$, and this π_{opn} can be honestly computed by using the same linear combination used to compute 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.

$\mathsf{CP}^{\mathsf{PKE}}_{\mathsf{opn}}.\mathsf{KeyGen}(\mathsf{ck})$: parse ck as $(\mathsf{ck}_1, \mathsf{ck}_2)$ with $\mathsf{ck}_1 \in \mathbb{G}_1^{d+1}$, sample $\gamma \leftarrow\!\!\$\, \mathbb{F}$, define $\mathsf{ek} := (\mathsf{ck}, \gamma \cdot \mathsf{ck}_1)$ and $\mathsf{vk} := [1, \gamma]_2$, and return $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk})$.

$\mathsf{CP}^{\mathsf{PKE}}_{\mathsf{opn}}.\mathsf{Prove}(\mathsf{ek}, (c_j)_{j\in[\ell]}, (p_j)_{j\in[\ell]})$: for $j \in [\ell]$ compute $\pi_j \leftarrow [\gamma \cdot p_j(s)]_1$, next compute $(\rho_1, \ldots, \rho_\ell) \leftarrow \mathcal{H}((c_j)_{j\in[\ell]})$ and output $\pi_{\mathsf{opn}} := \sum_j \rho_j \pi_j$.

$\mathsf{CP}^{\mathsf{PKE}}_{\mathsf{opn}}.\mathsf{Verify}(\mathsf{vk}, (c_j)_{j\in[\ell]}, \pi_{\mathsf{opn}})$: compute $(\rho_1, \ldots, \rho_\ell) \leftarrow \mathcal{H}((c_j)_{j\in[\ell]})$ and $c := \sum_j \rho_j c_j$. Output 1 if and only if $e(c, [\gamma]_2) = e(\pi_{\mathsf{opn}}, [1]_2)$.

Remark 11 (On Updatable SRS generation)**.** *Note that the SRS of this CP-SNARK can be generated by having access to the commitment key (without need of knowing its trapdoor), and it is easy to see how it can be generated in an updatable fashion, and the correctness of every element can be efficiently checked using a pairing. Generating the SRS of* $\mathsf{CP}_{\mathsf{opn}}$ after *the commitment key* ck *would however require an additional sequence of rounds in the SRS ceremony. Although this can be still useful when re-using an existing commitment key, it is annoying if the goal is to generate* ck *and the* $\mathsf{CP}_{\mathsf{opn}}$ *SRS together. In the latter case, however, it is easy to see that they can be generated together with a single sequence of rounds in the ceremony, i.e., such that at every round the i-th participant outputs its version of* $(\mathsf{ck}, \gamma \cdot \mathsf{ck}_1)$.

Efficiency. Key generation requires $d+1$ exponentiations in $\mathbb{G}_1$ to generate $\gamma \cdot \mathsf{ck}_1$, and one in $\mathbb{G}_2$ to compute $[\gamma]_2$. The prover can be implemented so as to require d^* $\mathbb{G}_1$-exponentiations and $O(\ell \cdot d^*)$ $\mathbb{F}$-operations, where $d^* \max_{j\in[\ell]}\{\mathsf{deg}(p_j)\}$. This is done by computing $p^*(X) \leftarrow \sum_j \rho_j p_j(X)$ and then $\pi_{\mathsf{opn}} \leftarrow [\gamma \cdot p^*(s)]_1$. Verification requires: 2 pairings, ℓ $\mathbb{G}_1$-exponentiations, and one hash computation.

Security. In the following theorem we state the security of $\mathsf{CP}^{\mathsf{PKE}}_{\mathsf{opn}}$.

Theorem 7.3. $\mathsf{CP}^{\mathsf{PKE}}_{\mathsf{opn}}$ *is a CP-SNARK for* $\mathcal{R}_{\mathsf{opn}}$ *over* CS_1 *(resp.* CS_2*) that is* $\mathtt{swh}^\ell$*-restricted complete, perfectly zero-knowledge and knowledge-sound under the mPKE assumption in the random oracle model.*

Proof. Completeness is obvious. Zero-knowledge is also rather easy to see: a simulator that knows γ can perfectly simulate proofs without knowing the witness. Before proving knowledge soundness we recall an useful form of the Chernoff-Hoeffding bound [DP09].

Lemma 7.4. *Let* $X := \sum_{j\in[n]} X_i$ *where* $X_1, \ldots, X_n$ *are independently distributed in* $[0, 1]$*. Then for all* $t > 0$*:*

$$\Pr[X < E[X] - t] \leq 2^{-2t^2/n}$$

Let $\mathcal{A}$ be an (non-uniform PT) adversary and $\mathcal{Z}$ be an auxiliary input distribution such that for any $\mathcal{E}$ the probability that $\mathcal{A}$ outputs a statement $(c_j)_{j\in[\ell]}$ and a valid proof π is ϵ in the game $\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}}$ (where $\mathcal{RG}$ is the dummy algorithm that outputs $\mathcal{R}_{\mathsf{opn}}$). Moreover, let W be the event that the adversary outputs a valid statement-proof tuple. (Obviously, $\Pr[W] = \epsilon$.)

Consider the following adversary $\mathcal{B}$ and auxiliary distribution $\mathcal{Z}'$ against the mPKE assumption. The distribution $\mathcal{Z}'(\Sigma)$ computes the structured reference string srs of $\mathsf{CP}_{\mathsf{opn}}$ from Σ, runs $\mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathcal{R}_{\mathsf{opn}}, \mathsf{srs})$ and outputs $\mathsf{srs}, \mathsf{aux}_Z$.

<u>Adversary $\mathcal{B}_{i,\boldsymbol{h}}(\Sigma, (\mathsf{srs}, \mathsf{aux}_Z); \rho)$:</u>

1. Let $K = 2\ell\epsilon^{-1}q(1+\lambda)$, parse $\rho = (h_k^{(j)})_{i<k\leq q, j\in[K]}$ where $h_k^{(j)} \in \mathbb{Z}_q^\ell$ and q is the maximum amount of random oracle queries made by an execution of $\mathcal{A}$.

2. Compute ck from Σ, run $\mathcal{A}(\mathcal{R}_{\mathsf{opn}}, \mathsf{ck}, \mathsf{srs}, \mathsf{aux}_Z)$ and answer the first $i-1$ queries of $\mathcal{A}$ to the random oracle with the values $\boldsymbol{h} = h_1, \dots, h_{i-1}$. Let st the state of $\mathcal{A}$ just before the i-th queried is sent.
3. For $j = 1 \dots K$ run the following:
   (a) Run $\mathcal{A}$ feeding it with the value $h_k^{(j)}$ at the k-th query.
   Let $\hat{\mathsf{x}}_j, \pi_j$ be the output of $\mathcal{A}$ and let $b_j \leftarrow \mathsf{Verify}(\mathsf{srs}, \hat{\mathsf{x}}_j, \pi_j)$.
   (b) Rewind $\mathcal{A}$ to the state st.
4. Assert $\sum_j b_j \geq \ell$, let H be a subset of of cardinality ℓ of the indexes j such that $b_j = 1$, we define the square matrix M which columns are the vectors $h_i^{(j)} \in \mathbb{Z}_q^\ell$ and $j \in H$.
5. Assert that M is full rank.
6. Assert that for all j, j' we have $\hat{\mathsf{x}}_j = \hat{\mathsf{x}}_{j'}$. If so parse them as $(c_j)_{j \in [\ell]}$.
7. Compute $(d_j)_{j \in [\ell]} = (\pi_j)_{j \in H} M^{-1}$ and output $(c_j)_{j \in [\ell]}, (d_j)_{j \in [\ell]}$.

The adversary $\mathcal{B}$ is parameterized by an index i and values $h_1, \dots, h_{1-1}$ where $h_j \in \mathbb{Z}_q^\ell$.

First we notice that if the adversary $\mathcal{B}$ does not abort then it outputs values $(c_j)_{j \in [\ell]}$ and $(d_j)_{j \in [\ell]}$ such that for all $j \in \ell : \gamma \cdot c_j = d_j$. Indeed the verification in step 3a, for any j, we set b_j to 1 if and only if $\gamma \cdot \sum_k h_{j,k} c_k = \pi_j$ where we parse $h_i^{(j)} = x_{j,1}, \dots, \rho_{j,\ell}$, thus $(\pi_j)_{j \in H} = (c_j)_{j \in [\ell]} \cdot M$.

We analyze the probability that $\mathcal{B}$ does not abort. Let Q_j be the event that the adversary $\mathcal{A}$ queries the random oracle with $(c_j)_{j \in [\ell]}$ (the output instance) at the j-th random oracle query. Let i be the index that maximizes the probability $\Pr[W \wedge Q_i]$. It is easy to see that $\Pr[W \wedge Q_i] \geq \frac{\epsilon}{q}$. Let $\boldsymbol{h}$ be the assignment of the first $i-1$ queries that maximize the probability $\Pr[W \wedge Q_i]$, by average argument, we notice that there must exist $\boldsymbol{h}$ such that, conditioned on the assignment $\Pr[W \wedge Q_i | \boldsymbol{h}] \geq \frac{\epsilon}{q}$.

Given an assignment aux_Z, we call it *good* if $\Pr[W \wedge Q_i | \boldsymbol{h}, \mathsf{aux}_Z] \geq \frac{\epsilon}{2q}$. By a simple average argument we have that with probability $\frac{1}{2}$ an output aux_Z of $\mathcal{Z}$ is good. Also we notice that if we fix $\boldsymbol{h}$ and aux_Z then the random variables $b_1, \dots, b_K$ are independent and if aux_Z is good then each of them has average greater or equal to $\frac{\epsilon}{2q}$, thus by the Chernoff-Hoeffding bound we have that:

$$\Pr[\sum_j b_j \geq \ell | \boldsymbol{h}, \mathsf{aux}_Z] \geq 1 - \mathsf{negl}(\lambda).$$

Thus the assertion in step 4 passes with overwhelming probability. We notice that the assertion in step 5 passes with overwhelming probability as the rows of M are random vectors in $\mathbb{Z}_q^\ell$, also the assertion in step 6 passes always cause fixing $\boldsymbol{h}$ and aux_Z the i-th query of the adversary $\mathcal{A}$ is deterministic function of srs. Putting all together the probability that $\mathcal{B}$ does not abort is greater than $\frac{1}{2} - \mathsf{negl}(\lambda)$.

We are ready to define the extractor for the knowledge soundness experiment. Roughly speaking the extractor calls the extractor of $\mathcal{B}$, however the reduction $\mathcal{B}$ is a *probabilistic* polynomial time algorithm. Thanks to the non-uniformity we can fix the randomness of $\mathcal{B}$ to a string ρ that maximizes the probability of $\mathcal{B}$ outputting valid tuples. Thus let $\mathcal{B}'$ such non-uniform PT that runs $\mathcal{B}$ with randomness set to ρ.

Let $\mathcal{E}$ be the extractor of $\mathcal{B}'$, assumed to exist thanks to the mKEA assumption. The extractor outputs vectors $\boldsymbol{a}^{(j)}$ for any $j \in [\ell]$. We let the extractor for $\mathcal{A}$ simply run $\mathcal{E}$ and output what it does. By the mPKE assumption, we have $c_j = \sum_k a_k^{(j)} s^k$ (as otherwise $\mathcal{B}$ would break the mPKE assumption). □

Remark 12. *(Efficiently composing* $\mathsf{CP}_{\mathsf{opn}}$ *with other SNARKs) All of the CP-SNARKs in this section apply* $\mathsf{CP}_{\mathsf{opn}}$ *to obtain extractability of the committed polynomials. More precisely, this is true only for polynomials of type* `swh`*; we assume the adversary always opens commitments of type* `rel`*. The proofs of the CP-SNARKs we present in this section are all of the form* $(\pi_{\mathsf{opn}}, \pi)$ *where the first part,* π_{opn}*, is a proof of knowledge of a valid opening for the commitments in input. A straightforward composition of these CP-SNARKs would incur in redundantly proving the knowledge of the openings of the same commitments; therefore, we do not use black-box composition: given a CP-SNARK* $\mathsf{CP} = (\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ *we define the algorithms* $\overline{\mathsf{Prove}}$ *and* $\overline{\mathsf{Verify}}$ *respectively working just as* Prove *and* Verify*, except that they do not compute/verify the proof* π_{opn}*.*

7.4 CP-SNARK for evaluation of a single polynomial

We define a CP-SNARK $\mathsf{CP}_{\mathsf{eval},1}$ for the relation $\mathsf{R}_{\mathsf{eval},1}((a,b),p) := p(a) \stackrel{?}{=} b$, where p is committed as $[p(s)]_1$. Hence $\mathsf{CP}_{\mathsf{eval},1}$ is complete for CS_1, and $\mathtt{swh}$-restricted[18] complete for CS_2. This scheme is essentially the evaluation proof technique of [KZG10] with an additional proof of knowledge.

$\mathsf{KeyGen}_{\mathsf{eval1}}(\mathsf{ck})$**:** execute $(\mathsf{ek}_{\mathsf{opn}}, \mathsf{vk}_{\mathsf{opn}}) \leftarrow \mathsf{KeyGen}_{\mathsf{opn}}(\mathsf{ck})$, parse ck as $([s^j]_1,[s^j]_2)_{j\in[0,d]}$ define $\mathsf{ek} := (\mathsf{ck}, \mathsf{ek}_{\mathsf{opn}})$ and $\mathsf{vk} := ([1,s]_2, \mathsf{vk}_{\mathsf{opn}})$, and return $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk})$.

$\mathsf{Prove}_{\mathsf{eval1}}(\mathsf{ek},(a,b),c,p)$**:** Compute a proof $\pi_{\mathsf{opn}} \leftarrow \mathsf{Prove}_{\mathsf{opn}}(\mathsf{ek}_{\mathsf{opn}}, c, p)$, the polynomial $w(X)$ such that $w(X)\cdot(X-a) \equiv p(X) - b$ set $\pi \leftarrow [w(s)]_1$, and output $(\pi_{\mathsf{opn}}, \pi)$.

$\mathsf{Verify}_{\mathsf{eval1}}(\mathsf{vk},(a,b),c,\pi)$**:** Parse $\pi = (\pi_{\mathsf{opn}}, [w]_1)$, and output 1 iff:

1. $\mathsf{Verify}_{\mathsf{opn}}(\mathsf{vk}_{\mathsf{opn}}, c, \pi_{\mathsf{opn}}) = 1$ and
2. $e([w]_1,[s-a]_2) = e(c-[b]_1,[1]_2)$.

Efficiency. We give efficiency ignoring the costs of $\mathsf{CP}_{\mathsf{opn}}$. Generating a proof requires $\mathsf{deg}(p)$ $\mathbb{G}_1$-exponentiations to compute π and $O(\mathsf{deg}(p))$ $\mathbb{F}$-operations to compute the polynomial $w(X)$. Verification requires: 2 pairings.

Security. In the following theorem we state the security of $\mathsf{CP}_{\mathsf{eval},1}$.

Theorem 7.5. *If $\mathsf{CP}_{\mathsf{opn}}$ is a $\mathtt{swh}$-restricted CP-SNARK for $\mathcal{R}_{\mathsf{opn}}$ and the d-SDH assumption holds for GenG, then $\mathsf{CP}_{\mathsf{eval},1}$ is a complete (resp. $\mathtt{swh}$-restricted complete), knowledge sound, and trapdoor-commitment zero-knowledge CP-SNARK for $\mathsf{R}_{\mathsf{eval},1}$ for CS_1 (resp. CS_2). Moreover, if $\mathsf{CP}_{\mathsf{opn}}$ has commitment-only SRS then $\mathsf{CP}_{\mathsf{eval},1}$ has commitment-only SRS.*

Proof. The proof of completeness and knowledge soundness follow from previous works [KZG10, CHM$^+$20] and is therefore omitted. To see trapdoor-commitment zero-knowledge, notice that with the trapdoor $s \in \mathbb{Z}_q$ we can compute $[w]_1 := (c-[b]_1)/(s-a)$, moreover, we can simulate π_{opn} using the simulator of $\mathsf{CP}_{\mathsf{opn}}$. □

7.5 CP-SNARK for batch evaluation of many polynomials

We define a CP-SNARK $\mathsf{CP}_{\mathsf{eval}}$ for the commitment schemes CS_1, CS_2 and the relation $\mathsf{R}_{\mathsf{eval}}$ which is the Cartesian product of $\ell \in \mathbb{N}$ instances of $\mathsf{R}_{\mathsf{eval},1}$. The $\mathsf{CP}_{\mathsf{eval}}$ we propose is complete for CS_1 and $\mathtt{swh}^\ell$-restricted complete for CS_2. This scheme is essentially a CP-SNARK version of the batched polynomial commitment evaluation technique in [GWC19, CHM$^+$20].

The intuition for the construction is as follows. To prove that two polynomials p and p' committed to c and c' evaluate to b and b' on the point a, by linearity of the polynomials and classical batch argument we can simply show that a random linear combination $p^* = \rho p + \rho' p'$ of p and p' evaluate to $\rho b + \rho' b'$. Notice that by the homomorphic property of the commitment scheme we can compute $c^* = \rho c + \rho' c'$ which is a valid commitment of p^*. Generalizing, of the ℓ points $a_1, \ldots, a_\ell$ on which we want to evaluate the proofs, we gather the ℓ^* distinct ones. For each of these we compute an evaluation proof by batching the polynomials together.

$\mathsf{KeyGen}_{\mathsf{eval}}(\mathsf{ck})$**:** this proceeds identically as the key generation of $\mathsf{CP}_{\mathsf{eval},1}$.

$\mathsf{Prove}_{\mathsf{eval}}(\mathsf{ek},(a_j,b_j)_{j\in[\ell]},(c_j)_{j\in[\ell]},(p_j)_{j\in[\ell]})$**:**

1. Let $W := \{j \in [\ell] : \mathsf{type}(c_j) = \mathtt{swh}\}$ be the set of indices of type-$\mathtt{swh}$ commitments. Compute $\pi_{\mathsf{opn}} \leftarrow \mathsf{Prove}_{\mathsf{opn}}(\mathsf{ek}_{\mathsf{opn}}, (c_j)_{j\in W}, (p_j)_{j\in W})$;
2. For $j \in [\ell]$ set $\rho_j \leftarrow \mathcal{H}(\hat{\mathsf{x}}\|j)$ and let $\{a_1^*, \ldots, a_{\ell^*}^*\} = \{a_j\}_{j\in[\ell]}$ (repeated values are not counted), let $P_1, \ldots, P_{\ell^*}$ be a partition of the set $[\ell]$ such that $P_k = \{j : a_j = a_k^*\}$;

[18] An extension to support evaluations on $\mathtt{rel}$-typed ccommitments in CS_2 is straightforward; it is omitted as it's not needed in our work.

3. For $k \in [\ell^*]$ compute $c^*_k \leftarrow \sum_{j\in P_k} \rho_j \cdot c_j$, $p^*_k = \sum_{j\in P_k} \rho_j \cdot p_j$, and $b^*_k = p^*_k(a^*_k)$;
   Compute $\pi_k \leftarrow \overline{\mathsf{Prove}_{\mathsf{eval1}}}(\mathsf{ek}, (a^*_k, b^*_k), c^*_k, p^*_k)$.
4. Return $\pi = (\pi_{\mathsf{opn}}, (\pi_j)_{j\in[\ell^*]})$.

$\mathsf{Verify}_{\mathsf{eval}}(\mathsf{vk}, (a_j, b_j)_{j\in[\ell]}, (c_j)_{j\in[\ell]}, \pi)$**:** Compute W as described in the step 1 of the prover, and compute $(\rho_j)_{j\in[\ell]}, (a^*_j)_{j\in[\ell^*]}, (c^*_j)_{j\in[\ell^*]}$ and $(P_j)_{j\in[\ell^*]}$ as described in steps 2 and 3 of the prover. Parse $\pi = (\pi_{\mathsf{opn}}, (\pi_j)_{j\in[\ell^*]})$, and return 1 iff :

1. $\mathsf{Verify}_{\mathsf{opn}}(\mathsf{srs}, (c_j)_{j\in W}, \pi_{\mathsf{opn}}) = 1$ and,
2. for all $k \in [\ell^*]$ we have $\overline{\mathsf{Verify}_{\mathsf{eval1}}}(\mathsf{vk}, (a^*_k, b^*_k), c^*_k, \pi_k) = 1$.

Efficiency. We give efficiency ignoring the costs of $\mathsf{CP}_{\mathsf{opn}}$. Generating a proof requires $\mathsf{deg}(p^*_k)$ $\mathbb{G}_1$-exponentiations and $O(\mathsf{deg}(p^*_k))$ $\mathbb{F}$-operations to compute each π_k. Verification requires $2\ell^*$ pairings, which can be reduced to a total of 2 using standard batching techniques.

Security. In the following theorem we state the security of $\mathsf{CP}_{\mathsf{eval}}$.

Theorem 7.6. *If $\mathsf{CP}_{\mathsf{opn}}$ is a $\mathtt{swh}^\ell$-restricted CP-SNARK for $\mathcal{R}_{\mathsf{opn}}$ and the d-SDH assumption holds for GenG, then $\mathsf{CP}_{\mathsf{eval}}$ is a CP-SNARK for CS_1 (resp. CS_2) that is: complete (resp. $\mathtt{swh}^\ell$-restricted complete), knowledge-sound (with partial opening of type-$\mathtt{rel}$ commitments) in the random oracle model, and trapdoor-commitment zero-knowledge in the SRS model. Moreover, if $\mathsf{CP}_{\mathsf{opn}}$ has commitment-only SRS then $\mathsf{CP}_{\mathsf{eval}}$ has commitment-only SRS.*

Proof sketch. The proof of this theorem is an extension of the one of Theorem 7.5; we only provide a sketch. The main difference is in the knowledge soundness. First, notice that by Theorem 7.5 we have that each of the ℓ^* polynomial evaluations is correct. The correctness of all the ℓ evaluations then follows from a classical batching argument using the randomizers $\rho_1, \ldots, \rho_\ell$. ☐

7.6 CP-SNARK for Polynomial Equations

We describe a CP-SNARK for polynomial equations that relies on the one for batched polynomial evaluations given in the previous section. This CP-SNARK is based on the optimizations proposed by [GWC19].

Although the formal general treatment of our scheme has several technical details, its intuition is simple. At the high-level, we verify each polynomial equation by sampling a random point, exploiting the Schwartz-Zippel Lemma and reducing the problem to proving polynomial evaluation. For example, we pick random point u and then reduce proving $a(X)b(X)c(X) + d(X) = 0$ to $a(u)b(u)c(u) + d(u) = 0$. Then, for each monomial of degree d at least 2 in the polynomial equation, we recursively prove evaluation for a monomial of degree $d - 1$. For example, assume monomial $a(u)b(u)c(u)$ above equals value y, then we could reduce to $y_a b(u)c(u) = y$ by providing y_a and relative proof to the verifier. We could then do this again for b by providing a proof that $b(u) = y_b$, y_b and then reducing to $y_a y_b c(u) = y$. At this point we obtained a linear equation and we can use the approach of $\mathsf{Prove}_{\mathsf{eval}}$. In the example above we first started from a and then moved to b leaving c last, but clearly there are different recursion strategies. Some of them will be more efficient than others. Below, we abstracted away this aspect through minimal set S defined as in the pseudocode.

We define a class of CP-SNARKs for subsets of the relation R_{eq} (see Section 3.1). In particular, let C be a checker, we implicitly parameterize the CP-SNARK with the checker C. Consider the following relation:

$$\mathsf{R}_{\mathsf{eq,C}} = \left\{ (G^{(j)}, \boldsymbol{v}^{(j)})_{j\in[k]}, (p_j)_{j\in[\ell]} : \begin{array}{c} \forall i \in [\ell], x \in \mathbb{Z}_q : \mathsf{C}(i, v_i(x)) = 1 \wedge \\ \mathsf{R}_{\mathsf{eq}}((G, \boldsymbol{v}), (p_j)_{j\in[\ell]}) = 1 \end{array} \right\}$$

We define a $(\mathbf{1}, \mathsf{C})$-leaky zero-knowledge CP-SNARK $\mathsf{CP}_{\mathsf{eq}}$ for the commitment scheme CS and the relation $\mathsf{R}_{\mathsf{eq,C}}$. Let $\mathcal{H}$ be a random oracle from $\{0,1\}^*$ to $\mathbb{Z}_q$.

$\mathsf{KeyGen}_{\mathsf{eq}}(\mathsf{ck})$**:** execute $(\mathsf{ek}_{\mathsf{opn}}, \mathsf{vk}_{\mathsf{opn}}) \leftarrow \mathsf{KeyGen}_{\mathsf{opn}}(\mathsf{ck})$ and $(\mathsf{ek}_{\mathsf{eval}}, \mathsf{vk}_{\mathsf{eval}}) \leftarrow \mathsf{KeyGen}_{\mathsf{eval}}(\mathsf{ck})$ define $\mathsf{ek} := (\mathsf{ek}_{\mathsf{opn}}, \mathsf{ek}_{\mathsf{eval}})$ and $\mathsf{vk} := (\mathsf{vk}_{\mathsf{opn}}, \mathsf{vk}_{\mathsf{eval}})$, and return $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk})$.

$\mathsf{Prove}_{\mathsf{eq}}(\mathsf{ek}, (G, \boldsymbol{v}), (c_j)_{j\in[\ell]}, (p_j)_{j\in[\ell]})$**:** Execute the following steps.

1. Let $W := \{j \in [\ell] : \mathsf{type}(c_j) = \mathtt{swh}\}$ be the set of indices of type-swh commitments. Compute $\pi_{\mathsf{opn}} \leftarrow \mathsf{Prove}_{\mathsf{opn}}(\mathsf{ek}_{\mathsf{opn}}, (c_j)_{j\in W}, (p_j)_{j\in W})$;
2. Let $\hat{\mathsf{x}} := ((G^{(j)}, \boldsymbol{v}^{(j)})_{j\in[k]}, (c_j)_{j\in[\ell]})$ and set $\rho \leftarrow \mathcal{H}(\hat{\mathsf{x}}\|\pi_{\mathsf{opn}})$. For any $l \in [k]$ if $\mathsf{deg}_X(G^{(l)}(X, v_1^{(l)}(X), \ldots, v_\ell^{(l)}(X))) > 0$, and, for $j \in [\ell]$, let $a_j^{(l)} \leftarrow v_j^{(l)}(\rho)$, $b_j^{(l)} = p_j(a_j^{(l)})$; otherwise, let $a_j^{(l)} \leftarrow v_j^{(l)}(0)$ and $b_j^{(l)} = p_j(a_j^{(l)})$.
3. For any $l \in [k]$ let $\{a^*_{1,l}, \ldots, a^*_{\ell^*,l}\} = \{a_j^{(l)}\}_{j\in[\ell]}$ (repeated values are not counted), let $P_1^{(l)}, \ldots, P_{\ell^*}^{(l)}$ be a partition of the set $[\ell]$ such that $P_t^{(l)} = \{j : a_j^{(l)} = a^*_{t,l}\}$;
4. Let S be the minimal subset of $[\ell]$ such that (1) exists an index i^* such that $\bar{S} = [\ell] \setminus S \subseteq P_{i^*}$, (2) the polynomial $G(x, X_1, \ldots, X_\ell)$ has degree zero or one in the variables $\{X_j\}_{j\in S}$.
5. Let $\sum_{j\in\bar{S}} l_j X_j + l_0$ be equivalent to the polynomial G with the variables $(X_j)_{j\in S}$ assigned to the values $(b_j)_{j\in S}$ and the variable X assigned to the value ρ.
6. Let $a^* = a^*_{i^*},\ b^* = -l_0, c^* = \sum_{j\in\bar{S}} l_j \cdot c_j$, $p^* = \sum_{j\in\bar{S}} l_j \cdot p_j$.
   Let $\hat{\mathsf{x}}' = ((a_j, b_j, c_j)_{j\in S}, (a^*, b^*, c^*)))$, namely $\hat{\mathsf{x}}'$ is a vector of $|S| + 1$ instances of $\mathsf{R}_{\mathsf{eval}}$.
   Compute $\pi_{\mathsf{eval}} \leftarrow \overline{\mathsf{Prove}_{\mathsf{eval}}}(\mathsf{ek}_{\mathsf{eval}}, \hat{\mathsf{x}}', ((p_j)_{j\in S}, p^*, p_\ell))$
7. Output $(\pi_{\mathsf{opn}}, \{b_j\}_{j\in S}, \pi_{\mathsf{eval}})$.

$\mathsf{Verify}_{\mathsf{eq}}(\mathsf{vk}, \hat{\mathsf{x}}, \pi)$**:** Parse $\pi = (\pi_{\mathsf{opn}}, \{b_j\}_{j\in S}, \pi_{\mathsf{eval}})$. Execute the steps 2,3, 4 and 5 of the prover (but do not compute the values $(b_j)_{j\in[\ell]}$, bur rather take $(b_j)_{j\in[S]}$ from the proof). Also compute W as in step 1. Compute the commitment c^* as in step 6 of the prover, set $a^* = a^*_i$ and $b^* = -l_0$. We observe that l_0 can be computed efficiently since it depends only on the linear terms of G involving the values $\{b_j\}_{j\in S}$ and the constant term of $G(\rho, \ldots)$.

Return 1 iff:

1. $\mathsf{Verify}_{\mathsf{opn}}(\mathsf{vk}_{\mathsf{opn}}, (c_j)_{j\in[W]}, \pi_{\mathsf{opn}}) = 1$,
2. Set $\hat{\mathsf{x}}'$ as in step 6 of the prover, $\overline{\mathsf{Verify}_{\mathsf{eval}}}(\mathsf{vk}_{\mathsf{eval}}, \hat{\mathsf{x}}', \pi_{\mathsf{eval}}) = 1$ and,
3. $\forall j \in [\ell] : \mathsf{C}(j, v_j(a_j)) = 1$.

Efficiency. We give efficiency ignoring the costs of $\mathsf{CP}_{\mathsf{opn}}$. Generating a proof requires generating a batched evaluation proof for $|S| + 1$ committed polynomials (see previous section). Verification requires $O(|G|)$ $\mathbb{F}$-operations for the partial evaluation of G and to recover the l_j coefficients, plus the cost of one batched evaluation verification (2 pairings).

Security. In the following theorem we state the security of $\mathsf{CP}_{\mathsf{eq}}$.

Theorem 7.7. *Let $\mathsf{CP}_{\mathsf{opn}}$ and $\mathsf{CP}_{\mathsf{eval}}$ be CP-SNARKs over commitment scheme CS for relations $\mathcal{R}_{\mathsf{opn}}$ and $\mathsf{R}_{\mathsf{eval}}$ respectively. Then $\mathsf{CP}_{\mathsf{eq}}$ is a CP-SNARK over CS that is knowledge-sound (with partial opening of type-*`rel` *commitments), and* `swh`*-typed $(\mathbf{1}, \mathsf{C})$-leaky zero-knowledge. Moreover, if $\mathsf{CP}_{\mathsf{opn}}$ and $\mathsf{CP}_{\mathsf{eval}}$ have commitment-only SRSs then $\mathsf{CP}_{\mathsf{eval}}$ has a commitment-only SRS.*

Before proving the theorem we make the following observation.

Remark 13 (On more fine-grained leakage)**.** *With a closer look, we observe that this scheme is actually (b, C)-leaky zero-knowledge, for a b such that $\mathsf{b}_i = 1$ if $i \in S$ and $\mathsf{b}_i = 0$ otherwise. This is because evaluations of polynomials are revealed only if the index j is included in S.*

Proof. Knowledge Soundness follows by the extractability of $\mathsf{CP}_{\mathsf{opn}}$, the Schwartz-Zippel Lemma and the knowledge soundness of $\mathsf{CP}_{\mathsf{eval}}$. In particular, it is enough to extract only from type-`swh` commitments as we only have to prove knowledge soundness with partial openings of type-`rel` commitments.

More in detail, for any benign relation sampler $\mathcal{RG}_{\mathsf{Com}}$ and auxiliary input sampler $\mathcal{Z}$ consider the adversary $\mathcal{A}$ that outputs an instance $\hat{\mathsf{x}} = (G, (v_j)_{j\in[\ell]}, (c_j)_{j\in[\ell]})$ and a proof π, along with polynomials p_j such that $\mathsf{type}(c_j) = \mathtt{rel}$. By the knowledge soundness of $\mathsf{CP}_{\mathsf{opn}}$ we can extract polynomials $(p_j)_{j\in[\ell]}$.

Moreover, since $\rho = \mathcal{H}(\hat{\mathsf{x}}\|\pi_{\mathsf{opn}})$ the value ρ is independent from the polynomials $G, \boldsymbol{v}$ and $(p_j)_{j\in[\ell]}$ and uniformly random over $\mathbb{Z}_q$. Thus applying the Schwartz-Zippel lemma, if the polynomial $G'(X) := G(X, (p_j(v_j(X)))_{j\in[\ell]})$ evaluates to 0 on ρ then $G'(X) \equiv 0$. We conclude noticing that, by the knowledge soundness of $\mathsf{CP}_{\mathsf{eval}}$ it holds that $\forall j \in S : p_j(a_j) = b_j$ and $\sum_{j\in\bar{S}} l_j p_j(a^*) = b^*$ thus $G'(\rho) = 0$.

We show that $\mathsf{CP}_{\mathsf{eq}}$ is $(\mathbf{1}, \mathsf{C})$-leaky zero-knowledge. Let $\hat{\mathsf{x}}$ a valid instance of $\mathsf{R}^{\mathsf{Com}}_{\mathsf{eq},\mathsf{C}}$. Consider the simulator $\mathcal{S}_{\mathsf{leak}}(\hat{\mathsf{x}})$ that computes $a_j \leftarrow v_j(\mathcal{H}(\hat{\mathsf{x}}\|\pi_{\mathsf{opn}}))$ for $j \in [\ell]$ and outputs the list $\mathcal{L} = \{(j, a_j)\}_{j\in[\ell]}$. By definition of $\hat{\mathsf{x}}$ and by inspection of $\mathcal{L}$, the list $\mathcal{L}$ is $(\mathbf{1}, \mathsf{C})$-bounded. The simulator $\mathcal{S}_{\mathsf{prv}}(\mathsf{td_k}, \hat{\mathsf{x}}, leak)$ simulates the proof π_{opn}, then parses $leak$ as $(b_j)_{j\in[\ell]}$ where $b_j = p_j(a_j)$, letting $\mathcal{S}_{\mathsf{prv}}'$ be simulator of $\mathsf{CP}_{\mathsf{eval}}$ and $\hat{\mathsf{x}}' = ((a_j, b_j, c_j)_{j\in S}, (a^*, b^*, c^*)))$, computes $\pi' \leftarrow \mathcal{S}_{\mathsf{prv}}'(\mathsf{td_k}, \hat{\mathsf{x}}')$ and outputs $(\pi_{\mathsf{opn}}, \{b_j\}_{j\in S}, \pi_{\mathsf{poly}})$.

The indistinguishability easily follows by the zero-knowledge of the proofs of $\mathsf{CP}_{\mathsf{eval}}$ and of $\mathsf{CP}_{\mathsf{opn}}$. □

7.7 A CP-SNARK for CS_2 for quadratic polynomial equations

Let us consider the following relation in which G is an ℓ-variate polynomial of degree 2:

$$\mathsf{R}_{\mathsf{qeq}}(G, (p_j)_{j\in[\ell]}) := G(p_1(X), \dots, p_\ell(X)) \stackrel{?}{\equiv} 0$$

$\mathsf{R}_{\mathsf{qeq}}$ is a simplification of R_{eq} in which the degree of G is restricted to 2, each $v_j(X) = X$, and we removed the first variable X.

Here we show a simple CP-SNARK for the commitment scheme CS_2 and the above relation $\mathsf{R}_{\mathsf{qeq}}$. This scheme is novel and to the best of our knowledge it did not appear in previous work. The techniques are inspired by the linear interactive proof compiler of [BCI+13].

The basic intuition is rather simple, when G satisfies the restriction above it is possible to homomorphically compute G over $(p_1(s), \dots, p_\ell(s))$ in the target group using pairings and the linear property of the commitments. Like for the previous scheme, our approach is based on Schwartz-Zippel. Only, this time we exploit the random point s hidden in the SRS of the commitment scheme for polynomial evaluation. Thus all the verifier needs to do is verify a pairing product for each of the monomials of the type $a(X)b(X)$. For this to be possible, it needs to have each of the two polynomials a and b in two distinct groups. This is the case if they are committed through different types, i.e., one as type `rel` and the other as type `swh`. Otherwise, if they are both committed in the same group, we let the prover send one of the two polynomials committed in the "symmetric" group. Like in $\mathsf{CP}_{\mathsf{eq}}$ we abstract the most efficient approach to do this through a minimal set, in this case set $\mathcal{J}$ as defined in the pseudocode.

$\mathsf{KeyGen}_{\mathsf{qeq}}(\mathsf{ck})$: execute $(\mathsf{ek}_{\mathsf{opn}}, \mathsf{vk}_{\mathsf{opn}}) \leftarrow \mathsf{KeyGen}_{\mathsf{opn}}(\mathsf{ck})$ and return $\mathsf{srs} := (\mathsf{ek}_{\mathsf{opn}}, \mathsf{vk}_{\mathsf{opn}})$.

$\mathsf{Prove}_{\mathsf{qeq}}(\mathsf{ek}, G, (c_j)_{j\in[\ell]}, (p_j)_{j\in[\ell]})$:
first, let $W := \{j \in [\ell] : \mathsf{type}(c_j) = \mathtt{swh}\}$ be the set of indices of type-`swh` commitments, and compute $\pi_{\mathsf{opn}} \leftarrow \mathsf{Prove}_{\mathsf{opn}}(\mathsf{ek}_{\mathsf{opn}}, (c_j)_{j\in W}, (p_j)_{j\in W})$. Then proceed as follows:

- Consider the undirected graph where $V = [\ell]$ and there is an edge $\{i, j\}$ if $\mathsf{type}(c_i) = \mathsf{type}(c_j)$ and the term $(X_i \cdot X_j)$ is non zero in G.
- Let $\mathcal{J}$ be the min-cut of such graph, namely the minimal set of nodes that cover all the edges of G.
- For any $j \in \mathcal{J}$:
  if $\mathsf{type}(c_j) = \mathtt{swh}$, compute $c'_j = [p_j(s)]_2$;
  if $\mathsf{type}(c_j) = \mathtt{rel}$, compute $c'_j = [p_j(s)]_1$;
- Let $\mathcal{C}' = \{c'_j\}_{j\in\mathcal{J}}$ and output $\pi := (\pi_{\mathsf{opn}}, \mathcal{C}')$.

Output $\pi := (\pi_{\mathsf{opn}}, \mathcal{C}')$.

$\mathsf{Verify}_{\mathsf{qeq}}(\mathsf{vk}, \hat{\mathsf{x}}, \pi)$: parse $\mathcal{C}' = \{c'_j\}_j$. Reconstruct the set $\mathcal{J}$ as in the prover algorithm, and return 1 if and only if all the following checks pass:

1. $\mathsf{Verify}_{\mathsf{opn}}(\mathsf{vk}_{\mathsf{opn}}, (c_j)_{j\in[W]}, \pi_{\mathsf{opn}}) = 1$, for W computed as in step 1 of the prover;
2. for all $j \in \mathcal{J}$, check $e(c_j, [1]_2) = e([1]_1, c'_j)$ (if $\mathsf{type}(c_j) = \mathtt{swh}$) or $e(c'_j, [1]_2) = e([1]_1, c_j)$ (if $\mathsf{type}(c_j) = \mathtt{rel}$);
3. $[\hat{G}((c'_j)_{j\in[\ell]}, (c'_j)_{j\in\mathcal{J}})]_T \stackrel{?}{=} [1]_T$, where $\hat{G}$ is a modified version of G where the computation of a quadratic term involving only c_j is performed as $e(c_j, c'_j)$ (or $e(c'_j, c_j)$).

Efficiency. Generating a proof requires $|\mathcal{J}|$ operations of $\mathbb{G}_1$ or $\mathbb{G}_2$ to compute each c'_j. Verification requires $2|\mathcal{J}|$ pairings in step 2 and $t+1$ pairings in step 3, where t is the number of quadratic terms in G. Here we ignored the cost of $\mathsf{CP_{opn}}$ as well as that to compute the min-cut $\mathcal{J}$; in our applications this is trivial and can be given as a parameter.

Security. In the following theorem we state the security of $\mathsf{CP_{qeq}}$.

Theorem 7.8. *If $\mathsf{CP_{opn}}$ is a CP-SNARK for $\mathcal{R}_{\mathsf{opn}}$ over CS_2 then $\mathsf{CP_{qeq}}$ is a complete, knowledge-sound (with partial opening of type-*`rel` *commitments) zero-knowledge CP-SNARK for $\mathsf{R_{qeq}}$ over CS_2 under the d-DLOG assumption for* GenG*. Moreover, if $\mathsf{CP_{opn}}$ has a commitment-only SRS then $\mathsf{CP_{qeq}}$ has a commitment-only SRS.*

Proof Sketch. We define the extractor of $\mathsf{CP_{qeq}}$ to be the same as the $\mathsf{CP_{opn}}$ extractor. By the knowledge soundness of $\mathsf{CP_{opn}}$, such extractor returns a tuple of polynomials $(p_j)_{j\in[\ell]}$ such that for every $j \in [\ell]$ it holds $c_j = [p_j(s)]_{g_j}$ for the appropriate group g_j.

We want to bound the probability that for the extracted polynomials it holds $G(p_1(X), \dots, p_\ell(X)) \not\equiv 0$ (while the proof accepts). Let us define $p^*(X) = G(p_1(X), \dots, p_\ell(X))$. Since the proof accepts we have $p^*(s) = G(p_1(s), \dots, p_\ell(s)) = 0$. Then we can factor $p^*(X)$ to recover the root s, and thus break d-DLOG assumption. □

7.8 CP-SNARKs for degree of committed polynomials

In this section we show two CP-SNARKs, $\mathsf{CP}^{(\star)}_{\mathsf{deg}}$ and $\mathsf{CP}^{(2)}_{\mathsf{deg}}$, for proving a bound on the degree of committed polynomials, namely they work for the universal relation $\mathcal{R}_{\mathsf{deg}}$ in which every $\mathsf{R_{deg}} \in \mathcal{R}_{\mathsf{deg}}$ consists of a vector $(d_j)_{j\in[\ell]}$ of degrees, such that every $d_j \in [d]$, and the relation is satisfied if and only if $\forall j : \mathsf{deg}(p_j) \le d_j$.

The basic idea of the schemes is the following. To prove that $\mathsf{deg}(p) \le d^*$ one commits to the shifted polynomial $p^*(X) = X^{d-d^*}p(X)$ and then proves that the polynomial equation $X^{d-d^*} \cdot p(X) - p^*(X) = 0$ using a CP-SNARK for polynomial equations. This idea is extended in order to batch together these proofs for several polynomials.

The two schemes $\mathsf{CP}^{(\star)}_{\mathsf{deg}}$ and $\mathsf{CP}^{(2)}_{\mathsf{deg}}$ follow this approach with the only difference that $\mathsf{CP}^{(2)}_{\mathsf{deg}}$ makes use of the optimized scheme $\mathsf{CP_{qeq}}$ for quadratic equations. Indeed, $X^{d-d^*} \cdot p(X) - p^*(X) = 0$ can be seen as a quadratic equation in which the polynomial X^{d-d^*} can be committed in $\mathbb{G}_2$ by the Derive algorithm.

Therefore we have that $\mathsf{CP}^{(\star)}_{\mathsf{deg}}$ can work with both commitment schemes CS_1 and CS_2, while $\mathsf{CP}^{(2)}_{\mathsf{deg}}$ works with CS_2 only. Both CP-SNARKs are $\mathtt{swh}^\ell$-restricted complete.

Finally, we remark that in the CP-SNARKs below we assume that the degree bounds are always strictly less than the maximal degree d supported by the commitment key ck. In fact, for such d a proof for $\mathsf{deg}(p) \le d$ is for free.

7.8.1 Scheme $\mathsf{CP}^{(\star)}_{\mathsf{deg}}$

We define the CP-SNARK $\mathsf{CP}^{(\star)}_{\mathsf{deg}} = (\mathsf{KeyGen_{deg}}, \mathsf{Prove_{deg}}, \mathsf{Verify_{deg}})$ as follows.

$\mathsf{KeyGen_{deg}}(\mathsf{ck})$**:** execute $(\mathsf{ek_{opn}}, \mathsf{vk_{opn}}) \leftarrow \mathsf{KeyGen_{opn}}(\mathsf{ck})$, execute $(\mathsf{ek_{eq}}, \mathsf{vk_{eq}}) \leftarrow \mathsf{KeyGen_{eq}}(\mathsf{ck})$, define $\mathsf{ek} := (\mathsf{ck}, \mathsf{ek_{opn}}, \mathsf{ek_{eq}})$ and $\mathsf{vk} := (\mathsf{vk_{opn}}, \mathsf{vk_{eq}})$, and return $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk})$.

$\mathsf{Prove_{deg}}(\mathsf{ek}, (d_j)_{j\in[\ell]}, (c_j)_{j\in[\ell]}, (p_j)_{j\in[\ell]})$**:**

1. Compute $\pi_{\mathsf{opn}} \leftarrow \mathsf{Prove_{opn}}(\mathsf{srs}, (c_j)_{j\in[\ell]}, (p_j)_{j\in[\ell]})$;
2. Let $\rho_1, \dots, \rho_\ell \leftarrow \mathcal{H}((d_j)_{j\in[\ell]}, (c_j)_{j\in[\ell]}, \pi_{\mathsf{opn}})$, let $\{d^*_1, \dots, d^*_{\ell^*}\} = \{d_j\}_{j\in[\ell]}$ (repeated values are not counted), and let $P_1, \dots, P_{\ell^*}$ be a partition of the set $[\ell]$ such that $P_k = \{j : d_j = d^*_k\}$;
3. For all $i \in [\ell^*]$ let $c'_i \leftarrow [p'(s)]_1 := [\sum_{j\in P_i} \rho_j \cdot p_j(s)]_1$ and $c^*_i \leftarrow [p^*_i(s)]_1 := [s^{d-d^*_i} \cdot \sum_{j\in P_i} \rho_j \cdot p_j(s)]_1$;
4. Compute $\pi^*_{\mathsf{opn}} \leftarrow \mathsf{Prove_{opn}}(\mathsf{srs}, (c^*_j)_{j\in[\ell^*]}, (p^*_j)_{j\in[\ell^*]})$.

5. For all $i \in [\ell^*]$, define $G_i(X, X'_i, X^*_i) = X^{d-d^*_i} \cdot X'_i - X^*_i$, $v^{(i)}_1(X) = v^{(i)}_2(X) = X$, and compute $\pi_{\mathsf{eq}} \leftarrow \overline{\mathsf{Prove}}_{\mathsf{eq}}(\mathsf{ek}_{\mathsf{eq}}, (G_i, \boldsymbol{v}^{(i)})_{i\in[\ell^*]}, (c'_i, c^*_i)_{i\in[\ell^*]}, (p'_i, p^*_i)_{i\in[\ell^*]})$
6. Return $(\pi_{\mathsf{opn}}, c^*_1, \ldots, c^*_{\ell^*}, \pi^*_{\mathsf{opn}}, \pi_{\mathsf{eq}})$.

$\mathsf{Verify}_{\mathsf{deg}}(\mathsf{vk}, (d_j)_{j\in[\ell]}, (c_j)_{j\in[\ell]}, \pi)$**:** Parse $\pi = (\pi_{\mathsf{opn}}, c^*_1, \ldots, c^*_{\ell^*}, \pi^*_{\mathsf{opn}}, \pi_{\mathsf{eq}})$, and compute $\rho_1, \ldots, \rho_\ell$ and $G_i, \boldsymbol{v}^{(i)}$ as the prover does. Return 1 iff :

1. $\mathsf{Verify}_{\mathsf{opn}}(\mathsf{vk}_{\mathsf{opn}}, (c_j)_{j\in[\ell]}, \pi_{\mathsf{opn}}) = 1$ and,
2. $\mathsf{Verify}_{\mathsf{opn}}(\mathsf{vk}_{\mathsf{opn}}, (c^*_j)_{j\in[\ell^*]}, \pi^*_{\mathsf{opn}}) = 1$ and,
3. $\mathsf{Verify}_{\mathsf{eq}}(\mathsf{vk}_{\mathsf{eq}}, (G_i, \boldsymbol{v}^{(i)})_{i\in[\ell^*]}, (c'_i, c^*_i)_{i\in[\ell^*]}, \pi_{\mathsf{eq}}) = 1$.

Efficiency. Generating a proof requires $\ell^* \cdot d^*$ $\mathbb{G}_1$-exponentiations and $O(\ell \cdot d^*)$ $\mathbb{F}$-operations, where $d^* = \max_{j\in[\ell]}\{\mathsf{deg}(p_j)\}$, the cost of generating two $\mathsf{CP}_{\mathsf{opn}}$ proofs and one $\mathsf{CP}_{\mathsf{eq}}$ proof. Verification requires verifying two $\mathsf{CP}_{\mathsf{opn}}$ proofs and one $\mathsf{CP}_{\mathsf{eq}}$ proof.

Remark 14 (Optimization)**.** *When $\mathsf{CP}^{(\star)}_{\mathsf{deg}}$ is used in a larger protocol that uses other invocations of $\mathsf{CP}_{\mathsf{eq}}$, we observe that these proofs can batched together (which in turn implies for example the use of the same random point, and of the same $\mathsf{CP}_{\mathsf{eval}}$ proof).*

Security. In the following theorem we state the security of $\mathsf{CP}^{(\star)}_{\mathsf{deg}}$.

Theorem 7.9. *If $\mathsf{CP}_{\mathsf{opn}}$ is a CP-SNARK for $\mathcal{R}_{\mathsf{opn}}$ and CS_1 (resp. CS_2), and $\mathsf{CP}_{\mathsf{eq}}$ is a CP-SNARK for $\mathcal{R}_{\mathsf{eq}}$, then $\mathsf{CP}^{(\star)}_{\mathsf{deg}}$ is a knowledge-sound and zero-knowledge CP-SNARK for CS_1 (resp. CS_2).*

Proof. Let $(p_j)_{j\in[\ell]}$ and $(p^*_j)_{j\in[\ell^*]}$ be the polynomials extracted from π_{opn} and π^*_{opn} by the knowledge soundness of $\mathsf{CP}_{\mathsf{opn}}$. We notice that $\rho_1, \ldots, \rho_\ell$ are uniformly random and independent of $(p_j)_{j\in[\ell]}$, since we can extract the polynomials before answering the random oracle query $((c_j)_{j\in[\ell]}, \pi_{\mathsf{opn}})$.

Thus with overwhelming probability, for every i the polynomial $p'_i(X) = \sum_{j\in P_i} \rho_j p_j(X)$ has degree equal to $\max_{j\in P_i} \mathsf{deg}(p_j)$. Suppose exists i such that p^*_i has degree bigger than d^*_i. Then for the same index i we have that $X^{d-d^*_i} p'_i(X) - p^*_i(X) \neq 0$. However, if this is the case, then we can build a reduction against the soundness of $\mathsf{CP}_{\mathsf{eq}}$.

Zero-knowledge is straightforward: the commitments $c^*_1, \ldots, c^*_{\ell^*}$ are deterministic functions of the random oracle $\mathcal{H}$, the values $(c_j)_{j\in[\ell]}$ and the trapdoor s, while the remaining proofs π^*_{opn} and π_{eq} can be generated by using the simulators of $\mathsf{CP}_{\mathsf{opn}}$ and $\mathsf{CP}_{\mathsf{eq}}$ respectively. ☐

7.8.2 Scheme $\mathsf{CP}^{(2)}_{\mathsf{deg}}$.

We define the CP-SNARK $\mathsf{CP}^{(2)}_{\mathsf{deg}} = (\mathsf{KeyGen}_{\mathsf{deg}}, \mathsf{Derive}_{\mathsf{deg}}, \mathsf{Prove}_{\mathsf{deg}}, \mathsf{Verify}_{\mathsf{deg}})$ as follows.

$\mathsf{KeyGen}_{\mathsf{deg}}(\mathsf{ck})$**:** execute $(\mathsf{ek}_{\mathsf{opn}}, \mathsf{vk}_{\mathsf{opn}}) \leftarrow \mathsf{KeyGen}_{\mathsf{opn}}(\mathsf{ck})$, execute $(\mathsf{ek}_{\mathsf{qeq}}, \mathsf{vk}_{\mathsf{qeq}}) \leftarrow \mathsf{KeyGen}_{\mathsf{qeq}}(\mathsf{ck})$, parse ck as $([s^j]_1, [s^j]_2)_{j\in[0,d]}$ define $\mathsf{ek} := (\mathsf{ck}, \mathsf{ek}_{\mathsf{opn}}, \mathsf{ek}_{\mathsf{qeq}})$ and $\mathsf{vk} := (([s^j]_2)_{j\in[0,d]}, \mathsf{vk}_{\mathsf{opn}}, \mathsf{vk}_{\mathsf{qeq}})$, and return $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk})$.

$\mathsf{Derive}_{\mathsf{deg}}((d_j)_{j\in[\ell]})$ generates a verification key for the vector of degrees $(d_j)_{j\in[\ell]}$ as follows. Let $\{d_j\}_{j\in[\ell]} := \{d^*_1, \ldots, d^*_{\ell^*}\}$ (repeated values are not counted), and set $\mathsf{vk}_{\boldsymbol{d}} := ([s^{d-d^*_j}]_2)_{i\in[\ell^*]}$.

$\mathsf{Prove}_{\mathsf{deg}}(\mathsf{ek}, (c_j)_{j\in[\ell]}, (p_j)_{j\in[\ell]})$**:**

1. Compute $\pi_{\mathsf{opn}} \leftarrow \mathsf{Prove}_{\mathsf{opn}}(\mathsf{srs}, (c_j)_{j\in[\ell]}, (p_j)_{j\in[\ell]})$;
2. Let $\rho_1, \ldots, \rho_\ell \leftarrow \mathcal{H}((d_j)_{j\in[\ell]}, (c_j)_{j\in[\ell]}, \pi_{\mathsf{opn}})$, let $\{d^*_1, \ldots, d^*_{\ell^*}\} = \{d_j\}_{j\in[\ell]}$ (repeated values are not counted), and let $P_1, \ldots, P_{\ell^*}$ be a partition of the set $[\ell]$ such that $P_k = \{j : a_j = a^*_k\}$;
3. For all $i \in [\ell^*]$ let $c'_i \leftarrow [p'(s)]_1 := [\sum_{j\in P_i} \rho_j \cdot p_j(s)]_1$ and $c^*_i \leftarrow [p^*(s)]_1 := [s^{d-d^*_i} \cdot \sum_{j\in P_i} \rho_j \cdot p_j(s)]_1$;
4. Compute $\pi^*_{\mathsf{opn}} \leftarrow \mathsf{Prove}_{\mathsf{opn}}(\mathsf{srs}, (c^*_j)_{j\in[\ell^*]}, (p^*_j)_{j\in[\ell^*]})$.
5. For all $i \in [\ell^*]$, define $G_i(\hat{X}_i, X'_i, X^*_i) = \hat{X}_i \cdot X'_i - X^*_i$, and compute

$$\pi_{\mathsf{qeq}} \leftarrow \overline{\mathsf{Prove}}_{\mathsf{qeq}}(\mathsf{ek}_{\mathsf{eq}}, (G_i, \boldsymbol{v}^{(i)})_{i\in[\ell^*]}, ([s^{d-d^*_i}]_2, c'_i, c^*_i)_{i\in[\ell^*]}, (X^{d-d^*_i}, p'_i, p^*_i)_{i\in[\ell^*]})$$

6. Return $(\pi_{\mathsf{opn}}, c_1^*, \ldots, c_{\ell^*}^*, \pi_{\mathsf{opn}}^*, \pi_{\mathsf{qeq}})$.

$\mathsf{Verify}_{\mathsf{deg}}(\mathsf{vk}_{\boldsymbol{d}}, (c_j)_{j\in[\ell]}, \pi)$**:** Parse $\pi = (\pi_{\mathsf{opn}}, c_1^*, \ldots, c_{\ell^*}^*, \pi_{\mathsf{opn}}^*, \pi_{\mathsf{eq}})$, and compute $\rho_1, \ldots, \rho_\ell$ and $G_i, \boldsymbol{v}^{(i)}$ as the prover does. Return 1 iff :

1. $\mathsf{Verify}_{\mathsf{opn}}(\mathsf{vk}_{\mathsf{opn}}, (c_j)_{j\in[\ell]}, \pi_{\mathsf{opn}}) = 1$ and,
2. $\mathsf{Verify}_{\mathsf{opn}}(\mathsf{vk}_{\mathsf{opn}}, (c_j^*)_{j\in[\ell^*]}, \pi_{\mathsf{opn}}^*) = 1$ and,
3. $\mathsf{Verify}_{\mathsf{qeq}}(\mathsf{vk}_{\mathsf{eq}}, (G_i)_{i\in[\ell^*]}, ([s^{d-d_i^*}]_2, c_i', c_i^*)_{i\in[\ell^*]}, \pi_{\mathsf{eq}}) = 1$.

The security proof of $\mathsf{CP}_{\mathsf{deg}}^{(2)}$ works essentially the same as that of $\mathsf{CP}_{\mathsf{deg}}^{(\star)}$ and is therefore omitted.

Theorem 7.10. *If* $\mathsf{CP}_{\mathsf{opn}}$ *is a CP-SNARK for* $\mathcal{R}_{\mathsf{opn}}$ *over* CS_2*, and* $\mathsf{CP}_{\mathsf{eq}}$ *is a CP-SNARK for* $\mathcal{R}_{\mathsf{eq}}$ *over* CS_2*, then* $\mathsf{CP}_{\mathsf{deg}}^{(2)}$ *is a knowledge-sound and zero-knowledge CP-SNARK for* $\mathsf{R}_{\mathsf{deg}}$ *over* CS_2*.*

Efficiency. From the shape of all the quadratic polynomials G_i and the construction of $\mathsf{CP}_{\mathsf{qeq}}$ in Section 7.7, we observe that the proof π_{qeq} is empty and it can be verified by checking, for every $i \in [\ell^*]$, $e(\sum_{j\in P_i} \rho_j c_j, [s^{d-d_i^*}]_2) = e(\pi_i, [1]_2)$. The cost of generating the rest of the proof requires generating two $\mathsf{CP}_{\mathsf{opn}}$ proofs, $\ell^* \cdot d^*$ $\mathbb{G}_1$-exponentiations and $O(\ell \cdot d^*)$ $\mathbb{F}$-operations, where $d^* = \max_{j\in[\ell]}\{\mathsf{deg}(p_j)\}$. Verification additionally requires verification of two $\mathsf{CP}_{\mathsf{opn}}$ proofs.

7.9 A general-purpose CP-SNARK for $\mathcal{R}_{\mathsf{php}}$

Given the CP-SNARKs presented in the previous section, it is possible to construct CP-SNARKs for the commitment schemes CS_1, CS_2 and for *any* PHP verifier checks, i.e., for the relation $\mathcal{R}_{\mathsf{php}}$ discussed in Section 3.1. Such a CP-SNARK $\mathsf{CP}_{\mathsf{php}}$ can be obtained with three main building block CP-SNARKs: one for $\mathcal{R}_{\mathsf{opn}}$ (see Section 7.3), one for proving a bound on the degree of committed polynomials, and one for polynomial equations.

8 Our Compiler for Universal Commit-and-Prove zkSNARKs

In this section we show how to compile PHPs into CP-SNARKs. We present the compiler in Section 8.1. It can be instantiated with the same building blocks presented in the previous section, plus additional ones that we present in Section 8.2,

8.1 Compiling to Commit-and-Prove Universal Interactive Arguments

We show how to adapt the compiler of Section 6.2 to produce a *commit-and-prove* succinct interactive argument in the SRS model.

Let PHP be a PHP protocol for a universal relation $\mathcal{R}$ such that for any triple $(\mathsf{R}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$, the witness splits into an $\ell+1$-tuple $\mathsf{w} := ((\mathsf{u}_j)_{j\in[\ell]}, \omega) \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell \times \mathcal{D}_\omega$.

We show how to compile PHP to a *commit-and-prove* UIA for $\mathcal{R}$ in which prover and verifier take as inputs commitments $c_1, \ldots, c_\ell$ to $\mathsf{u}_1, \ldots, \mathsf{u}_\ell$ respectively. More in detail, UIA is a universal commit-and-prove argument for $\mathcal{R}$ and a type-based commitment scheme CS^* such that the commitments taken as input are of type $\mathtt{lnk}$ and satisfy full-fledged hiding. The reason to require these commitments to be hiding (instead of our weaker somewhat-hiding notion) is that these are supposed to be "regular" commitments that may be generated independently of this proof system and that, for a general application scenario, should hide messages even if they are re-used an unbounded number of times for different proofs.[19] We summarize the requirements on CS^* in the following definition.

Definition 8.1 (CP-Compiling Commitment Scheme)**.** *Let* $\mathcal{R}$ *be a universal relation such that for any* $(\mathsf{R}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$*,* $\mathsf{w} := ((\mathsf{u}_j)_{j\in[\ell]}, \omega) \in \mathcal{D}_1 \times \ldots \mathcal{D}_\ell \times \mathcal{D}_\omega$*. We say* $\mathsf{CS}^* = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCom})$ *is a* CP-compiling commitment scheme *if it is a type-based commitment scheme for a class of types* $\mathcal{T} = \{\mathtt{rel}, \mathtt{swh}, \mathtt{lnk}\}$*, such that:*

[19] Note that this for example rules out the use of polynomial commitments with "bounded-use randomizers" such as the one in [KZG10].

- *commitments of type* `rel` *and* `swh` *are for messages that are polynomials* $\mathbb{F}_{<d}[X]$ *for a given bound* $d \in \mathbb{N}$*;*
- *commitments of type* `lnk` *are for messages in* $\mathcal{D}$ *such that for all* $i \in [\ell]$, $\mathcal{D}_i \subseteq \mathcal{D}$*;*
- *it is* $\mathcal{T}$*-binding;*
- *it is* `swh`*-somewhat-hiding and* `lnk`*-hiding.*

Additional building blocks. Besides the requirements of Section 6.2, we additionally require from the CS^* and from PHP the following properties:

1. The PHP has a straight-line extractor (see Definition 3.3). Specifically, there exists an efficient extractor $\mathsf{WitExtract}$ such that $\mathsf{WitExtract}((p_j)_{j\in[\mathsf{n}^*]}) = \mathsf{w}$.
2. CS^* is equipped with a zero-knowledge CP-SNARK $\mathsf{CP}_{\mathsf{link}} = (\mathsf{KeyGen}_{\mathsf{link}}, \mathsf{Prove}_{\mathsf{link}}, \mathsf{VerProof}_{\mathsf{link}})$ that can "link" a tuple of `lnk`-typed commitments (opening to $(\mathsf{u}_j)_{j\in[\ell]}$) with a tuple of n^* `swh`-typed commitments. The linking relation should also enforce that the latter commitments open to polynomials that somehow contain a witness for a universal relation. Specifically, $\mathsf{CP}_{\mathsf{link}}$ is a $(\{\mathtt{lnk}\}^{\ell} \times (\mathtt{swh})^{\mathsf{n}^*})$-restricted complete ZK CP-SNARK for the universal relation $\mathcal{R}_{\mathsf{link}}$ parametrized by the algorithm $\mathsf{WitExtract}$ and by a PT decoding algorithm Decode:

$$\mathsf{R}_{\mathsf{link}}((\mathsf{u}_j)_{j\in[\ell]}, (p_j)_{j\in[\mathsf{n}^*]}, \omega) := \mathsf{WitExtract}((p_j)_{j\in[\mathsf{n}^*]}) \stackrel{?}{=} (\mathsf{Decode}((\mathsf{u}_j)_{j\in[\ell]}), \omega)$$

We additionally require the Decode algorithm for a rather technical reason. Namely, the commitment scheme CS could encode the witness blocks u_i in different way, the decoding algorithm casts back to strings the encoding used by the commitment scheme CS.

The commit-and-prove compiler. Let $\mathsf{UIA} = (\mathsf{KeyGen}, \mathsf{Derive}, \mathbb{P}, \mathbb{V})$ be the interactive protocol for $\mathcal{R}$ from the last section. We show how we can make it commit-and-prove with some simple modifications.

In what follows, to distinguish the commitments (and the associated openings) taken as input by the protocol, from the commitments generated during the interaction, we denote the former ones with a hat.

Consider the interactive protocol that is the same as UIA but with the following modifications:

- The KeyGen algorithm does not sample a commitment key from $\mathsf{CS.Setup}$ but instead takes a commitment key ck of the CS^* commitment scheme.
- The prover on input ek, $\hat{\mathsf{x}} = (\mathsf{x}, (\hat{c}_j)_{j\in[\ell]})$ and $\hat{\mathsf{w}} = ((\mathsf{u}_j)_{j\in[\ell]}, (\hat{o}_j)_{j\in[\ell]}, \omega)$, executes the same as $\mathbb{P}(\mathsf{vk}, \mathsf{x}, ((\mathsf{u}_j)_{j\in[\ell]}, \omega))$. Let $(c_j)_{j\in[k]}, (p_j)_{j\in[k]}, (o_j)_{j\in[k]}$ be the k-tuples of commitments, polynomials and openings corresponding to the indices of the witness-carrying polynomials.
- At the last round the prover computes

$$\pi_{\mathsf{link}} \leftarrow \mathsf{Prove}_{\mathsf{link}}(\mathsf{ek}_{\mathsf{link}}, ((\hat{c}_j)_{j\in[\ell]}, (c_j)_{j\in[k]}), ((\mathsf{u}_j)_{j\in[\ell]}, (p_j)_{j\in[k]}), ((\hat{o}_j)_{j\in[\ell]}, (o_j)_{j\in[k]}), \omega)$$

- At the last round the verifier additionally checks $b_{\mathsf{link}} \leftarrow \mathsf{Verify}_{\mathsf{link}}(\mathsf{vk}_{\mathsf{link}}, ((\hat{c}_j)_{j\in[\ell]}, (c_j)_{j\in[\mathsf{n}^*]}), \pi_{\mathsf{link}})$, and output 1 if all the CP-SNARK proofs verify.

Theorem 8.1. *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* $\mathcal{F}$ *and* $\mathcal{R}$*, let* CS^* *be a compiling commitment scheme as in Definition 8.1 equipped with CP-SNARKs* $\mathsf{CP}_{\mathsf{opn}}$ *for* $\mathcal{R}_{\mathsf{opn}}$*,* $\mathsf{CP}_{\mathsf{php}}$ *for a relation* $\mathcal{R}_{\mathsf{php}}$*, and* $\mathsf{CP}_{\mathsf{link}}$ *for* $\mathsf{R}_{\mathsf{link}}$*. Then we have:*

- *If* PHP *has witness-carrying polynomials, then the scheme* UIA *defined above is a universal commit and prove interactive argument in the SRS model for* $\mathcal{R}'$ *such that:*

$$(\mathsf{R}, \mathsf{x}, (\mathsf{u}_j)_{j\in[\ell]}, \omega) \in \mathcal{R}' \iff (\mathsf{R}, \mathsf{x}, \mathsf{Decode}((\mathsf{u}_j)_{j\in[\ell]}), \omega) \in \mathcal{R}.$$

- *If, for a checker* C*,* PHP *(resp.* $\mathsf{CP}_{\mathsf{php}}$*) is* $(\mathsf{b}+\mathbf{1}, \mathsf{C})$*-bounded honest-verifier zero knowledge (resp. trapdoor-commit* (b, C)*-leaky zero-knowledge), and both* $\mathsf{CP}_{\mathsf{opn}}$ *and* $\mathsf{CP}_{\mathsf{link}}$ *are trapdoor-commitment zero-knowledge, then* UIA *is trapdoor-commitment honest-verifier zero-knowledge.*

Note that the analogous of Remark 7 holds for this theorem as well.
While a proof of Theorem 8.1 is in Appendix C, we provide an intuition for the case $\ell = 1$. To prove knowledge soundness we should be able to extract $(\hat{o}, \mathsf{u}, \omega)$—valid CP-witnesses for the CP-version of R—from $\hat{c}, \mathsf{x}$. Let us assume that the verifier accepts. We can run the $\mathsf{CP}_{\mathsf{link}}$ extractor and obtain $\mathsf{u}, \hat{o}, \omega$ as well vas ector of polynomials $\boldsymbol{p}$ with respective openings for the c_j-s. By knowledge soundness of $\mathsf{CP}_{\mathsf{link}}$, u, ω as defined above extracted from the polynomials in $\boldsymbol{p}$. In turn, we can claim these polynomials encode valid witnesses for relation R because, if they didn't, we could obtain "valid" polynomials $\boldsymbol{p}'$ by running the extractor of $\mathsf{CP}_{\mathsf{php}}$. These would also be valid openings to commitments $\boldsymbol{c}$. If the polynomials in $\boldsymbol{p}'$ were distinct from the polynomials in $\boldsymbol{p}$ then we would be able to break binding; therefore, polynomials in $\boldsymbol{p}$ and $\boldsymbol{p}'$ must be identical and encode witnesses for R. In order to prove zero-knowledge we extend the simulator from Theorem 6.1 by appending to its output that of the simulator of $\mathsf{CP}_{\mathsf{link}}$. a zero-knowledge CP-SNARK. We run the latter by feeding it the appropriate commitments c_j-s corresponding to the k witness-carrying polynomials.

8.2 Pairing-Based Instantiations of our Building Blocks

8.2.1 Commitment Scheme

We describe the polynomial commitment scheme CS^* which supports types $\mathtt{lnk}, \mathtt{swh}, \mathtt{rel}$. The scheme is an extension of CS_g for $g \in \{1, 2\}$. The algorithms $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCom})$ of CS^*_g are defined as follows:

$\mathsf{Setup}(1^\lambda, d)$**:** run $\mathsf{ck}' \leftarrow \mathsf{Setup}_g(1^\lambda, d)$, sample random $\alpha \leftarrow\!\!\$\, \mathbb{F}$ and output $\mathsf{ck} = \mathsf{ck}', [\alpha, \alpha s, \alpha s^2]_1$.

$\mathsf{Commit}(\mathsf{ck}, \mathsf{type}, p) \rightarrow (c, o)$**:** if $\mathsf{type} \neq \mathtt{lnk}$ output the same as Commit_g, else sample $o \leftarrow\!\!\$\, \mathbb{F}$ and output $[p(s) + \alpha \cdot o]_1$.

$\mathsf{VerCom}(\mathsf{ck}, \mathsf{type}, c, p, o)$**:** if $\mathsf{type} \neq \mathtt{lnk}$ output the same as VerCom_g, else check if $c \stackrel{?}{=} p([s]_1) + o[\alpha]_1$.

Remark 15. *Notice that the values $[\alpha s, \alpha s^2]_1$ are not needed for hiding, however they are useful for the CP-SNARK for polynomial evaluation that we present next.*

8.2.2 Basic suite of CP-SNARKs for CS^*

CP-SNARK for $\mathcal{R}_{\mathsf{opn}}$. As described in Section 7.3 we can obtain trivially a CP-SNARK in the AGM. A trivial extension of the construction $\mathsf{CP}^{\mathsf{PKE}}_{\mathsf{opn}}$ of Section 7.3 is also suitable for CS^*. The only difference is that for the security analysis we need to rely on the following assumption:

Assumption 4 (mmPKE)**.** *The (multi-instance, multi-base) Power Knowledge of Exponent (mmPKE) assumption holds for a bilinear group generator* GenG *if for every efficient non-uniform adversary* $\mathcal{A}$ *and a degree bounds* $d_1, d_2 \in \mathbb{N}$ *there exists an efficient extractor* $\mathcal{E}$ *such that for any benign distribution* $\mathcal{Z}$ *the following probability is negligible in* λ*:*

$$\Pr\left[\exists j : \begin{array}{c} d_j = \gamma \cdot c_j \wedge \\ c_j \neq \sum_{k=0}^{d_1} a_k^{(j)} s^j + \alpha \sum_{k=0}^{d_2} b_k^{(j)} s^j \end{array} : \begin{array}{c} \mathsf{bgp} \leftarrow\!\!\$\, \mathsf{GenG}(1^\lambda); \\ \mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathsf{bgp}); \\ s, \alpha, \gamma \leftarrow\!\!\$\, \mathbb{Z}_q; \\ \Sigma = \left(\begin{array}{c} ([s^j]_1, [s^j]_2, [\gamma s^j]_1, [\gamma]_2)_{j \in [d_1]} \\ ([\alpha s^j]_1, [\alpha s^j]_2, [\alpha\gamma s^j]_1, [\alpha\gamma s^j]_2)_{j \in [d_2]} \end{array} \right); \\ (c_j)_{j \in [\ell']}, (d_j)_{j \in [\ell']} \leftarrow \mathcal{A}(\mathsf{bgp}, \Sigma, \mathsf{aux}_Z); \\ (\boldsymbol{a}^{(j)}, \boldsymbol{b}^{(j)})_{j \in [\ell']} \leftarrow \mathcal{E}(\mathsf{bgp}, \Sigma, \mathsf{aux}_Z) \end{array}\right].$$

CP-SNARK for $\mathsf{R}_{\mathsf{eval},1}$. We define a zero-knowledge CP-SNARK $\mathsf{CP}_{\mathsf{eval},1}$ for CS^*_g and the relation $\mathsf{R}_{\mathsf{eval},1}((a, b), p) := p(a) \stackrel{?}{=} b$, where p is committed as $[p(s) + \alpha \cdot o]_1$.

Kate et al. [KZG10] describe a method to do evaluation proofs for hiding polynomial commitments. In a nutshell, in their case a commitment to p is an element $[p(s) + \alpha \cdot o(s)]_1$ where o is a random polynomial of degree $\mathsf{deg}(p)$ (or degree b, if one aims to support at most b evaluation proofs) and the evaluation proof for a point a reveals $o(a)$. This technique, however, cannot be seen as a full-fledged commit-and-prove zero-knowledge proof as one should know a priori how many evaluation proofs are generated for a given commitment. More technically, in the commit-and-prove framework, a simulator would only take as input a commitment and must simulate a proof which must be indistinguishable from a real one, independently of how many other proofs have been already (or will be) generated. It is also interesting to note that for a polynomial p of degree d giving more than d evaluations of p on distinct points reveal the polynomial; thus one may think that zero-knowledge would no longer be needed. However, there are some applications in which one can use a commitment in more than d evaluation proofs without necessarily revealing d evaluations of the committed polynomial. This is for example the case if one shows evaluations of linear combinations of various committed polynomials to known constant, e.g., proving that $\rho_1 p_1(a) + \rho_2 p_2(a) = 0$. In this case, the technique from [KZG10] would leak information on the random polynomials o_1, o_2 and would be usable a limited number of times.

Motivated by this problem, we propose a different technique for proving an unbounded number of evaluations of committed polynomials in zero-knowledge.

Let us provide a brief intuition of our technique. In our CP-SNARK the prover additionally computes a $\mathtt{flh}$-typed commitment $\tilde{c}$ to the 0 polynomial using fresh randomness, and then proves that (1) $\tilde{c}$ indeed commits to the 0 polynomial, and (2) that the polynomial p committed in $c + \tilde{c}$ evaluates to b on the point a. The idea is that in the step (2) the prover masks the opening material of c using the *fresh* opening material of $\tilde{c}$. In particular, the prover picks a degree-2 polynomial for the opening of $\tilde{c}$ because we want to assure that the mask in (2) is uniformly random even given the value $\tilde{c}$ and the leakage (one evaluation point) in step (1).

$\mathsf{KeyGen}_{\mathsf{eval1}}(\mathsf{ck})$: execute $(\mathsf{ek}_{\mathsf{opn}}, \mathsf{vk}_{\mathsf{opn}}) \leftarrow \mathsf{KeyGen}_{\mathsf{opn}}(\mathsf{ck})$, parse ck as $([s^j]_1, [s^j]_2)_{j\in[0,d]}$ define $\mathsf{ek} := (\mathsf{ck}, \mathsf{ek}_{\mathsf{opn}})$ and $\mathsf{vk} := ([1, s]_2, \mathsf{vk}_{\mathsf{opn}})$, and return $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk})$.

$\mathsf{Prove}_{\mathsf{eval1}}(\mathsf{ek}, (a, b), c, p)$: Sample random degree-2 polynomial $\tilde{o}(X)$ and set $\tilde{c} = [\alpha\tilde{o}(s)]_1$, compute a proof $\pi_{\mathsf{opn}} \leftarrow \mathsf{Prove}_{\mathsf{opn}}(\mathsf{ek}_{\mathsf{opn}}, (c, \tilde{c}), (p, 0), (o, \tilde{o}))$, and set $(x) \leftarrow H(\hat{\mathsf{x}} \| \pi_{\mathsf{opn}} \| \tilde{c})$:

1. Let $y_1 \leftarrow \tilde{o}(x)$ and let $w'_1(X)$ such that $w'_1(X) \cdot (X - x) \equiv \tilde{o}(X) - y_1$;
2. Let $y_2 \leftarrow o(a) + \tilde{o}(a)$ and let $w(X), w'_2(X)$ such that $w(X) \cdot (X - a) \equiv p(X) - b$ and $w'_2(X) \cdot (X - a) \equiv o(X) + \tilde{o}(X) - y$.

set $\pi \leftarrow ([w'_1(s), w(s) + \alpha w'_2(s)]_1, y_1, y_2)$ and output $(\pi_{\mathsf{opn}}, \pi)$.

$\mathsf{Verify}_{\mathsf{eval1}}(\mathsf{vk}, (a, b), c, \pi)$: Parse $\pi = (\pi_{\mathsf{opn}}, ([w', w]_1, y_1, y_2))$, and output 1 iff:

1. $\mathsf{Verify}_{\mathsf{opn}}(\mathsf{vk}_{\mathsf{opn}}, c, \pi_{\mathsf{opn}}) = 1$,
2. $e([w']_1, [s - x]_2) = e(\tilde{c}, [1]_2) - [\alpha y_1]_T$, and
3. $e([w]_1, [s - a]_2) = e(c + \tilde{c}, [1]_2) - [b]_T - [\alpha y_2]_T$.

Theorem 8.2. *If* $\mathsf{CP}_{\mathsf{opn}}$ *is a* $\mathtt{flh}$*-restricted CP-SNARK for* $\mathcal{R}_{\mathsf{opn}}$ *and the d-SDH assumption holds for* GenG*, then* $\mathsf{CP}_{\mathsf{eval},1}$ *is a complete, knowledge sound, and trapdoor-commitment zero-knowledge CP-SNARK for* $\mathsf{R}_{\mathsf{eval},1}$ *for* CS^* *Moreover, if* $\mathsf{CP}_{\mathsf{opn}}$ *has commitment-only SRS then* $\mathsf{CP}_{\mathsf{eval},1}$ *has commitment-only SRS.*

Proof. The proof of knowledge soundness follow similar to [KZG10, CHM+20]. In particular, we notice that the second verification equation shows that $\tilde{c}$ is indeed a polynomial commitment to 0, thus, by the homomorphic property of the commitment scheme we have that $c + \tilde{c}$ is a commitment to p. Then the third equation shows that $p(a) = b$.

For zero-knowledge notice that $\tilde{o}$ is random degree-2 polynomial thus even given the evaluation points $\tilde{o}(x_1)$ and $\tilde{o}(s)$ the evaluation $\tilde{o}(a)$ is still uniformly random over $\mathbb{F}$ and it can be used to mask the value $o(a)$. We can simulate by sampling $\tilde{c} \leftarrow_\$ \mathbb{G}_1$ and $y_1, y_2 \leftarrow_\$ \mathbb{Z}_q^2$, then set $[w']_1 = (\tilde{c} - [\alpha]y_1)/(s - x)$ and $[w]_1 = (c + \tilde{c} - [b]_1 - [\alpha]_1 y_2)/(s - a)$. □

CP-SNARK for $\mathsf{R}_{\mathsf{eval}}$ and R_{eq}. Similar to Section 7.5 and Section 7.6, we can define the CP-SNARKs $\mathsf{CP}_{\mathsf{eval}}$ and $\mathsf{CP}_{\mathsf{eq}}$ for CS^*, indeed such constructions use $\mathsf{CP}_{\mathsf{eval},1}$ as a black-box thus we could easily instantiate `lnk`-restricted complete version of them using the $\mathsf{CP}_{\mathsf{eval},1}$ presented in the previous paragraph.

Efficiency. We give efficiency ignoring the costs of $\mathsf{CP}_{\mathsf{opn}}$. For $\mathsf{CP}_{\mathsf{eval},1}$ generating a proof requires $\mathsf{deg}(p) + 6$ $\mathbb{G}_1$-exponentiations to compute $\boldsymbol{\pi}$ and $O(\mathsf{deg}(p))$ $\mathbb{F}$-operations to compute the polynomials $w(X), w_1'(X), w_2'(X)$, verification requires 4 pairings. For $\mathsf{CP}_{\mathsf{eval}}$ generating a proof requires $\ell^* \cdot (\mathsf{deg}(p_k^*)+1)$ $\mathbb{G}_1$-exponentiations and $O(\ell^* \mathsf{deg}(p_k^*))$ $\mathbb{F}$-operations, verification requires $4\ell^*$ pairings, where we recall that ℓ^* is computed as the cardinality of the set of all the evaluation points (in particular $\ell^* \leq \ell$). For the counting for $\mathsf{CP}_{\mathsf{eq}}$ we refer to Section 7.6.

8.2.3 CP-SNARK for Linking Commitments

Finally, we propose instantiations for the $\mathsf{CP}_{\mathsf{link}}$ CP-SNARK that support our `lnk`-typed commitments and the $\mathsf{WitExtract}$ straight-line extractors of our PHPs.

Let us first consider the $\mathsf{WitExtract}$ algorithm of our PHPs for R1CS of Section 4.4; this simply uses one polynomial, $\hat{w}(X)$, and returns its evaluations on $\mathbb{H}' := \mathbb{H} \setminus \mathbb{L}$, i.e., $\boldsymbol{w} := (w(\phi_{\mathbb{H}}^{-1}(|\mathsf{x}| + 1)), \ldots, w(\phi_{\mathbb{H}}^{-1}(n)))$. Our goal is to support use cases in which one has commitments $\hat{c}_j$ to vectors u_j and wants to prove that $\boldsymbol{w} = ((\mathsf{u}_j)_{j\in[\ell]}, \omega)$.

We consider the following albraic setting. Let η be the generator of $\mathbb{H}$ so that $\mathbb{H} = (\eta, \eta^2, \ldots, \eta^n)$ and $\mathbb{H} \setminus \mathbb{L}$ can be partitioned in ordered form as $\mathbb{H}' = (W_1, \ldots, W_{\ell+1})$, where the sets $W_1, \ldots, W_\ell$ have the same cardinality. We define V as a "prefix" of $\mathbb{H}$, i.,e., $V = \{\eta, \ldots, \eta^{|V|}\}$. Although `lnk`-typed commitments in CS^* are defined for polynomials, we assume a canonical encoding of a vector u into a polynomial $\hat{\mathsf{u}}(X)$ via interpolation in V. This means that the Decode algorithm corresponding to the linking relation is the one that outputs $\hat{\mathsf{u}}_1(V), \ldots, \hat{\mathsf{u}}_\ell(V)$.[20]

Once fixed this setting, proving the linking between commitments $(\hat{c}_j)_{j\in[\ell]}$ to $(\hat{\mathsf{u}}_j)_{j\in[\ell]}$ and a commitment c to $\hat{w}$ requires to prove that there is a vector ω such that

$$(\hat{\mathsf{u}}_1(V), \ldots, \hat{\mathsf{u}}_\ell(V), \omega) = (\hat{w}(W_1), \ldots, \hat{w}(W_\ell), \hat{w}(W_{\ell+1}))$$

We create a CP-SNARK for this language in two steps. First we show a scheme $\mathsf{CP}_{\mathsf{link}}^{(1)}$ that proves

$$(\hat{\mathsf{u}}_1'(W_1), \ldots, \hat{\mathsf{u}}_\ell'(W_\ell), \omega) = (\hat{w}(W_1), \ldots, \hat{w}(W_\ell), \hat{w}(W_{\ell+1}))$$

for some freshly committed $(\hat{\mathsf{u}}_j')_{j\in[\ell]}$, and then a scheme $\mathsf{CP}_{\mathsf{link}}^{(2)}$ which runs $\mathsf{CP}_{\mathsf{link}}^{(1)}$ and additionally proves that for all $j \in [\ell]$, it holds $\hat{\mathsf{u}}_j'(W_j) = \hat{\mathsf{u}}_j(V)$.

Finally, at the end of the section, we discuss how to extend these results to support the $\mathsf{WitExtract}$ algorithm of our PHPs for R1CS-lite, in which the extractor uses two polynomials $\hat{a'}(X), \hat{b'}(X)$ (instead of one) and computes the witness as $\boldsymbol{w} := (\hat{a'}(\phi_{\mathbb{H}}^{-1}(\ell+1)) \cdot \hat{b}(\phi_{\mathbb{H}}^{-1}(\ell+1)), \ldots, \hat{a'}(\phi_{\mathbb{H}}^{-1}(n) \cdot \hat{b}(\phi_{\mathbb{H}}^{-1}(\ell+1)))$.

Scheme $\mathsf{CP}_{\mathsf{link}}^{(1)}$. We first show a scheme for tuple $\mathsf{Decode}, \mathsf{WitExtract}$ where $W_i = V_i$ for all $i \in [\ell]$ and the sets are disjoint. In particular, we can consider universal CP-SNARK for $\mathcal{R}_{\mathsf{link}}$ where each relation $\mathsf{R}_{\mathsf{link}}$ in the family is defined by a list of sets $(W_j)_{j\in[\ell]}$.

We let $\mathcal{Z}_i(X)$ be the vanishing polynomial on W_i, let $\hat{\mathcal{Z}}_i(X) := \prod_{1<j<i} \mathcal{Z}_j(X)$ (we set $\hat{\mathcal{Z}}_1(X) \equiv 1$). The intuition is to let the prover compute an (affine) decomposition of the polynomial $p_{i^*}(X)$ using the bases $\mathcal{Z}_1(X), \ldots, \mathcal{Z}_\ell(X)$, and similarly compute an (affine) decomposition of the polynomial $\hat{p}_j(X)$ using the base $\mathcal{Z}_j(X)$. If the statement holds then, for any i, the $\mathcal{Z}_i(X)$-coefficient of the decomposition of p and of the decomposition of $\hat{p}_i$ are the same polynomials.

$\mathsf{KeyGen}_{\mathsf{link}}(\mathsf{ck})$: execute and output $(\mathsf{ek}_{\mathsf{opn}}, \mathsf{vk}_{\mathsf{opn}}) \leftarrow \mathsf{KeyGen}_{\mathsf{opn}}(\mathsf{ck})$.

$\mathsf{Derive}_{\mathsf{link}}(\mathsf{srs}, \mathsf{R})$: Parse R as $(W_j)_{j\in[\ell]}$ and output $\mathsf{vk}_{\mathsf{R}} = ([\mathcal{Z}_i(s)]_2, [\hat{\mathcal{Z}}_i(s)]_2)_{j\in[\ell]}$.

[20] We parse the evaluation of a polynomial $p(W)$ on an ordered set W as a vector in $\mathbb{F}^{|W|}$.

$\mathsf{Prove}_{\mathsf{link}}(\mathsf{ek}, ((\hat{c}_j)_{j\in[\ell]}, (c_j)_{j\in[\mathsf{n}^*]}), ((\mathsf{u}_j)_{j\in[\ell]}, (p_j)_{j\in[\mathsf{n}^*]}), ((\hat{o}_j)_{j\in[\ell]}, (o_j)_{j\in[\mathsf{n}^*]}))$:

1. For $i \in [\ell]$ compute $q_i(X), \mathsf{u}'_i(X)$ such that $\mathsf{u}_i(X) \equiv q_i(X)\mathcal{Z}_i(X) + \mathsf{u}'_i(X)$.
   Sample $\gamma_i \leftarrow_{\$} \mathbb{F}$, set $o'_i = \mathcal{Z}_i(X)\gamma_i + o_i$ and
   $$c'_i = [\mathsf{u}'_i(s) + \alpha o'_i(s)]_1, \qquad\qquad [d_i] = [q_i(s) - \alpha\gamma_i].$$
2. Let $p := p_{i^*}$ and compute the polynomial $q(X)$ such that $p(X) - \sum_{i\in[\ell]} \hat{\mathcal{Z}}_i(X) \cdot \mathsf{u}'_i(X) \equiv \hat{\mathcal{Z}_{\ell+1}}(X) \cdot q(X)$;
3. Sample $\beta(X) = \beta_0 + X\beta_1$ random polynomial of degree 1 and set
   $$[d]_1 \leftarrow [q(s) + \alpha\beta(s))]_1.$$
4. Compute
   $$\pi_{\mathsf{opn}} \leftarrow \mathsf{Prove}_{\mathsf{opn}}\left(\mathsf{ek}_{\mathsf{opn}}, \begin{array}{ll} \hat{\mathsf{x}} & = ((\hat{c}_j)_{j\in[\ell]}, c_{i^*}, (c'_j)_{j\in[\ell]}, ([d_j]_1)_{j\in[\ell]}, [d]_1), \\ \hat{\mathsf{w}} & = ((\mathsf{u}_j)_{j\in[\ell]}, p, (\mathsf{u}'_j)_{j\in[\ell]}, (q_j)_{j\in[\ell]}, q), ((\hat{o}_j)_{j\in[\ell]}, 0, (\beta_j)_{j\in[\ell]}, (\gamma_j)_{j\in[\ell]}, \beta) \end{array}\right)$$
   (Namely, a proof for the opening for all the commitments to polynomials computed up to here and the all the commitments of the instance.)
5. Prove that indeed $p(X) - \sum_{i\in[\ell]} \hat{\mathcal{Z}}_i(X) \cdot \mathsf{u}'_i(X) \equiv \hat{\mathcal{Z}_{\ell+1}}(X) \cdot q(X)$ using random point evaluation. Specifically, let $\tau = \Big((\hat{c}_j)_{j\in[\ell]} \| (c_j)_{j\in[\mathsf{n}^*]} \| (c'_j)_{j\in[\ell]} \| ([d_i]_1)_{i\in[\ell]} \| [d]_1 \| \pi_{\mathsf{opn}}\Big)$. Let $x \leftarrow H(\tau)$ and let $\tilde{o}(X) := -\beta(X)\hat{\mathcal{Z}_{\ell+1}}(X) - \sum_j o'_j \hat{\mathcal{Z}}_j(X)$, compute $z = \tilde{o}(x)$, let $w(X)$ be the polynomial such that $w(X) \cdot (X - x) \equiv \tilde{o}(X) - z$.
6. Output $(\pi_{\mathsf{opn}}, (c'_j)_{j\in[\ell]}, [(d_j)_{j\in[\ell]}, d, \alpha w(s)]_1, z)$.

$\mathsf{Verify}(\mathsf{ck}, c, (\hat{c}_j)_{j\in[\ell]}, \pi)$: Parse $\pi = (\pi_{\mathsf{opn}}, (c'_j)_{j\in[\ell]}, [(d_j)_{j\in[\ell]}, d, w]_1, z$ and output 1 if and only if

1. For $j \in [\ell]$ check $e(\hat{c}_j, [1]_2) = e([d_j]_1, [\mathcal{Z}_i(s)]_2) + e(c'_j, [1]_2)$.
2. $\mathsf{Verify}_{\mathsf{opn}}(\mathsf{vk}_{\mathsf{opn}}, ((\hat{c}_j)_{j\in[\ell]}, c_{i^*}, (c'_j)_{j\in[\ell]}, ([d_j]_1)_{j\in[\ell]}, [d]_1), \pi_{\mathsf{opn}}) = 1$,
3. $e([w]_1, [s-x]) = e(c, [1]_2) - e([d]_1, [\hat{\mathcal{Z}_{\ell+1}}(s)]_2) - \sum_{i\in[\ell]} e(c'_i, [\hat{\mathcal{Z}}_i(s)]_2) - [\alpha z]_T$.

Theorem 8.3. *If* $\mathsf{CP}_{\mathsf{opn}}$ *is a CP-SNARK for* $\mathcal{R}_{\mathsf{opn}}$*, and* $\mathsf{CP}_{\mathsf{deg}}$ *is a CP-SNARK for* $\mathsf{R}_{\mathsf{deg}}$*, then* $\mathsf{CP}^{(1)}_{\mathsf{link}}$ *is* $\{\mathtt{swh}, \mathtt{lnk}\}^{\ell} \times (\mathtt{swh})^{\mathsf{n}^*}$*-restricted complete, knowledge-sound, and zero-knowledge. Moreover, if* $\mathsf{CP}_{\mathsf{opn}}$ *and* $\mathsf{CP}_{\mathsf{deg}}$ *have commitment-only SRS then* $\mathsf{CP}_{\mathsf{eval},1}$ *has a commitment-only SRS.*

Proof. We start with completeness. The check in step 1 holds in fact:

$$\begin{aligned} & e([d_j]_1, [\mathcal{Z}_i(s)]_2) + e(c'_j, [1]_2) = \\ & \quad = [(q_i(s) + \alpha\beta_i)\mathcal{Z}_i(s) + \mathsf{u}'(s) - \alpha(\mathcal{Z}_i(s)\beta_i + o_i)]_T = \\ & \quad = [q_i(s)\mathcal{Z}_i(s) + \mathsf{u}'(s) + \alpha o_i]_T = e([\hat{c}_i]_1, [1]_2) \end{aligned}$$

The check in step 2 of the verifier holds by the completeness of $\mathsf{CP}_{\mathsf{opn}}$. For the last check, notice the relation $\mathsf{R}_{\mathsf{link}}$ can be expressed as $\forall i \in [\ell] : p_{i^*}(W_i) = \mathsf{u}_i(W_i)$. By the definition of the $\mathsf{u}'_i(X)$ we have $\mathsf{u}_i(W_i) = \mathsf{u}'_i(W_i)$ for $i \in [\ell]$. Moreover:

$$\begin{aligned} & e(c_{i^*}, [1]_2) - e([d]_1, [\hat{\mathcal{Z}_{\ell+1}}(s)]_2) - \sum_{i\in[\ell]} e(c'_i, [\hat{\mathcal{Z}}_i(s)]_2) - [\alpha z]_T \\ = & [\underbrace{p_{i^*}(s) - q(s)\hat{\mathcal{Z}_{\ell+1}}(s) - \sum_{i\in[\ell]} \hat{\mathcal{Z}}_i(s) \cdot \mathsf{u}'_i(s)}_{=0} + \alpha(o - \beta(s)\hat{\mathcal{Z}_{\ell+1}}(s) - \sum_j o_j \hat{\mathcal{Z}}_j(s) - \tilde{o}(x))]_T \\ = & [\alpha(\tilde{o}(s) - \tilde{o}(z))]_T = [\alpha w(s) \cdot (s-x)]_T = e([w]_1, [s-x]_2) \end{aligned}$$

Now we move to knowledge soundness.

Let $d(X), o_d(X), p(X), o_p(X), (\mathsf{u}_j)_{j\in[\ell]}, (\hat{o}_j)_{j\in[\ell]}, (d_j, \mathsf{u}_j, \mathsf{u}'_j, o_{d_j}, o_{\mathsf{u}_j}, o_{\mathsf{u}'_j})_{j\in[\ell]}$ be the output of the extractor of $\mathsf{CP}_{\mathsf{opn}}$. First we show that for any j and $\eta \in W_i$ it musts be $\mathsf{u}'(\eta) = \mathsf{u}(\eta)$. In fact, suppose not, if $o_{\mathsf{u}_j}(s) - o_{\mathsf{u}'_j}(s) - o_{q_j}(X)v_j(s) = 0$ then it musts be that $P_j(X) := \mathsf{u}_j(X) - \mathsf{u}'_j(X) - q_j(X)\mathcal{Z}_j(X) \not\equiv 0$, since the third equation of the verifier holds then s is a zero of the polynomial P_j. Thus we can break the d-DLOG assumption. The other case is when $o_{\mathsf{u}_j}(s) - o_{\mathsf{u}'_j}(s) - o_{q_j}(s)v_j(s) \neq 0$, in this we could sample s and on challenge the value $[\alpha]_1$, since the third equation of the verifier holds, we can compute the value α as $(\mathsf{u}_j(s) - \mathsf{u}'_j(s) - q_j(s)\mathcal{Z}_j(s))/(o_{\mathsf{u}_j}(s) - o_{\mathsf{u}'_j}(s) - o_{q_j}(s)\mathcal{Z}_j(s))$.

Suppose $\exists i, \eta : p_{i^*}(\eta) \neq \mathsf{u}_i(\eta) = \mathsf{u}'_i(\eta)$ and $\eta \in W_i$. We let:

$$P(X) := p(X) - \hat{\mathcal{Z}_{\ell+1}}(X)d(X) - \sum_j \hat{\mathcal{Z}}_j(X)\hat{p}_j(X)$$

Notice that the Eq. 3 of the verification algorithm implies that $P(x) = 0$ and x is uniformly random and independent of $P(X)$ because x is sampled after the proof π_{opn} is computed (and therefore the polynomial can be extracted before x is sampled). By the Swartz-Zippel lemma we have that $P(X) \equiv 0$. The proof that $P(x) = 0$ follows the same line of the proof of knowledge soundness of $\mathsf{CP}_{\mathsf{eval},1}$, therefore omitted. By our hypothesis, the polynomial $P(X)$ on point η evaluates to $p(\eta) - \hat{p}_i(\eta) \neq 0$, which leads to contradiction.

We now prove zero-knowledge. We can simulate π_{opn} using the simulator of $\mathsf{CP}_{\mathsf{opn}}$, moreover, given the trapdoor s for any $j \in [\ell]$ we can sample a random value r_j and set $c'_j = [r_j]_1$ and set $[q_j] = (\hat{c}_j - c'_j)/\mathcal{Z}_i(s)$ and we can sample $d, z \leftarrow_\$ \mathbb{Z}_q$ and define $[w]_1 = [\alpha\tilde{o}(s) - \tilde{o}(x)]_1/(s - x)$.

We show this is indistinguishable from the real distribution with an hybrid argument. In the first hybrid we are additionally given the witness $p, (\hat{p}_j)_{j\in[\ell]}, o, (\hat{o}_j)_{j\in[\ell]}$. We compute the proofs $\pi_{\mathsf{deg}}, \pi_{\mathsf{opn}}$ as the real prover would do and we compute the value $q(s)$, which is deterministically defined given the witness, we sample d, z as in the simulator and then compute β_0, β_1 such that:

$$\begin{cases} q(s) + \alpha(\beta + s\beta_1) = d \\ o - (\beta_0 + x\beta_1) - \sum_j \hat{\mathcal{Z}}_j(s)o_j = z \end{cases}$$

The group element $[w]_1$ is computed as the simulator does. Notice the marginal distribution of β_0, β_1 is the uniform distribution over $\mathbb{Z}_q^2$.

In the next hybrid we sample d, z at random, compute $\beta(X) := \beta_0 + X\beta_1$ as described before, compute $d' = q(s) + \alpha\beta(s)$ and $z' = \tilde{o}(x)$ Compute $[w]_1$ as in the simulator and output $(\pi_{\mathsf{opn}}, \pi_{\mathsf{deg}}, [d', w]_1, z')$. This hybrid is equivalent to the previous one in fact $d' = d$ and $z' = z$.

The next hybrid we compute the proof $[w]_1$ in the same way the prover does. Notice that once fixed the witness and d', z' the value w is deterministically defined, thus the two hybrids are equivalent.

□

We give efficiency ignoring the costs of $\mathsf{CP}_{\mathsf{opn}}$. Generating a proof requires approximately $\ell+3$ multi-exponentiations with bases of size $\max|W_i|$. and $O(\ell \max|W_i|)$ $\mathbb{F}$-operations. Verification requires $3\ell+3$ pairings.

Scheme $\mathsf{CP}^{(2)}_{\mathsf{link}}$. We show a scheme for tuple $\mathsf{Decode}, \mathsf{WitExtract}$ where there exists a subset V of a subgroup $\mathbb{H} = \langle\eta\rangle$ of $\mathbb{F}$, and let the values for $j \in [\ell]$ $\theta_j = \eta^{j|V|+|x|}$ (recall that in our PHPs for R1CS of Section 4.4, both the instance and the witness are commit in the witness-carrying polynomial) and set $W_j = \theta_j \cdot V$. (We can assume that $\mathbb{H}$ is big enough so that $W_j \neq W_{j'}$ for all j, j' where $j \neq j'$.)

The intuition is to first "shift" the polynomials u_j computing polynomials u'_j such that $\mathsf{u}_j(V) = \mathsf{u}'_j(\theta_j \cdot V)$ and secondly to apply the CP-SNARK $\mathsf{CP}^{(1)}_{\mathsf{link}}$. To prove the soundness of the shifted polynomials we make black-box use of $\mathsf{CP}_{\mathsf{eq}}$ from the previous section, however the scheme is only leaky zero-knowledge, thus to obtain zero-knowledge we additionally need to randomize the polynomials u'_j.

$\mathsf{KeyGen}_{\mathsf{link}}(\mathsf{ck})$: execute $(\mathsf{ek}_{\mathsf{eval}}, \mathsf{vk}_{\mathsf{eval}}) \leftarrow \mathsf{KeyGen}_{\mathsf{eq}}(\mathsf{ck})$ execute $(\mathsf{ek}^{(1)}_{\mathsf{link}}, \mathsf{vk}^{(1)}_{\mathsf{link}}) \leftarrow \overline{\mathsf{KeyGen}}^{(1)}_{\mathsf{link}}(\mathsf{ck})$ and return $\mathsf{srs} := ((\mathsf{ek}_{\mathsf{eq}}, \mathsf{ek}^{(1)}_{\mathsf{link}}), (\mathsf{vk}_{\mathsf{eq}}, \mathsf{vk}^{(1)}_{\mathsf{link}}))$.

$\mathsf{Derive}_{\mathsf{link}}(\mathsf{srs}, \mathsf{R})$: Parse R as $V, (W_j)_{j\in[\ell]}$, compute $\mathsf{vk}_{\mathsf{R}}^{(1)} \leftarrow \mathsf{CP}_{\mathsf{link}}^{(1)}.\mathsf{Derive}(\mathsf{srs}, \mathsf{R})$, and output $\mathsf{vk}_{\mathsf{R}} = \mathsf{vk}_{\mathsf{R}}^{(1)}, ([\mathcal{Z}_i(s)]_1)_{j\in[\ell]}$.

$\mathsf{Prove}_{\mathsf{link}}(\mathsf{ek}, \hat{\mathsf{x}} = ((\hat{c}_j)_{j\in[\ell]}, (c_j)_{j\in[\mathsf{n}^*]}), ((\mathsf{u}_j)_{j\in[\ell]}, (p_j)_{j\in[\mathsf{n}^*]}), ((\hat{o}_j)_{j\in[\ell]}, (o_j)_{j\in[\mathsf{n}^*]}))$:

1. For any $j \in [\ell]$ let:
$$\mathsf{u}'_j(X) = \mathsf{u}_j(X/\theta_j) + \mathcal{Z}_V(X/\theta_j)\beta_j(X/\theta_j),$$
where $\beta_j(X)$ is a uniformly random degree-2 polynomial in $\mathbb{F}[X]$. Set $c'_j = [\mathsf{u}'_j(s)]_1$ for $j \in [\ell]$.
2. Prove that for $j \in [\ell]$, $\mathsf{u}_j(X)$ and $\mathsf{u}'_j(\theta_j \cdot X)$ agree on V, namely, prove that $\mathsf{u}_j(X) - \mathsf{u}'_j(\theta_j X) \equiv 0 \mod \mathcal{Z}_V(X)$ using $\mathsf{CP}_{\mathsf{eq}}$.
Specifically, compute $h_j(X)$ such that $h_j(X)\mathcal{Z}_V(X) \equiv \mathsf{u}_j(X) - \mathsf{u}'_j(X\theta_j)$ and set $\mathsf{h}_j = [h_j(s)]_1$ for $j \in [\ell]$; Let
$$G_j(X, (X_j)_{j\in[3\ell+1]}) = X_i - X_{2i} - X_{3i} \cdot X_{3\ell+1} \qquad j \in [\ell],$$
$$v_j(X) = \begin{cases} X & j \in [\ell] \cup [2\ell+1, 3\ell] \\ \theta_j X & j \in [\ell+1, 2\ell] \end{cases}$$
and set $\boldsymbol{v} = (v_j)_{j\in[3\ell]}$. Compute π_{eq} as the output of
$$\mathsf{Prove}_{\mathsf{eq}}\left((G_i, \boldsymbol{v})_{i\in[\ell]}, (\hat{c}_j)_{j\in[\ell]}, (c'_j)_{j\in[\ell]}, (\mathsf{h}_j)_{j\in[\ell]}, [\mathcal{Z}_V(s)]_1,\ ((\mathsf{u}_j)_{j\in[\ell]}, (\mathsf{u}'_j)_{j\in[\ell]}, (h_j)_{j\in[\ell]}),\ ((o_j)_{j\in[\ell]}, \mathbf{0})\right)$$
3. Compute $\pi \leftarrow \mathsf{CP}_{\mathsf{link}}^{(1)}.\overline{\mathsf{Prove}}(\mathsf{ek}_{\mathsf{link}}^{(1)}, (([\mathsf{u}'_j(s)]_1)_{j\in[\ell]}, (c_j)_{j\in[\mathsf{n}^*]}), ((\mathsf{u}'_j)_{j\in[\ell]}, (o_j)_{j\in[\mathsf{n}^*]}), 0)$.
4. Output $\pi = ((c'_j)_{j\in[\ell]}, (\mathsf{h}_j)_{j\in[\ell]}, \pi_{\mathsf{link}})$

$\mathsf{Verify}(\mathsf{vk}_{\mathsf{link}}, c, (\hat{c}_j)_{j\in[\ell]}, \pi)$: Parse $\pi = ((c'_j)_{j\in[\ell]}, (\mathsf{h}_j)_{j\in[\ell]}, \pi_{\mathsf{eq}}, \pi_{\mathsf{link}})$ output 1 iff:

1. $\mathsf{Verify}_{\mathsf{eq}}(\mathsf{vk}_{\mathsf{eq}}, (G_i, \boldsymbol{v})_{i\in[\ell]}, (\hat{c}_j)_{j\in[\ell]}, (c'_j)_{j\in[\ell]}, (\mathsf{h}_j)_{j\in[\ell]}, [\mathcal{Z}_V(s)]_1,\ \pi_{\mathsf{eq}}) = 1$.
2. $\mathsf{CP}_{\mathsf{link}}^{(1)}.\overline{\mathsf{Verify}}(\mathsf{vk}_{\mathsf{link}}^{(1)}, ((c'_j)_{j\in[\ell]}, (c_j)_{j\in[\mathsf{n}^*]}), \pi_{\mathsf{link}}) = 1$.

We will use the leaky-zero knowledge of $\mathsf{CP}_{\mathsf{eq}}$. Before stating the theorem we describe the checker C that upon input an index and a value $c \in \mathbb{F}$ outputs 1 if and only if $x \notin \mathbb{H}$. Moreover, we require from $\mathsf{CP}_{\mathsf{eq}}$ to be leaky zero-knowledge only for the input commitments $(c'_j)_{j\in[\ell]}$. As noted in Remark 13 this is the case for our $\mathsf{CP}_{\mathsf{eq}}$.

Theorem 8.4. *If $\mathsf{CP}_{\mathsf{eq}}$ is a CP-SNARK for R_{eq}, and $\mathsf{CP}_{\mathsf{link}}^{(1)}$ is $CP-SNARK$ for $\mathsf{R}_{\mathsf{link}}$ then $\mathsf{CP}_{\mathsf{link}}^{(2)}$ is $(\mathtt{lnk})^{\ell} \times (\mathtt{swh})^{\mathsf{n}^*}$-restricted complete, knowledge-sound. Moreover if $\mathsf{CP}_{\mathsf{eq}}$ is $((0^{\ell}, 1^{\ell}, 0^{\ell}), \mathsf{C})$-leaky zero-knowledge then $\mathsf{CP}_{\mathsf{link}}^{(2)}$ is zero-knowledge. Moreover, if $\mathsf{CP}_{\mathsf{link}}^{(1)}$ and $\mathsf{CP}_{\mathsf{eq}}$ have commitment-only SRSs then $\mathsf{CP}_{\mathsf{link}}^{(2)}$ has commitment-only SRS.*

Proof. We start with completeness. Notice that by definition of $\mathsf{u}'_j(X)$ for $j \in [\ell]$, we have that $\mathsf{u}_j(V) = \mathsf{u}'_j(\theta_j \cdot V)$. Thus, by definition of $h_j(X)$ and by the completeness of $\mathsf{CP}_{\mathsf{eq}}$ the first check of the verifier holds. By definition of $\mathsf{WitExtract}$, for any $j \in [\ell]$ the value $\mathsf{u}'_j(\theta_j \cdot V)$ and $p^*(\theta_j \cdot V)$ agree thus by the completeness of $\mathsf{CP}_{\mathsf{link}}^{(1)}$ the third check of the verifier holds.

We prove knowledge soundness. From the extractor of $\mathsf{CP}_{\mathsf{eq}}$ we can extract the polynomials $(\hat{\mathsf{u}}_j)_{j\in[\ell]}$ with the opening material $(\hat{o}_j)_{j\in[\ell]}$ such that $\hat{\mathsf{u}}_j(V) = \mathsf{u}'_j(\theta_j V)$. By the knowledge soundness of $\mathsf{CP}_{\mathsf{link}}^{(1)}$ we have for all $j \in [\ell]$, $\mathsf{u}_j(V) = \mathsf{u}'_j(\theta_j V) = p^*(\theta_j V)$.

Finally, we prove zero-knowledge. Let $\mathcal{S}_{\mathsf{eq}} = (\mathcal{S}_{\mathsf{leak}}, \mathcal{S})$ be the simulator of $\mathsf{CP}_{\mathsf{eq}}$. The simulator, for any $j \in [\ell]$, sample c'_j, h_j uniformly at random from $\mathbb{G}_1$. Then, let $\hat{\mathsf{x}}_{\mathsf{eq}} = (G_j, \boldsymbol{v})_{j\in[3\ell]}, ((\hat{c}_j)_{j\in[\ell]}, (c'_j)_{j\in[\ell]}, (\mathsf{h}_j)_{j\in[\ell]})$, it runs $\mathcal{S}_{\mathsf{leak}}(1^{\lambda}, \hat{\mathsf{x}}_{\mathsf{eq}})$ of $\mathsf{CS}_{\mathsf{eq}}$ and obtains $\{(j, x_j)\}_{j\in[\ell+1,2\ell]}$. It samples uniformly random value $Leak = (y_j)_{j\in[\ell+1,2\ell]}$ and runs $\mathcal{S}(\mathsf{td}, \hat{\mathsf{x}}_{\mathsf{eq}}, Leak)$ obtaining π_{eq}. Then it simulates the proof $\pi_{\mathsf{link}}^{(1)}$ using the simulator of $\mathsf{CP}_{\mathsf{link}}^{(1)}$.

Through an hybrid argument we can show that the proof is statistically close to a proof where for any $j \in [\ell]$ the value c'_j, h_j are computed as in the real proof and the value y_j is computed as $\mathsf{u}'(x_j)$. Indeed, for any fixed polynomial $\mathsf{u}_j(X)$, the following system of equations hold:

$$\begin{cases} \mathsf{u}'_j(s) = \mathsf{u}_j(s/\theta_j) + \mathcal{Z}_V(s/\theta_j) \cdot \beta_j(s/\theta_j) \\ h_j(s) = (\mathsf{u}(s) - \mathsf{u}'(\theta_j s))/\mathcal{Z}_V(s) = \beta_j(s) \\ y_j = \mathsf{u}_j(x_j/\theta_j) + \mathcal{Z}_V(x_j/\theta_j) \cdot \beta_j(x_j/\theta_j) \end{cases}$$

Recall that $\beta_j(X)$ is an uniformly random degree-2 polynomial thus the tuple $\beta_j(s/\theta_j), \beta_j(s), \beta_j(x_j/\theta_j)$ is uniformly random over $\mathbb{F}^3$ with overwhelming probability (it is not when $x_j/\theta_j \in \{s, s/\theta_j\}$ or when $s \in V$). Therefore they are uniformly distributed as sampled by the simulator.

We can conclude the proof of zero-knowledge through another hybrid step where we switch the simulated proofs $\pi^{(1)}_{\mathsf{link}}$ and π_{eq} with real proof. □

Efficiency. We give efficiency ignoring the costs of $\mathsf{CP}_{\mathsf{opn}}$. For $\mathsf{CP}^{(1)}_{\mathsf{link}}$ generating a proof requires approximately $\ell + 3$ multi-exponentiations with bases of size $|V|$ and $O(\ell|V|)$ $\mathbb{F}$-operations, verification requires $3\ell+3$ pairings. For $\mathsf{CP}^{(2)}_{\mathsf{link}}$ generating a proof requires the computation of $\mathsf{CP}^{(1)}_{\mathsf{link}}$ and additionally 2ℓ multi-exponentiations with bases of size $|V|$ and a proof for $\mathsf{CP}_{\mathsf{eq}}$ that costs approximately 2 multi-exponentiations with bases of size $|V|$.

Extension for the WitExtract of our PHPs for R1CS-lite. We discuss how to support the WitExtract algorithm of our PHPs for R1CS-lite. Let us recall that this algorithm uses the two polynomials $\hat{a}'(X), \hat{b}'(X)$ and computes the witness as $\boldsymbol{w} := (\hat{a}'(\phi_{\mathbb{H}}^{-1}(\ell+1)) \cdot \hat{b}(\phi_{\mathbb{H}}^{-1}(\ell+1)), \dots, \hat{a}'(\phi_{\mathbb{H}}^{-1}(n) \cdot \hat{b}(\phi_{\mathbb{H}}^{-1}(\ell+1)))$.

Given the scheme $\mathsf{CP}^{(2)}_{\mathsf{link}}$ described above, which supports the WitExtract algorithm for a single polynomial, we can obtain a scheme that supports the WitExtract of our PHPs for R1CS-lite with the following simple extension. The prover computes a commitment to a polynomial $\hat{c}'(X)$ such that $\forall \eta \in \mathbb{H} : \hat{c}'(\eta) = \hat{a}'(\eta) \cdot \hat{b}'(\eta)$, proves that this is the case, and then runs the proving algorithm of $\mathsf{CP}^{(2)}_{\mathsf{link}}$ for the single polynomial $\hat{c}'(X)$.

The proof for $\forall \eta \in \mathbb{H} : \hat{c}'(\eta) = \hat{a}'(\eta) \cdot \hat{b}'(\eta)$ can be done by committing to the polynomial $h(X) := (\hat{a}'(X) \cdot \hat{b}'(X) - \hat{c}'(X))/\mathcal{Z}_{\mathbb{H}}(X)$ and then showing that $h(X) \cdot \mathcal{Z}_{\mathbb{H}}(X) - \hat{a}'(X) \cdot \hat{b}'(X) + \hat{c}'(X) = 0$. The latter equation check can be added to the set of equations already proven in $\mathsf{CP}^{(2)}_{\mathsf{link}}$ using $\mathsf{CP}_{\mathsf{eq}}$. The $\mathsf{CP}^{(2)}_{\mathsf{link}}$ CP-SNARK with this extension requires: $2n$ more exponentiations in $\mathbb{G}_1$ to commit to $\hat{c}'(X)$ and $h(X)$, two more commitments, and finally the cost of the evaluation proof used in $\mathsf{CP}_{\mathsf{eq}}$ gets increased: instead of 2 multi-exponentiations with bases of size $|V|$, it is one multi-exponentiation with bases of size $|V|$ and another multi-exponentiation with bases of size n.

9 Our Universal zkSNARKs

We describe different options to obtain universal zkSNARKs in the SRS model by applying our compiler from Section 6 to our PHP constructions of Sections 4.3–4.4 and our CP-SNARKs for pairing-based polynomial commitments of Section 7. The results are a collection of zkSNARKs that offer different tradeoffs in terms of (mainly) SRS size, proof size, and verification time.

9.1 Available Options to Compile Our PHPs

We discuss how to combine some of our CP-SNARKs in section 7 to obtain CP-SNARKs for the $\mathsf{CP}_{\mathsf{php}}$ relation. 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 obtained via the product of

$$\mathsf{R}_{\mathsf{deg}}((d_j)_{j\in[\mathsf{n_p}]}, (p_j)_{j\in[\mathsf{n}(0)+1,\mathsf{n}^*]}) \ \wedge\ \mathsf{R}_{\mathsf{eq}}((G'_1, \boldsymbol{v}_1), (p_j)_{j\in[\mathsf{n}^*]}) \ \wedge\ \mathsf{R}_{\mathsf{eq}}((G'_2, \boldsymbol{v}_2), (p_j)_{j\in[\mathsf{n}^*]})$$

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 the specialization $\mathsf{R}_{\mathsf{qeq}}(G'_2, (p_j)_{j\in[\mathsf{n}^*]})$ introduced in Section 7.7.

Given the above considerations, we consider two main options for applying our compiler to our PHPs:

Commitment scheme CS_1: this is applied to $\mathsf{PHP}_{\mathsf{lite1x}}$, $\mathsf{PHP}_{\mathsf{lite2x}}$, $\mathsf{PHP}_{\mathsf{r1cs1x}}$, and $\mathsf{PHP}_{\mathsf{r1cs2x}}$.

- For $\mathsf{CP}_{\mathsf{opn}}$ we can use either $\mathsf{CP}^{\mathsf{AGM}}_{\mathsf{opn}}$, secure in the algebraic group model, or $\mathsf{CP}^{\mathsf{PKE}}_{\mathsf{opn}}$ that relies on the mPKE assumption (see Section 7.3).
- To prove the first and second polynomial checks we use (twice) $\mathsf{CP}_{\mathsf{eq}}$ of Section 7.6.
- To prove $\mathsf{R}_{\mathsf{deg}}$, we use $\mathsf{CP}^{(\star)}_{\mathsf{deg}}$ of Section 7.8, with the optimization of Remark 14.

Commitment scheme CS_2: this is applied to $\mathsf{PHP}_{\mathsf{lite1}}$, $\mathsf{PHP}_{\mathsf{lite2}}$, $\mathsf{PHP}_{\mathsf{r1cs1}}$, and $\mathsf{PHP}_{\mathsf{r1cs2}}$.

- For $\mathsf{CP}_{\mathsf{opn}}$ we can use either $\mathsf{CP}^{\mathsf{AGM}}_{\mathsf{opn}}$, secure in the algebraic group model, or $\mathsf{CP}^{\mathsf{PKE}}_{\mathsf{opn}}$ that relies on the mPKE assumption (see Section 7.3).
- To prove $\mathsf{R}_{\mathsf{deg}}$, we use $\mathsf{CP}^{(2)}_{\mathsf{deg}}$ of Section 7.8.2
- To prove the first polynomial check we use $\mathsf{CP}_{\mathsf{eq}}$ of Section 7.6.
- To prove the second polynomial check we use $\mathsf{CP}_{\mathsf{qeq}}$ of Section 7.7.

9.2 Instantiating the PHPs with the appropriate zero-knowledge bounds

Our compiler accounts for using a CP-SNARK $\mathsf{CP}_{\mathsf{php}}$ that can be (b, C)-leaky-ZK, which in turn requires the PHP protocol to be $(\mathbf{1} + \mathsf{b})$-bounded ZK (see Theorem 6.1).[21]

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 of Section 7.6. Its leaky-ZK 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 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 what is b for the b-leaky-ZK of $\mathsf{CP}_{\mathsf{eq}}$ when it is used to prove the first polynomials checks in all our PHPs, and how such b impacts the instantiation of these PHPs.

PHPs for R1CS-lite. The first polynomial check is the same in both constructions, and for the sake of the relation R_{eq} the polynomial G'_1 can be written as

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

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

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

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 $\mathsf{b} = (\mathsf{b}_a, \mathsf{b}_b, \mathsf{b}_s, \mathsf{b}_q, \mathsf{b}_r) = (0, 1, 0, 0, 0)$ (cf. Remark 13).

From Theorem 6.1, $\mathsf{PHP}_{\mathsf{lite1}}$ and $\mathsf{PHP}_{\mathsf{lite2}}$ need to be $(1, 2, 1, 1, 1)$-bounded ZK. Moreover, note that all the "$+\mathbf{1}$" evaluations due to the commitment are all in the same point (the secret exponent s). This is relevant because, by Remark 1, we can optimize the degrees and instantiate $\mathsf{PHP}_{\mathsf{lite1}}$ and $\mathsf{PHP}_{\mathsf{lite2}}$ with $\hat{a}' \in \mathbb{F}_{\leq n+1}[X]$, $\hat{b}' \in \mathbb{F}_{\leq n+2}[X]$, $q_s \in \mathbb{F}_{\leq 1}[X]$, $r_s \in \mathbb{F}_{\leq 1}[X]$.

[21] The $+\mathbf{1}$ essentially comes from the fact that the commitment reveals one evaluation of each oracle polynomial.

PHPs for R1CS. All constructions share the same first polynomial check, which can be written as

$$G_1'(X_a, X_b, X_w, X_s, X_q, X_r) := X_a \cdot X_b \cdot g_{a,b} + X_a \cdot g_a + X_b \cdot g_b + X_s + X_q \cdot g_q + X_r \cdot g_r + g_0$$

and whose goal is to prove that on a given y, $G_1'((p_j(y))_{j\in[6]}) = 0$, i.e.,

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

Similarly to the above, $\mathsf{CP_{eq}}$ chooses a set S of size 1, revealing only the evaluation of $\hat{b}(y)$. Thus, $\mathsf{CP_{eq}}$ for G_1 is b−leaky-ZK with $\mathsf{b} = (\mathsf{b}_a, \mathsf{b}_b, \mathsf{b}_w, \mathsf{b}_s, \mathsf{b}_q, \mathsf{b}_r) = (0, 1, 0, 0, 0, 0)$. Due to Theorem 6.1, these constructions need to be $(1, 2, 1, 1, 1, 1)$-bounded ZK, where the $+\mathbf{1}$ evaluations correspond to the evaluation of the secret exponent of the commitments. Similarly to the previous case, the optimizations of Remark 4 apply to these PHPs as well.

9.3 Our zkSNARKs

In Table 4 we summarize the efficiency of the zkSNARKs schemes obtained through the different options to instantiate the compiler on all our PHPs (the table only shows the instantiation in the AGM model, see later for the differences when $\mathsf{CP_{opn}} = \mathsf{CP^{PKE}_{opn}}$). We comment how these measures are computed. The final numbers are obtained after considering the efficiency of the single CP-SNARKs from Section 7.

- The universal SRS srs is the commitment key instantiated using the maximal degree D of the given PHP, and the KeyGen cost is the cost of generating this commitment key. This follows from the fact that all the CP-SNARKs used in this instantiation are commitment-only.
- The verification key $\mathsf{vk_R}$ of the specialized SRS $\mathsf{srs_R}$ for an R1CS-lite (resp. R1CS) relation involving matrices of dimension n and density at most m includes $\mathtt{rel}$-type commitments to the relation polynomials and the specialized SRSs for the CP-SNARKs. In our case, the latter only includes $[s]_2$ used to verify a proof in $\mathsf{CP_{eval}}$, and $[s, s^{\mathsf{D}-n+2}, s^{\mathsf{D}-m+2}]_2$ used in the verification of $\mathsf{CP^{(2)}_{deg}}$ when using CS_2. The Derive complexity is the cost of generating these $\mathtt{rel}$-type commitments.
- The proof includes one commitment per polynomial sent by the PHP prover, one $\mathsf{CP_{opn}}$ proof per PHP round, two $\mathsf{CP_{deg}}$ proofs, one $\mathsf{CP_{eq}}$ proof for the first polynomial check, and a proof for the second polynomial check, which is done using $\mathsf{CP_{eq}}$ for CS_1 or using $\mathsf{CP_{qeq}}$ for CS_2. The cost of the prover is the sum of: the committing cost which corresponds to the PHP proof length (translated into $\mathbb{G}_1$ exponentiations), the cost of generating the CP-SNARK proofs, and the PHP prover complexity (which are $\mathbb{F}$ operations). Note that in the CS_2 instantiation, the $\mathsf{CP_{qeq}}$ proof is empty since for every quadratic term of G_2' we have exactly one commitment in $\mathbb{G}_1$ and another in $\mathbb{G}_2$.
- Verification involves running the PHP verifier, $D_{\mathcal{V}}$, and to run verification of the CP-SNARK proofs for $\mathsf{CP_{opn}}$, $\mathsf{CP_{deg}}$, $\mathsf{CP_{eq}}$ for the first polynomial check, and $\mathsf{CP_{eq}}$ (resp. $\mathsf{CP_{qeq}}$) for the second check in the CS_1 (resp. CS_2) instantiation. In our summary we only count the number of pairings, as this is the most expensive cost. Each proof of $\mathsf{CP_{opn}}$, $\mathsf{CP_{deg}}$ and $\mathsf{CP_{eq}}$ requires 2 pairings while a $\mathsf{CP_{qeq}}$ proof (for the G polynomial used in our case) needs 3 pairings. Several of these pairings have a common $\mathbb{G}_2$ argument, and thus can be batched using standard techniques; the numbers in the table are the ones after batching.

In Table 1 we present a comparison between a selection of our schemes and previous work.

Instantiations under mPKE. For the versions of our zkSNARKs based on the mPKE assumption, i.e., instantiated with $\mathsf{CP_{opn}} = \mathsf{CP^{PKE}_{opn}}$, the efficiency decreases as follows: SRS[22] size is increased by D elements of $\mathbb{G}_1$ and 1 element of $\mathbb{G}_2$, proof size is increased by 4 elements of $\mathbb{G}_1$, the verifier needs 1 more pairing (after batching), and the prover needs l more exponentiations in $\mathbb{G}_1$, where $l = 3n + 2m$ for $\Pi^{(1)}_{\mathsf{lite2x}}, \Pi^{(2)}_{\mathsf{lite2}}, \Pi^{(1)}_{\mathsf{r1cs2x}}, \Pi^{(2)}_{\mathsf{r1cs2}}$, $l = 3n + 4m$ for $\Pi^{(1)}_{\mathsf{lite1x}}, \Pi^{(2)}_{\mathsf{lite1}}, \Pi^{(1)}_{\mathsf{r1cs3}}$, and $l = 3n + 6m$ for $\Pi^{(1)}_{\mathsf{r1cs1x}}, \Pi^{(2)}_{\mathsf{r1cs1}}$.

[22] Note that by remark 11, the SRS is still updatable.

It is worth noting that all our instantiations under the mPKE assumption are significantly more efficient than the instantiation of Marlin [CHM+20] with the polynomial commitments based on mPKE. The latter would require 11 more elements of $\mathbb{G}_1$ in the proof (1 per commitment), while the proving time requires $11n + 5m$ more exponentiations in $\mathbb{G}_1$.

	PHP	CS	Π		size $\lvert\mathsf{srs}\rvert$	size $\lvert\mathsf{vk_R}\rvert$	size $\lvert\pi\rvert$	time KeyGen	time Derive	time Prove	time Verify
R1CS-lite	$\mathsf{PHP_{lite1x}}$	CS_1	$\Pi^{(1)}_{\mathsf{lite1x}}$	$\mathbb{G}_1$	$2M$	5	11	$2M$	$10m$	$8n{+}8m{-}2\ell$	2 pairings
				$\mathbb{G}_2$	1	1	—	1	—	—	2 pairings
				$\mathbb{F}$	—	—	3	—	$O(m\log m)$	$O(m\log m)$	$O(\ell{+}\log m)$
	$\mathsf{PHP_{lite1}}$	CS_2	$\Pi^{(2)}_{\mathsf{lite1}}$	$\mathbb{G}_1$	$2M$	—	10	$2M$	—	$8n{+}6m{-}2\ell$	7 pairings
				$\mathbb{G}_2$	$2M$	11	—	$2M$	$16m$	—	7 pairings
				$\mathbb{F}$	—	—	2	—	$O(m\log m)$	$O(m\log m)$	$O(\ell{+}\log m)$
	$\mathsf{PHP_{lite2x}}$	CS_1	$\Pi^{(1)}_{\mathsf{lite2x}}$	$\mathbb{G}_1$	M	16	11	M	$16m$	$8n{+}4m{-}2\ell$	2 pairings
				$\mathbb{G}_2$	1	1	—	1	—	—	2 pairings
				$\mathbb{F}$	—	—	3	—	$O(m\log m)$		$O(\ell{+}\log m)$
	$\mathsf{PHP_{lite2}}$	CS_2	$\Pi^{(2)}_{\mathsf{lite2}}$	$\mathbb{G}_1$	M	—	10	M	—	$8n{+}3m{-}2\ell$	7 pairings
				$\mathbb{G}_2$	M	27	—	M	$24m$	—	7 pairings
			(aka LunarLite)	$\mathbb{F}$	—	—	2	—	$O(m\log m)$	$O(m\log m)$	$O(\ell{+}\log m)$
R1CS	$\mathsf{PHP_{r1cs1x}}$	CS_1	$\Pi^{(1)}_{\mathsf{r1cs1x}}$	$\mathbb{G}_1$	$3M$	6	12	$3M$	$18m$	$9n{+}12m{-}\ell$	2 pairings
				$\mathbb{G}_2$	1	1	—	1	—	—	2 pairings
				$\mathbb{F}$	—	—	3	—	$O(m\log m)$	$O(m\log m)$	$O(\ell{+}\log m)$
	$\mathsf{PHP_{r1cs1}}$	CS_2	$\Pi^{(2)}_{\mathsf{r1cs1}}$	$\mathbb{G}_1$	$3M$	—	11	$3M$	—	$9n{+}9m{-}\ell$	7 pairings
				$\mathbb{G}_2$	$3M$	12	—	$3M$	$27m$	—	7 pairings
				$\mathbb{F}$	—	—	2	—	$O(m\log m)$	$O(m\log m)$	$O(\ell{+}\log m)$
	$\mathsf{PHP_{r1cs2x}}$	CS_1	$\Pi^{(1)}_{\mathsf{r1cs2x}}$	$\mathbb{G}_1$	M	42	12	M	$42m$	$9n{+}4m{-}\ell$	2 pairings
				$\mathbb{G}_2$	1	1	—	1	—	—	2 pairings
				$\mathbb{F}$	—	—	3	—	$O(m\log m)$		$O(\ell{+}\log m)$
	$\mathsf{PHP_{r1cs2}}$	CS_2	$\Pi^{(2)}_{\mathsf{r1cs2}}$	$\mathbb{G}_1$	M	—	11	M	—	$9n{+}3m{-}\ell$	7 pairings
			(aka Lunar1cs	$\mathbb{G}_2$	M	60	—	M	$57m$	—	7 pairings
			fast & short)	$\mathbb{F}$	—	—	2	—	$O(m\log m)$	$O(m\log m)$	$O(\ell{+}\log m)$
	$\mathsf{PHP_{r1cs3}}$	CS_1	$\Pi^{(1)}_{\mathsf{r1cs3}}$	$\mathbb{G}_1$	$3M$	12	12	$3M$	$12m$	$9n{+}8m{-}\ell$	2 pairings
			(aka Lunar1cs	$\mathbb{G}_2$	1	1	—	1	—	—	2 pairings
			short vk)	$\mathbb{F}$	—	—	5	—	$O(m\log m)$	$O(m\log m)$	$O(\ell{+}\log m)$

Table 4: Efficiency summary of our zkSNARKs with universal and updatable SRS in the AGM model (i.e., using $\mathsf{CP_{opn}} = \mathsf{CP^{AGM}_{opn}}$) for R1CS-lite and R1CS relations with $n \times n$ matrices, each of density $\leq m$, and inputs of length ℓ. For field operations, we simplified using that $m = O(n)$. M is the largest value of m supported by the PHPs.

9.4 Our CP-SNARKs

By using the commit-and-prove variant of our compiler described in Section 8.1, we obtain commit-and-prove variants of our zkSNARKs in Table 4. Below we discuss their efficiency.

Let us consider proving R1CS or R1CS-lite relations in which a portion of the witness vector w is committed. Assume there are l commitments, $(\hat{c}_j)_{j\in[l]}$, such that each $\hat{c}_i$ commits to a vector of dimension v encoded in a low-degree extension $\mathsf{u}_i(X)$ of degree $v-1$. Also, we recall that according to our compiler, each CP-SNARK variant works the same as the corresponding zkSNARK except that it additionally runs the $\mathsf{CP^{(2)}_{link}}$ proof system.

In the case of the PHPs for R1CS, adding the $\mathsf{CP^{(2)}_{link}}$ proof requires in addition: $n + v(3l+2) + l$ exponentiations in $\mathbb{G}_1$ for the prover, $(4l+2)$ elements of $\mathbb{G}_1$ and one element of $\mathbb{F}$ in the proof, and

$l+3$ pairings to the verifier.

In the case of the PHPs for R1CS-lite, adding the $\mathsf{CP}_{\mathsf{link}}^{(2)}$ proof (with the modification to deal with the two polynomials) requires in addition: $4n+v(3l+1)+l$ exponentiations in $\mathbb{G}_1$ for the prover, $(4l+4)$ elements of $\mathbb{G}_1$ and two elements of $\mathbb{F}$ in the proof, and $l+3$ pairings to the verifier.

Acknowledgements

Research leading to these results has been partially supported 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 a research gift from Protocol Labs. The second author was partially supported by the before mentioned projects when he was a postdoctoral fellow at IMDEA Software Institute where he performed part of the research leading to this paper. 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

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73. Springer, Heidelberg, May 2004.

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

[BBC+17] Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. Computational integrity with a public random string from quasi-linear PCPs. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 551–579. Springer, Heidelberg, April / May 2017.

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

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

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

[BCFK19] Daniel Benarroch, Matteo Campanelli, Dario Fiore, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. Cryptology ePrint Archive, Report 2019/1255, 2019. `https://eprint.iacr.org/2019/1255`.

[BCG+15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304. IEEE Computer Society Press, May 2015.

[BCG+19] Eli Ben-Sasson, Alessandro Chiesa, Lior Goldberg, Tom Gur, Michael Riabzev, and Nicholas Spooner. Linear-size constant-query IOPs for delegating computation. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 494–521. Springer, Heidelberg, December 2019.

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

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

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

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

[BGG+90] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Hastad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 37–56. Springer, Heidelberg, August 1990.

[BGG19] Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *FC 2018 Workshops*, volume 10958 of *LNCS*, pages 64–77. Springer, Heidelberg, March 2019.

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

[CD98] Ronald Cramer and Ivan Damgard. Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 424–441. Springer, Heidelberg, August 1998.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019.

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

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

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[DRZ20] Vanesa Daza, Carla Ràfols, and Alexandros Zacharakis. Updateable inner product argument with logarithmic verifier and applications. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 527–557. Springer, Heidelberg, May 2020.

[EG14] Alex Escala and Jens Groth. Fine-tuning Groth-Sahai proofs. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 630–649. Springer, Heidelberg, March 2014.

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Heidelberg, August 2013.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

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

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

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

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 113–122. ACM Press, May 2008.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, December 1994.

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

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

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

[IS90] K. G. Ivanov and E. B. Saff. *Behavior of the Lagrange Interpolants in the Roots of Unity*, pages 81–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.

[Ish19] Yuval Ishai. Efficient zero-knowledge proofs: A modular approach. `https://simons.berkeley.edu/talks/tbd-79`. Also see `https://zkproof.org/2020/08/12/information-theoretic-proof-systems/`, 2019.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

[KPPS20] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, and Dawn Song. MIRAGE: Succinct arguments for randomized algorithms with applications to universal zk-SNARKs. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2129–2146. USENIX Association, August 2020.

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

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

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

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 49–62. ACM Press, June 2016.

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

[TB04] Lloyd Trefethen and Jean-Paul Berrut. Barycentric lagrange interpolation. *SIAM Review*, 46(3):501–517, 2004.

[WTs+18] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018.

[WZC+18] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 675–692. USENIX Association, August 2018.

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 733–764. Springer, Heidelberg, August 2019.

[ZGK+17a] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *2017 IEEE Symposium on Security and Privacy*, pages 863–880. IEEE Computer Society Press, May 2017.

[ZGK+17b] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. A Zero-Knowledge Version of vSQL. Cryptology ePrint Archive, Report 2017/1146, 2017. `https://eprint.iacr.org/2017/1146`.

[ZGK+18] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vRAM: Faster verifiable RAM with program-independent preprocessing. In *2018 IEEE Symposium on Security and Privacy*, pages 908–925. IEEE Computer Society Press, May 2018.

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy*, pages 859–876. IEEE Computer Society Press, May 2020.

A Constraint Systems

Below we define a less compact version of R1CS-lite (Definition 4.5) in which satisfiability is expressed via three vectors and three checks. This form is sometimes more convenient work with.

Definition A.1 (LongR1CS-lite)**.** *Let $\mathbb{F}$ be a finite field and $n, m \in \mathbb{N}$ be positive integers. The universal relation $\mathcal{R}_{LongR1CS\text{-}lite}$ is the set of triples*

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

where $\boldsymbol{L}, \boldsymbol{R} \in \mathbb{F}^{n\times n}$, $\max\{||\boldsymbol{L}||, ||\boldsymbol{R}||\} \le m$, $\boldsymbol{x} \in \mathbb{F}^{\ell-1}$, $\boldsymbol{a'}, \boldsymbol{b'}, \boldsymbol{c'} \in \mathbb{F}^{n-\ell}$, and for $\boldsymbol{a} := (1, \boldsymbol{x}, \boldsymbol{a'})$, $\boldsymbol{b} := (\boldsymbol{1}, \boldsymbol{b'})$, $\boldsymbol{c} := (1, \boldsymbol{x}, \boldsymbol{c'})$ it holds

$$\boldsymbol{a} \circ \boldsymbol{b} - \boldsymbol{c} = 0 \quad \wedge \quad \boldsymbol{a} + \boldsymbol{L} \cdot \boldsymbol{c} = 0 \quad \wedge \quad \boldsymbol{b} + \boldsymbol{R} \cdot \boldsymbol{c} = 0$$

Lemma A.1. *Let R (resp. $\hat{\mathsf{R}}$) be a LongR1CS-lite (resp. R1CS-lite) relation with matrices $\{\boldsymbol{L}, \boldsymbol{R}\}$. Then for any $\boldsymbol{x} \in \mathbb{F}^{\ell-1}$, it holds $\boldsymbol{x} \in \mathcal{L}(\mathsf{R})$ if and only if $\boldsymbol{x} \in \mathcal{L}(\hat{\mathsf{R}})$.*

Proof. Case I: $\boldsymbol{x} \in \mathcal{L}(\mathsf{R}) \Rightarrow \boldsymbol{x} \in \mathcal{L}(\hat{\mathsf{R}})$. Let $(\boldsymbol{a'}, \boldsymbol{b'}, \boldsymbol{c'})$ be a witness for $\boldsymbol{x} \in \mathcal{L}(\mathsf{R})$. Then $\hat{\boldsymbol{c'}} := \boldsymbol{a'} \circ \boldsymbol{b'}$ is a witness for $\boldsymbol{x} \in \mathcal{L}(\hat{\mathsf{R}})$. To see this, note that by LongR1CS-lite definition, we have that $\boldsymbol{L} \cdot (\boldsymbol{a} \circ \boldsymbol{b}) \circ \boldsymbol{R} \cdot (\boldsymbol{a} \circ \boldsymbol{b}) = \boldsymbol{a} \circ \boldsymbol{b}$, for $\boldsymbol{a} := (1, \boldsymbol{x}, \boldsymbol{a'})$ and $\boldsymbol{b} := (\boldsymbol{1}, \boldsymbol{b'})$. Finally, noticing that $(1, \boldsymbol{x}, \hat{\boldsymbol{c'}}) = \boldsymbol{a} \circ \boldsymbol{b}$ concludes this part of the proof.

Case II: $\boldsymbol{x} \in \mathcal{L}(\mathsf{R}) \Leftarrow \boldsymbol{x} \in \mathcal{L}(\hat{\mathsf{R}})$. Let $\hat{\boldsymbol{c'}}$ be a witness for $\boldsymbol{x} \in \mathcal{L}(\hat{\mathsf{R}})$, namely for $\hat{\boldsymbol{c}} := (1, \boldsymbol{x}, \hat{\boldsymbol{c'}})$ it holds $\hat{\boldsymbol{c}} = \boldsymbol{L} \cdot \hat{\boldsymbol{c}} \circ \boldsymbol{R} \cdot \hat{\boldsymbol{c}}$.

Let $\tilde{\boldsymbol{a}} := -\boldsymbol{L} \cdot \hat{\boldsymbol{c}}$, $\tilde{\boldsymbol{b}} := -\boldsymbol{R} \cdot \hat{\boldsymbol{c}}$, and $\boldsymbol{c'} := \hat{\boldsymbol{c'}}$, and let $\boldsymbol{a'}, \boldsymbol{b'}$ be the last $n - \ell$ rows of $\tilde{\boldsymbol{a}}$ and $\tilde{\boldsymbol{b}}$ respectively. By the satisfiability of R1CS-lite we have that

$$\begin{pmatrix} 1 \\ \boldsymbol{x} \\ \boldsymbol{c'} \end{pmatrix} = \hat{\boldsymbol{c}} = \boldsymbol{L} \cdot \hat{\boldsymbol{c}} \circ \boldsymbol{R} \cdot \hat{\boldsymbol{c}} = \tilde{\boldsymbol{a}} \circ \tilde{\boldsymbol{b}} = \begin{pmatrix} \boldsymbol{a''} \\ \boldsymbol{a'} \end{pmatrix} \circ \begin{pmatrix} \boldsymbol{b''} \\ \boldsymbol{b'} \end{pmatrix}$$

which implies that $\boldsymbol{c'} = \boldsymbol{a'} \circ \boldsymbol{b'}$, and thus for $\boldsymbol{a} := (1, \boldsymbol{x}, \boldsymbol{a'})$, $\boldsymbol{b} := (\boldsymbol{1}, \boldsymbol{b'})$, and $\boldsymbol{c} := (1, \boldsymbol{x}, \boldsymbol{c'})$, the Hadamard constraint of R1CS-lite must hold.

Finally, note that from the definition of the first ℓ rows of $\boldsymbol{R}$ it holds $\tilde{\boldsymbol{b}} = (\boldsymbol{1}, \boldsymbol{b'})$, and thus $\tilde{\boldsymbol{a}} = (1, \boldsymbol{x}, \boldsymbol{a'})$. Therefore, for $\boldsymbol{a}, \boldsymbol{b}$ as above the linear constraints of R1CS-lite are also satisfied.

This concludes the proof that $(\boldsymbol{a'}, \boldsymbol{b'}, \boldsymbol{c'})$ is a satisfying witness for $\boldsymbol{x} \in \mathcal{L}(\mathsf{R})$. □

A.1 Proof of Lemma 4.5

Proof. We do the proof by showing equivalence with the LongR1CS-lite relation of definition A.1; by lemma A.1 one can then obtain the proof.

First, we claim that equation (1) is equivalent to

$$\forall \eta \in \mathbb{H} : a(\eta) + \sum_{\eta' \in \mathbb{H}} \boldsymbol{L}_{\eta,\eta'} \cdot a(\eta') \cdot b(\eta') \quad = 0 \tag{17}$$

$$\forall \eta \in \mathbb{H} : b(\eta) + \sum_{\eta' \in \mathbb{H}} \boldsymbol{R}_{\eta,\eta'} \cdot a(\eta') \cdot b(\eta') \quad = 0 \tag{18}$$

To see this, we observe that we can group the checks inside equations (17) (resp. (18)) by doing a linear combination with linearly independent polynomials $\{\mathcal{L}_\eta^{\mathbb{H}}(X)\}_{\eta\in\mathbb{H}}$; then, these two equations can be merged into a single one by introducing a new random variable Z.

$$\sum_{\eta\in\mathbb{H}} a(\eta)\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) + \sum_{\eta,\eta'\in\mathbb{H}} \boldsymbol{L}_{\eta,\eta'}\cdot a(\eta')\cdot b(\eta')\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) = 0 \in \mathbb{F}[X]$$
$$\sum_{\eta\in\mathbb{H}} b(\eta)\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) + \sum_{\eta,\eta'\in\mathbb{H}} \boldsymbol{R}_{\eta,\eta'}\cdot a(\eta')\cdot b(\eta')\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) = 0 \in \mathbb{F}[X]$$
$$\Updownarrow$$
$$a(X) + Z\cdot b(X) + \sum_{\eta,\eta'\in\mathbb{H}} \boldsymbol{L}_{\eta,\eta'}\cdot a(\eta')\cdot b(\eta')\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) + Z\sum_{\eta,\eta'\in\mathbb{H}} \boldsymbol{R}_{\eta,\eta'}\cdot a(\eta')\cdot b(\eta')\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) = 0 \in \mathbb{F}[X,Z]$$

In one direction, given a satisfying witness $(\boldsymbol{a'},\boldsymbol{b'},\boldsymbol{c'})$ for R1CS-lite, we can build polynomials $a'(X), b'(X)$ that satisfy equations (17)–(18) by defining: $\boldsymbol{a} := (1,\boldsymbol{x},\boldsymbol{a'})$, $a(X) = \sum_{\eta\in\mathbb{H}} \boldsymbol{a}_{\phi_{\mathbb{H}}(\eta)}\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) \in \mathbb{F}_{\leq n-1}[X]$ (and similarly for $b(X)$ from $\boldsymbol{b} = (\boldsymbol{1},\boldsymbol{b'})$), $a'(X) := (a(X) - \sum_{\eta\in\mathbb{L}} \boldsymbol{x}_{\phi_{\mathbb{H}}(\eta)}\cdot\mathcal{L}_\eta^{\mathbb{L}}(X))/\mathcal{Z}_{\mathbb{L}}(X)$, and $b'(X) := (b(X)-1)/\mathcal{Z}_{\mathbb{L}}(X)$.

In the other direction, let $a'(X), b'(X)$ be polynomials such that their extensions with the public input satisfy equations (17)–(18). Then, we build a witness $(\boldsymbol{a'},\boldsymbol{b'},\boldsymbol{c'})$ for R1CS-lite by defining: $\boldsymbol{b'} := (b(\phi_{\mathbb{H}}^{-1}(\ell+1)),\ldots,b(\phi_{\mathbb{H}}^{-1}(n))$, $\boldsymbol{a'} := (a'(\phi_{\mathbb{H}}^{-1}(\ell+1)),\ldots,a'(\phi_{\mathbb{H}}^{-1}(n))$, and $\boldsymbol{c'} := (a'(\phi_{\mathbb{H}}^{-1}(\ell+1))\cdot b(\phi_{\mathbb{H}}^{-1}(\ell+1)),\ldots,a'(\phi_{\mathbb{H}}^{-1}(n)\cdot b(\phi_{\mathbb{H}}^{-1}(\ell+1)))$. □

A.2 Proof of Lemma 4.8

Proof. Similarly to the proof above, we claim that equation (11) is equivalent to

$$\forall\eta\in\mathbb{H}: a(\eta) + \sum_{\eta'\in\mathbb{H}} \boldsymbol{L}_{\eta,\eta'}\cdot z(\eta') = 0 \quad (19)$$
$$\forall\eta\in\mathbb{H}: b(\eta) + \sum_{\eta'\in\mathbb{H}} \boldsymbol{R}_{\eta,\eta'}\cdot z(\eta') = 0 \quad (20)$$
$$\forall\eta\in\mathbb{H}: -a(\eta)\cdot b(\eta) + \sum_{\eta'\in\mathbb{H}} \boldsymbol{O}_{\eta,\eta'}\cdot z(\eta') = 0 \quad (21)$$

To see this, we observe that we can group the checks inside equations (19)–(21) by doing a linear combination with linearly independent polynomials $\{\mathcal{L}_\eta^{\mathbb{H}}(X)\}_{\eta\in\mathbb{H}}$; then, these three equations can be merged into a single one by introducing new random variables Z_L, Z_R, Z_O.

$$\sum_{\eta\in\mathbb{H}} a(\eta)\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) + \sum_{\eta,\eta'\in\mathbb{H}} \boldsymbol{L}_{\eta,\eta'}\cdot z(\eta')\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) = 0 \in \mathbb{F}[X]$$
$$\sum_{\eta\in\mathbb{H}} b(\eta)\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) + \sum_{\eta,\eta'\in\mathbb{H}} \boldsymbol{R}_{\eta,\eta'}\cdot z(\eta')\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) = 0 \in \mathbb{F}[X]$$
$$-\sum_{\eta\in\mathbb{H}} a(\eta)\cdot b(\eta)\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) + \sum_{\eta,\eta'\in\mathbb{H}} \boldsymbol{O}_{\eta,\eta'}\cdot z(\eta')\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) = 0 \in \mathbb{F}[X]$$
$$\Updownarrow$$
$$\sum_{\eta\in\mathbb{H}} \big(Z_L a(\eta) + Z_R b(\eta) - Z_O a(\eta) b(\eta)\big)\mathcal{L}_\eta^{\mathbb{H}}(X) + \sum_{\substack{\eta,\eta'\in\mathbb{H}\\ M\in\{L,R,O\}}} Z_M \boldsymbol{M}_{\eta,\eta'} z(\eta')\mathcal{L}_\eta^{\mathbb{H}}(X) = 0 \in \mathbb{F}[X, Z_L, Z_R, Z_O]$$

In one direction, given a satisfying witness $\boldsymbol{w}$ for R1CS, we can build polynomials $a(X), b(X), w(X)$ that satisfy equations (19)–(21): first, let $\boldsymbol{a} = -\boldsymbol{L}\cdot\boldsymbol{z}$ and $\boldsymbol{b} = -\boldsymbol{R}\cdot\boldsymbol{z}$ with $\boldsymbol{z} = (1,\boldsymbol{x},\boldsymbol{w})$, second, define by interpolation $a(X) = \sum_{\eta\in\mathbb{H}} \boldsymbol{a}_{\phi_{\mathbb{H}}(\eta)}\cdot\mathcal{L}_\eta^{\mathbb{H}}(X) \in \mathbb{F}_{\leq n-1}[X]$ and similarly $b(X)$, and finally define $z(X) := \sum_{\eta\in\mathbb{L}} \boldsymbol{x'}_{\phi_{\mathbb{H}}(\eta)}\cdot\mathcal{L}_\eta^{\mathbb{L}}(X) + \sum_{\eta\in\mathbb{H}\setminus\mathbb{L}} \boldsymbol{w}_{\phi_{\mathbb{H}}(\eta)}\cdot\mathcal{L}_\eta^{\mathbb{H}\setminus\mathbb{L}}(X)\cdot\mathcal{Z}_{\mathbb{L}}(X)$.

In the other direction, let $a(X), b(X), w(X)$ be such that they satisfy equations (19)–(21), for $z(X) := \sum_{\eta\in\mathbb{L}} \boldsymbol{x'}_{\phi_{\mathbb{H}}(\eta)}\cdot\mathcal{L}_\eta^{\mathbb{L}}(X) + w(X)\cdot\mathcal{Z}_{\mathbb{L}}(X)$. Then, we build a witness $\boldsymbol{w}$ for R1CS by defining $\boldsymbol{w} := (z(\phi_{\mathbb{H}}^{-1}(\ell+1)),\ldots,z(\phi_{\mathbb{H}}^{-1}(n)))$. □

A.3 Reduction to Arithmetic Circuit Satisfiability

Let us recall the arithmetic circuit satisfiability problem.

Definition A.2 (Arithmetic Circuit Satisfiability)**.** *Let $\mathbb{F}$ be a finite field. The universal arithmetic circuit satisfiability relation $\mathcal{R}_{\mathcal{C}}$ is the set of triples $(C, (\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{w})$, where $C : \mathbb{F}^{\ell_{in}} \times \mathbb{F}^{\ell_{wit}} \to \mathbb{F}^{\ell_{out}}$ is an arithmetic circuit with ℓ_{in} public inputs, ℓ_{wit} private inputs, and ℓ_{out} public outputs, such that $C(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{y}$.*

Gennaro et al. [GGPR13] proved how to encode arithmetic circuit satisfiability with a quadratic arithmetic program (which is the polynomial version of R1CS). We summarize below their result.

Theorem A.2 ([GGPR13])**.** *Let $C : \mathbb{F}^{\ell_{in}} \times \mathbb{F}^{\ell_{wit}} \to \mathbb{F}^{\ell_{out}}$ be an arithmetic circuit with N multiplication gates. Then there exists an R1CS $\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O} \in \mathbb{F}^{(N+\ell_{out})\times n}$ with $n = \ell_{in} + \ell_{out} + N + 1$ such that for any $\boldsymbol{x}$, $\exists \boldsymbol{w} : C(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{y}$ if and only if $\exists \boldsymbol{c'}$ that makes $(1, \boldsymbol{x}, \boldsymbol{y})$ accepted by $(\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O})$.*

In the following theorem we show a similar method to encode arithmetic circuit satisfiability with R1CS-lite.

Theorem A.3. *Let $C : \mathbb{F}^{\ell_{in}} \times \mathbb{F}^{\ell_{wit}} \to \mathbb{F}^{\ell_{out}}$ be an arithmetic circuit with N multiplication gates. Then there exists an R1CS-lite $\{\boldsymbol{L}, \boldsymbol{R}\} \in \mathbb{F}^{n\times n}$ with $\ell = \ell_{in} + \ell_{out} + 1$, $n = \ell + N$, such that for any $\boldsymbol{x}$, $\exists \boldsymbol{w} : C(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{y}$ if and only if $\exists \boldsymbol{w'}$ that makes $(1, \boldsymbol{x}, \boldsymbol{y})$ accepted by $\{\boldsymbol{L}, \boldsymbol{R}\}$.*

Proof. We do the proof by building matrices for the LongR1CS-lite relation. By the equivalence of R1CS-lite LongR1CS-lite shown in lemma A.1, this shows a reduction to R1CS-lite.

Our goal is to define $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ and matrices $\boldsymbol{L}, \boldsymbol{R}$ such that the satisfiability of C can be expressed as follows:

$$\begin{cases} \boldsymbol{a} + \boldsymbol{L} \cdot \boldsymbol{c} = 0 \\ \boldsymbol{b} + \boldsymbol{R} \cdot \boldsymbol{c} = 0 \\ \boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c} \end{cases} \tag{22}$$

with $\boldsymbol{a} = (1, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{a'})$, $\boldsymbol{b} = (\boldsymbol{1}, \boldsymbol{b'})$ and $\boldsymbol{c} = (1, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c'})$.

Let us partition $[n]$ into $I_{\text{in}} = \{2, \ldots, \ell_{\text{in}} + 1\}$, $I_{\text{out}} = \{\ell_{\text{in}} + 2, \ldots, \ell\}$, $I_{\text{mid}} = \{\ell + 1, \ldots, n\}$.

Let us label all the multiplication gates of C with integers in I_{mid}, and for every such multiplication gate j let us denote by a_j, b_j and c_j its left input, right input and output respectively. Also recall that $\boldsymbol{c}$ contains the public input and output (including the constant 1) as a prefix. Then the consistency of every multiplication gate can be checked as:

$$\forall j \in I_{\text{mid}} : \begin{cases} a_j + \boldsymbol{L}_j \cdot \boldsymbol{c} = 0 \\ b_j + \boldsymbol{R}_j \cdot \boldsymbol{c} = 0 \\ a_j \cdot b_j - c_j = 0 \end{cases}$$

for appropriate row vectors $\boldsymbol{L}_j, \boldsymbol{R}_j$ which express the linear subcircuits for the left and input wires.

Next, we add constraints for the public outputs:

$$\forall j \in I_{\text{out}} : \begin{cases} a_j + \boldsymbol{L}_j \cdot \boldsymbol{c} = 0 \\ b_j - c_1 = 0 \\ a_j \cdot b_j - c_j = 0 \end{cases}$$

The first constraint checks correctness of outputs that are obtained from possible linear subcircuits on multiplication gates outputs. The second and third constraints impose a dummy value, 1, on b_j, and thus ensure that $c_j = a_j$.

Finally, recalling that $\boldsymbol{a} = (1, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{a'})$, $\boldsymbol{b} = (\boldsymbol{1}, \boldsymbol{b'})$ and $\boldsymbol{c} = (1, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c'})$, we can add the following (dummy) constraints for the public inputs:

$$\forall j \in \{1\} \cup I_{\text{in}} : \begin{cases} a_j - c_j = 0 \\ b_j - c_1 = 0 \\ a_j \cdot b_j - c_j = 0 \end{cases}$$

We conclude by showing how to define matrices $\boldsymbol{L}$ and $\boldsymbol{R}$ such that all the constraints above are compactly represented by the equations (22).

$$\boldsymbol{L} = \begin{pmatrix} -\boldsymbol{I}_{\ell_{\text{in}}+1} \mid \boldsymbol{0} \\ \boldsymbol{L}_{\ell_{\text{in}}+2} \\ \vdots \\ \boldsymbol{L}_n \end{pmatrix}$$

In order to define $\boldsymbol{R}$, we define auxiliary matrix $\boldsymbol{E}$ as the $\ell \times n$ matrix where each row is the unit vector $\boldsymbol{e}_1 \in \mathbb{F}^n$.

$$\boldsymbol{E} = \begin{pmatrix} 1 & \dots & 0 & \dots & 0 \\ & & \vdots & & \\ 1 & \dots & 0 & \dots & 0 \end{pmatrix}, \quad \boldsymbol{R} = \begin{pmatrix} -\boldsymbol{E} \\ \boldsymbol{R}_{\ell+1} \\ \vdots \\ \boldsymbol{R}_n \end{pmatrix}$$

□

A.4 Comparing R1CS and R1CS-lite

We compare the efficiency of the R1CS and R1CS-lite constraint systems that result from applying the results of theorems A.2 and A.3 to the same circuit C. Let $\hat{\boldsymbol{L}}, \hat{\boldsymbol{R}}, \hat{\boldsymbol{O}}$ and $\boldsymbol{L}, \boldsymbol{R}$ be the resulting R1CS and R1CS-lite matrices respectively.

From the theorems statements it s clear that R1CS matrices have $\ell_{\text{in}} + 1$ less rows than R1CS-lite ones.

Next, we analyze their densities. For ease of comparison, we show how the R1CS matrices are obtained. This works very similarly to the proof of our theorem A.3 with a few differences at the end. For $\boldsymbol{c} = (1, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c}')$, the satisfiability can be expressed as the following constraints

$$\begin{cases} (\boldsymbol{L}' \cdot \boldsymbol{c}) \circ (\boldsymbol{R}' \cdot \boldsymbol{c}) = [\boldsymbol{0}_{N\times\ell} \mid \boldsymbol{I}] \cdot \boldsymbol{c} \\ \boldsymbol{L}_{out} \cdot \boldsymbol{c} = \boldsymbol{0} \end{cases} \quad \text{with } \boldsymbol{L}' = \begin{pmatrix} \boldsymbol{L}_{\ell+1} \\ \vdots \\ \boldsymbol{L}_n \end{pmatrix}, \quad \boldsymbol{R}' = \begin{pmatrix} \boldsymbol{R}_{\ell+1} \\ \vdots \\ \boldsymbol{R}_n \end{pmatrix}$$

where the i-th row of $\boldsymbol{L}_{out}$ checks that the i-th output $\boldsymbol{y}_i$ is correctly obtained from a linear subcircuit with inputs from $(1, \boldsymbol{x}, \boldsymbol{c}')$.

Then one can set

$$\hat{\boldsymbol{L}} = \begin{pmatrix} \boldsymbol{L}' \\ \boldsymbol{L}_{out} \end{pmatrix}, \quad \hat{\boldsymbol{R}} = \begin{pmatrix} \boldsymbol{R}' \\ (1, 0, \dots 0) \\ \vdots \\ (1, 0, \dots 0) \end{pmatrix}, \quad \hat{\boldsymbol{O}} = \begin{pmatrix} \boldsymbol{0} \mid \boldsymbol{I} \\ \boldsymbol{0}_{\ell_{\text{out}} \times n} \end{pmatrix}$$

Let us now analyze their densities. For R1CS we have:

$$||\hat{\boldsymbol{L}}|| = ||\boldsymbol{L}'|| + ||\boldsymbol{L}_{out}||, \quad ||\hat{\boldsymbol{R}}|| = ||\boldsymbol{R}'|| + \ell_{\text{out}}, \quad ||\hat{\boldsymbol{O}}|| = N$$

whereas for R1CS-lite we have:

$$||\boldsymbol{L}|| = ||\boldsymbol{L}'|| + ||\boldsymbol{L}_{out}|| + |I_{\text{in}}| + 1, \quad ||\boldsymbol{R}|| = ||\boldsymbol{R}'|| + \ell,$$

Basically, the two R1CS-lite matrices have each $\ell_{\text{in}} + 1$ more entries than their R1CS counterparts. If we consider the total size of the constraint system, we also have that

$$||\boldsymbol{L}|| + ||\boldsymbol{R}|| < ||\hat{\boldsymbol{L}}|| + ||\hat{\boldsymbol{R}}|| + ||\hat{\boldsymbol{O}}||$$

holds as long as $N > 2(\ell_{\text{in}} + 1)$, which is likely to be the case.

B Our Protocol for Lincheck

In this section we describe a PHP for the following relation

$$\boldsymbol{f} = \boldsymbol{M}\boldsymbol{g}$$

where $f, g \in \mathbb{F}^n$ and $\boldsymbol{M} \in \mathbb{F}^{n\times n}$ is a sparse matrix. This PHP is used as a building block for the constructions in Section 4.

B.1 Preliminaries

The equation $\boldsymbol{f} = \boldsymbol{M}\boldsymbol{g}$ can be rewritten as

$$\forall \eta \in \mathbb{H} \quad f(\eta) - \sum_{\eta'} M_{\eta,\eta'} \cdot g(\eta') = 0 \tag{23}$$

We can rewrite it equivalently as a linear combination through Lagrange polynomials:

$$\sum_{\eta} \mathcal{L}^{\mathbb{H}}_{\eta}(Y)\big(f(\eta) - \sum_{\eta'} M_{\eta,\eta'} \cdot g(\eta')\big) = 0 \tag{24}$$

that is

$$F(Y) = \sum_{\eta,\eta'} \mathcal{L}^{\mathbb{H}}_{\eta}(Y) M_{\eta,\eta'} g(\eta') \tag{25}$$

From here (we mark substitutions in blue):

$$\begin{aligned}
F(Y) &= \sum_{\substack{\eta,\eta' \\ M_{\eta,\eta'}\neq 0}} \mathcal{L}^{\mathbb{H}}_{\eta}(Y) M_{\eta,\eta'} g(\eta') && (26)\\
&= \sum_{\kappa\in\mathbb{K}} \mathsf{val}(\kappa) g(\mathsf{col}(\kappa)) \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(Y) && (27)\\
&= \sum_{\kappa\in\mathbb{K}} \mathsf{val}(\kappa) \big(\sum_{\eta'} g(\eta') \mathcal{L}^{\mathbb{H}}_{\mathsf{col}(\kappa)}(\eta')\big) \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(Y) && (28)\\
&= \sum_{\eta'} g(\eta') \big(\sum_{\kappa\in\mathbb{K}} \mathsf{val}(\kappa) \mathcal{L}^{\mathbb{H}}_{\mathsf{col}(\kappa)}(\eta') \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(Y)\big) && (29)\\
&= \sum_{\eta'} g(\eta') V(\eta', Y) && (30)\\
&= \sum_{\eta'} D(\eta', Y) && (31)
\end{aligned}$$

Thus we are defining:

$$\begin{aligned}
F(Y) &:= \sum_{\eta} \mathcal{L}^{\mathbb{H}}_{\eta}(Y) f(\eta) && (32)\\
D(X,Y) &:= g(X) V(X,Y) && (33)\\
V(X,Y) &:= \sum_{\kappa\in\mathbb{K}} \mathsf{val}(\kappa) \mathcal{L}^{\mathbb{H}}_{\mathsf{col}(\kappa)}(X) \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(Y) && (34)
\end{aligned}$$

A high-level view of the lincheck protocol:

1. **Sumcheck.** After sampling a random y_0, we carry out a sumcheck protocol on $\sigma := F(y_0)$. Specifically we check the claim $\sigma = \sum_{\eta'} D'(\eta')$ where $D'(X) := g(X)V(X, y_0)$.
2. **Check product of polynomials.** As a step in the sumcheck protocol from the step before, we need to recursively check the structure of D'. We want to check that D' is of the form $D'(X) = g(X)V(X, y_0)$. In order to do this, we sample a random point x and recursively check a claim on $g(x)$ (to which the verifier has oracle access) and on $V(x, y)$ (for which we use the following step).
3. **Check sparse matrix polynomial.** The polynomial V encodes a sparse matrix. We use a specialized protocol for this type of claim, described later on in the following subsection.

B.2 An Holographic Protocol for Points of Sparse Matrices

The new protocol we describe here is, together with sumcheck, the main ingredient for our lincheck. The protocol allows to verify that a polynomial $\tilde{V}$ in input correctly "encodes" the point with indices (x^*, y^*) for a matrix encoded as described in Definition 4.3. We point out that the indices (x^*, y^*) do not necessarily have to be valid row/column position of the matrix (they are arbitrary field elements).

Remark 16. *The approach below can be generalized to a different number of polynomial and different rational functions.*

See Section 4.1 for background on this section. Consider a sparse matrix $\boldsymbol{M}$ and let $(\mathsf{row}, \mathsf{col}, \mathsf{val})$ be its encoding (as by Definition 4.3) to which we assume the verifier has oracle access. Let x^*, y^* be any two points in the field $\mathbb{F}$. Let us define the polynomial $\tilde{V}$ as

$$\tilde{V} := \sum_{\kappa \in \mathbb{K}} \mathsf{val}(\kappa) \mathcal{L}^{\mathbb{H}}_{\mathsf{col}(\kappa)}(x^*) \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(y^*) \tag{35}$$

We now define a protocol that, fixed a matrix and fixed x^*, y^* allows a verifier to check that $\tilde{V}$ is correctly defined as in (35). Its analysis (efficiency and soundness) are analogous to those in Section 4.

Intuition

By definition of $\tilde{V}$, we know that

$$\tilde{V}(\kappa) = \mathsf{val}(\kappa) \mathcal{L}^{\mathbb{H}}_{\mathsf{col}(\kappa)}(x^*) \mathcal{L}^{\mathbb{H}}_{\mathsf{row}(\kappa)}(y^*) \text{ for every } \kappa \in \mathbb{K}$$

For every $\kappa \in \mathbb{K}$ we can write this as:

$$\tilde{V}(\kappa) = \mathsf{val}(\kappa) \frac{\mathsf{col}(\kappa)}{n} \cdot \frac{\mathcal{Z}_{\mathbb{H}}(x^*)}{x^* - \mathsf{col}(\kappa)} \cdot \frac{\mathsf{row}(\kappa)}{n} \cdot \frac{\mathcal{Z}_{\mathbb{H}}(y^*)}{y^* - \mathsf{row}(\kappa)}$$

where $n := |\mathbb{H}|$. The latter is equivalent to

$$n^2 \tilde{V}(\kappa)(x^* - \mathsf{col}(\kappa))(y^* - \mathsf{row}(\kappa)) = \mathsf{val}(\kappa)\mathsf{col}(\kappa)\mathsf{row}(\kappa)\mathcal{Z}_{\mathbb{H}}(x^*)\mathcal{Z}_{\mathbb{H}}(y^*)$$

Let us define polynomial B as

$$B(X') := n^2 \tilde{V}(X')(x^* - \mathsf{col}(X'))(y^* - \mathsf{row}(X')) - \mathsf{val}(X')\mathsf{col}(X')\mathsf{row}(X')\mathcal{Z}_{\mathbb{H}}(x^*)\mathcal{Z}_{\mathbb{H}}(y^*)$$

and we observe that the above is equivalent to:

$$B(X') \equiv 0 \pmod{\mathcal{Z}_{\mathbb{K}}(X')}$$

In order to check this relation in the protocol, the prover can send $\pi(X') := \frac{B(X')}{\mathcal{Z}_{\mathbb{K}}(X')}$ and the verifier can check

$$\pi(y)\mathcal{Z}_{\mathbb{K}}(y) = B(y)$$

for a random point y.

B.3 The linear check protocol

$\mathcal{P}(\mathbb{F}, M, F, G)$ $\qquad\qquad$ $\mathcal{V}^{F,G,\mathsf{row},\mathsf{col},\mathsf{val}}(\mathbb{F}))$

$\longleftarrow$ $y_0 \leftarrow\!\!\$\ \mathbb{F}$

$$D' := \sum_{\eta \in \mathbb{H}} G(\eta) V(\eta, y_0)$$

$$(q', r') \leftarrow \mathsf{DivPoly}_{\mathbb{H}}(D', F(y_0))$$

$\longrightarrow$ q', r'

$\longleftarrow$ $x \leftarrow\!\!\$\ \mathbb{F}$

$$y_G \leftarrow G(x)$$

$$\sigma_V \leftarrow V(x, y_0)$$

$$\tilde{V} := \sum_{\kappa \in \mathbb{K}} \mathsf{val}(\kappa) \Lambda_{\mathbb{H}}(x, \mathsf{col}(\kappa)) \Lambda_{\mathbb{H}}(y_0, \mathsf{row}(\kappa))$$

Let $\tilde{r}$ s.t. $\tilde{V}(X) = \sigma_V / n + X \tilde{r}(X)$

$$\pi(Y) := \frac{n^2}{\mathcal{Z}_{\mathbb{K}}(Y)} \tilde{V}(Y)(x - \mathsf{col}(Y))(y_0 - \mathsf{row}(Y)) - \mathsf{val}(Y)\mathsf{col}(Y)\mathsf{row}(Y)\mathcal{Z}_{\mathbb{H}}(x)\mathcal{Z}_{\mathbb{H}}(y_0)$$

$\longrightarrow$ $\sigma_V, \tilde{r}, \pi$

Figure 4: Online phase of our lincheck PHP.

Let $\tilde{p}(X) := \dfrac{\sigma_V}{n} + X\tilde{r}(X)$.

1) $n^2 \tilde{p}(X)(x - \mathsf{col}(X))(y_0 - \mathsf{row}(X)) - \mathcal{Z}_{\mathbb{H}}(x)\mathcal{Z}_{\mathbb{H}}(y_0)\mathsf{val}(X)\mathsf{col}(X)\mathsf{row}(X) - \pi(X)\mathcal{Z}_{\mathbb{K}}(X) \stackrel{?}{=} 0$

2) $G(X)\sigma_V - F(y_0)/n - q'(x)\mathcal{Z}_{\mathbb{H}}(x) - x r'(x) \stackrel{?}{=} 0$

Figure 5: Decision phase of our Lincheck PHP

C Additional Material for Section 6

C.1 Succinct Interactive Arguments in the SRS.

Definition C.1 (Universal Commit and Prove Interactive Argument in the SRS model)**.** *A Universal Commit and Prove Interactive Argument in the SRS model (Universal CP-UIA) is a tuple* $\mathsf{UIA} = (\mathsf{KeyGen}, \mathsf{Derive}, \bar{\mathcal{P}}, \bar{\mathcal{V}})$ *of PT algorithms where all the algorithms work as in Def. 5.4 (universal CP-SNARKs) but where* $\bar{\mathcal{P}}$ *and* $\bar{\mathcal{V}}$ *form an interactive protocol. In particular:*

- $\bar{\mathcal{P}}(\mathsf{ek},\ \hat{\mathsf{x}} = (\mathsf{ck}, \mathsf{x}, (\hat{c}_j)_{j\in[\ell]}),\ \hat{\mathsf{w}} = ((\mathsf{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega)),\ \bar{\rho}_1, \dots, \bar{\rho}_i)$*: the next-message function of* $\bar{\mathcal{P}}$ *takes as input the evaluation key* ek*, the instance* $\hat{\mathsf{x}}$*, the witness* $\hat{\mathsf{w}}$ *and the first i-th messages* $\bar{\rho}_1, \dots, \bar{\rho}_i$ *from* $\bar{\mathcal{V}}$*.*
- $\bar{\mathcal{V}}(\mathsf{srs}, \mathsf{vk}, \hat{\mathsf{x}}, \pi_1, \dots, \pi_i)$*: the next-message function of* $\bar{\mathcal{V}}$ *takes as input the verification key* vk*, the instance* $\hat{\mathsf{x}}$ *and the first i-th messages* $\pi_1, \dots, \pi_i$ *from the prover* $\bar{\mathcal{P}}$*.*
  At the last round of interaction the verifier outputs a decision bit b*.*

When $\ell = 0$ *we simply call it a Universal Interactive Argument in the SRS model (Universal UIA). Furthermore, we define the properties of knowledge-soundness as in Def. 5.4 and of trapdoor-commitment honest-verifier zero-knowledge similar to Def. 5.7. Specifically:*

Knowledge Soundness. *Let* $\mathcal{RG}$ *be a relation generator and* $\mathcal{Z}$ *an auxiliary input distribution, and let* $\mathcal{RG}_{\mathsf{Com}}$ *as in Def. 5.4,* UIA *has knowledge soundness for* $\mathcal{RG}$ *and* $\mathcal{Z}$*, denoted* $\mathsf{KSND}(\mathcal{RG}, \mathcal{Z})$ *for brevity, if for every (non-uniform) efficient adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *there exists a (non-uniform) efficient extractor* $\mathcal{E}$ *such that* $\Pr\left[\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}}(\lambda) = 1\right] = \mathsf{negl}$*. We say that* Π *is knowledge sound if there exist benign* $\mathcal{RG}$ *and* $\mathcal{Z}$ *such that* Π *is* $\mathsf{KSND}(\mathcal{RG}, \mathcal{Z})$*.*

$\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}}(\lambda) \to b$

$((\mathsf{ck}, \mathsf{R}), \mathsf{aux}_R) \leftarrow \mathcal{RG}_{\mathsf{Com}}(1^\lambda)\ ;\ \mathsf{srs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{ck}, \mathsf{R})\ ;\ \mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathsf{R}, \mathsf{aux}_R, \mathsf{srs})$

$\left(\mathsf{R}, \hat{\mathsf{x}} = (\mathsf{x}, (c_j)_{j\in[\ell]}), st\right) \leftarrow \mathcal{A}_0(\mathsf{R}, \mathsf{ck}, \mathsf{srs}, \mathsf{aux}_R, \mathsf{aux}_Z)\ ;\ (\tau, b') \leftarrow \langle \mathcal{A}_1(st), \bar{\mathcal{V}}(\mathsf{Derive}(\mathsf{vk}, \mathsf{R}), \mathsf{x}, (c_j)_{j\in[\ell]}) \rangle$

$\hat{\mathsf{w}} = ((\mathsf{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega) \leftarrow \mathcal{E}(\mathsf{R}, \mathsf{srs}, \mathsf{aux}_R, \mathsf{aux}_Z)\ ;\ \textbf{return } b = b' \wedge \neg \mathsf{R}^{\mathsf{Com}}_{\mathsf{ck},\mathsf{N}}(\hat{\mathsf{x}}, \hat{\mathsf{w}})$

Trapdoor-commitment honest-verifier zero-knowledge. *A CP-UIA scheme* UIA *is trapdoor-commit honest-verifer zero-knowledge in the SRS model for a family of universal relations* $\{\mathcal{R}_\mathsf{N}\}_{\mathsf{N}\in\mathbb{N}}$ *if there exists a simulator* $\mathcal{S}$ *such that for all adversaries* $\mathcal{A}$*, for all* $\mathsf{N} \in \mathbb{N}$*, for all* $\mathsf{R} \in \mathcal{R}_\mathsf{N}$*, for all* $(\mathsf{ck}, \mathsf{td}) \in \mathcal{S}_{\mathsf{ck}}(1^\lambda)$*, and for all* (x, w) *such that* $\mathsf{R}(\mathsf{x}, \mathsf{w}) = 1$*:*

$$\Pr\left[\begin{array}{r} (\mathsf{srs}, \mathsf{td}_\mathsf{k}) \leftarrow \mathsf{KeyGen}(\mathsf{ck}, \mathsf{N}) \\ \mathsf{srs}_\mathsf{R} \leftarrow \mathsf{Derive}(\mathsf{srs}, \mathsf{R}) \\ (\tau, b) \leftarrow \langle \mathbb{P}(\mathsf{ek}, \hat{\mathsf{x}}, \hat{\mathsf{w}}), \mathbb{V}(\mathsf{vk}, \hat{\mathsf{x}}) \rangle \end{array} : \mathcal{A}(\mathsf{srs}, \mathsf{td}_\mathsf{k}, \mathsf{td}, \mathsf{R}, \mathsf{x}, \mathsf{w}, \tau) = 1\right] \approx$$

$$\Pr\left[\begin{array}{r} (\mathsf{srs}, \mathsf{td}_\mathsf{k}) \leftarrow \mathsf{KeyGen}(\mathsf{ck}, \mathsf{N}) \\ \tau \leftarrow \mathcal{S}(\mathsf{td}_\mathsf{k}, \mathsf{td}, \mathsf{R}, \mathsf{x}) \end{array} : \mathcal{A}(\mathsf{srs}, \mathsf{td}_\mathsf{k}, \mathsf{td}, \mathsf{R}, \mathsf{x}, \mathsf{w}, \tau) = 1\right]$$

where recall that $\hat{\mathsf{x}} = (\mathsf{x}, (c_j)_{j\in[\ell]})$ *and* $\hat{\mathsf{w}} = ((\mathsf{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega)$*.*

Succinctness. *We say that a public-coin CP-UIA scheme* UIA *is succinct if, for any input* $\mathsf{x} \in \{0,1\}^n$*, both its total communication complexity (the sum of the length of all prover's messages) and the runtime of* $\bar{\mathcal{V}}$ *are at most poly-logarithmic in* n*.*

C.2 Proof of Theorem 6.1

Knowledge Soundness. Let $\mathcal{Z}$ be a benign auxiliary input distribution and let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be a non-uniform PT adversary for the knowledge soundness game described in Def. C.1. In Figure 6 we describe a sequence of hybrid experiments, the experiments are parameterized by an algorithm $\mathcal{E}^*$ that they run internally.

$\mathbf{H}_0(1^\lambda, \mathcal{E}^*)$

$\mathsf{pp} \leftarrow \mathsf{ParGen}(1^\lambda)$; $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{N})$; $\mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathsf{srs})$;
$(\mathsf{R}, \mathsf{x}, st) \leftarrow \mathcal{A}_0(\mathsf{srs}, \mathsf{aux}_Z)$; $(\tau, b) \leftarrow \langle \mathcal{A}_1(st), \mathbb{V}(\mathsf{Derive}(\mathsf{vk}, \mathsf{R}), \mathsf{x} \rangle$;
$\mathsf{w} \leftarrow \mathcal{E}^*(\mathsf{srs}, \mathsf{aux}_Z)$;
$b_E \leftarrow \mathsf{R}(\mathsf{x}, \omega)$; $b_W \leftarrow b$

$\mathbf{H}_1(1^\lambda, \mathcal{E}^*)$

$\mathsf{pp} \leftarrow \mathsf{ParGen}(1^\lambda)$; $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{N})$; $\mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathsf{srs})$;
$(\mathsf{R}, \mathsf{x}, st) \leftarrow \mathcal{A}_0(\mathsf{srs}, \mathsf{aux}_Z)$; $(\tau, b) \leftarrow \langle \mathcal{A}_1(st), \mathbb{V}(\mathsf{Derive}(\mathsf{vk}, \mathsf{R}), \mathsf{x} \rangle$;
$\boxed{(p'_j, o'_j)_{j \in [\mathsf{n}^*]} \leftarrow \mathcal{E}_{\mathsf{php}}(\mathcal{R}_{\mathsf{php}}, \mathsf{ck}, \mathsf{vk}_{\mathsf{php}}, \mathsf{aux}_Z^{\mathsf{PHP}})}$;
$\boxed{b_V \leftarrow \bigwedge_{k \in \mathsf{n}_\mathsf{e}} \big(G_k(X, \{p'_j(v_{k,j}(X))\}_{j \in [\mathsf{n}^*]}, \{\pi_j\}_{j \in [\mathsf{m}^*]}) \equiv 0\big) \wedge \bigwedge_{j \in [\mathsf{n}^*]} \big(\mathsf{deg}(p'_j) \leq d_j\big) \wedge \bigwedge_{j \in [\mathsf{n}^*]} \mathsf{VerCom}(\mathsf{ck}, c_j, p'_j, o'_j)}$;
$\mathsf{w} \leftarrow \mathcal{E}^*(\mathsf{srs}, \mathsf{aux}_Z)$;
$b_E \leftarrow \mathsf{R}(\mathsf{x}, \omega)$; $\boxed{b_W \leftarrow b \wedge b_V}$;

$\mathbf{H}_2(1^\lambda, \mathcal{E}^*)$

$\mathsf{pp} \leftarrow \mathsf{ParGen}(1^\lambda)$; $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{N})$; $\mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathsf{srs})$;
$(\mathsf{R}, \mathsf{x}, st) \leftarrow \mathcal{A}_0(\mathsf{srs}, \mathsf{aux}_Z)$; $(\tau, b) \leftarrow \langle \mathcal{A}_1(st), \mathbb{V}(\mathsf{Derive}(\mathsf{vk}, \mathsf{R}), \mathsf{x} \rangle$;
$(p'_j, o'_j)_{j \in [\mathsf{n}^*]} \leftarrow \mathcal{E}_{\mathsf{php}}(\mathcal{R}_{\mathsf{php}}, \mathsf{ck}, \mathsf{vk}_{\mathsf{php}}, \mathsf{aux}_Z^{\mathsf{PHP}})$;
$b_V \leftarrow \bigwedge_{k \in \mathsf{n}_\mathsf{e}} \big(G_k(X, \{p'_j(v_{k,j}(X))\}_{j \in [\mathsf{n}^*]}, \{\pi_j\}_{j \in [\mathsf{m}^*]}) \equiv 0\big) \wedge \bigwedge_{j \in [\mathsf{n}^*]} \big(\mathsf{deg}(p'_j) \leq d_j\big) \wedge \bigwedge_{j \in [\mathsf{n}^*]} \mathsf{VerCom}(\mathsf{ck}, c_j, p'_j, o'_j)$;
$\boxed{\textbf{for } i \in [\mathsf{r}] : (p_{i,j}, o_{i,j})_{j \in [\mathsf{n}(i)]} \leftarrow \mathcal{E}'_j(\mathsf{ek}_{\mathsf{opn}}, \mathsf{aux}_Z) \; ; \; b_V^i \leftarrow \bigwedge_{j \in [\mathsf{n}(i)]} \mathsf{VerCom}(\mathsf{ck}, c_{i,j}, p_{i,j}, o_{i,j})}$
$\mathsf{w} \leftarrow \mathcal{E}^*(\mathsf{srs}, \mathsf{aux}_Z)$;
$b_E \leftarrow \mathsf{R}(\mathsf{x}, \omega)$; $\boxed{b_W \leftarrow b \wedge b_V \wedge (\wedge_i b_V^i)}$;

$\mathbf{H}_3(1^\lambda, \mathcal{E}^*)$

$\mathsf{pp} \leftarrow \mathsf{ParGen}(1^\lambda)$; $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{N})$; $\mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathsf{srs})$;
$(\mathsf{R}, \mathsf{x}, st) \leftarrow \mathcal{A}_0(\mathsf{srs}, \mathsf{aux}_Z)$; $(\tau, b) \leftarrow \langle \mathcal{A}_1(st), \mathbb{V}(\mathsf{Derive}(\mathsf{vk}, \mathsf{R}), \mathsf{x} \rangle$;
$(p'_j, o'_j)_{j \in [\mathsf{n}^*]} \leftarrow \mathcal{E}_{\mathsf{php}}(\mathcal{R}_{\mathsf{php}}, \mathsf{ck}, \mathsf{vk}_{\mathsf{php}}, \mathsf{aux}_Z^{\mathsf{PHP}})$;
$b_V \leftarrow \bigwedge_{k \in \mathsf{n}_\mathsf{e}} \big(G_k(X, \{p'_j(v_{k,j}(X))\}_{j \in [\mathsf{n}^*]}, \{\pi_j\}_{j \in [\mathsf{m}^*]}) \equiv 0\big) \wedge \bigwedge_{j \in [\mathsf{n}^*]} \big(\mathsf{deg}(p'_j) \leq d_j\big) \wedge \bigwedge_{j \in [\mathsf{n}^*]} \mathsf{VerCom}(\mathsf{ck}, c_j, p'_j, o'_j)$;
$\textbf{for } i \in [\mathsf{r}] : (p_{i,j}, o_{i,j})_{j \in [\mathsf{n}(i)]} \leftarrow \mathcal{E}'_j(\mathsf{ck}, \mathsf{aux}_R, \mathsf{aux}_Z)$; $b_V^i \leftarrow \bigwedge_{j \in [\mathsf{n}(i)]} \mathsf{VerCom}(\mathsf{ck}, c_{i,j}, p_{i,j}, o_{i,j})$
$\boxed{\textbf{let } (p_j)_{j \in [\mathsf{n}^*]} := (p_{i,j})_{i \in [\mathsf{r}], j \in [\mathsf{n}(i)]} \; ; \; b_{\mathsf{CS}} \leftarrow \forall j : p'_j = p_j}$;
$\mathsf{w} \leftarrow \mathcal{E}^*(\mathcal{R}, \mathsf{srs}, \mathsf{aux}_R, \mathsf{aux}_Z)$;
$b_E \leftarrow \mathsf{R}(\mathsf{x}, \omega)$; $\boxed{b_W \leftarrow b \wedge b_V \wedge (\wedge_i b_V^i) \wedge b_{\mathsf{CS}}}$;

Figure 6: Hybrid Experiments for Proof of Theorem 6.1

Consider the hybrid $\mathbf{H}_0(1^\lambda, \mathcal{E}^*)$. Let b_W^j (resp. b_E^j) be the event of the flag b_W (resp. b_E) being true in the hybrid experiment$\mathbf{H}_j$. Formally the events should be parameterized by the extractor $\mathcal{E}^*$ that the hybrid is running. However, it is clear that the variable b_W^j does not depend on the specific of $\mathcal{E}^*$, thus for a cleaner presentation we omit it. On the other hand b_E^j depends on $\mathcal{E}^*$, thus when needed we will refer to $b_E^j[\mathcal{E}^*]$ to specify that the event is parameterized by the extractor $\mathcal{E}^*$. By a simple derivation:

$$\Pr\left[\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{Z},\mathcal{A},\mathcal{E}^*}\right] \le \Pr\left[b_W^0\right] - \Pr\left[b_E^0[\mathcal{E}^*]\right].$$

Let $\mathbf{H}_1(1^\lambda, \mathcal{E}^*)$ be the same as $\mathbf{H}_0$ but where the variable b_W is computed differently. Specifically, let $\mathcal{E}_{\mathsf{php}}$ be the extractor for the CP-SNARK $\mathsf{CP}_{\mathsf{php}}$ and the adversary $\mathcal{A}_{\mathsf{php}}$ that runs the same as the adversary $\mathcal{A}$ but that simply outputs the proof π_{php}, the openings $(p_j)_{j\in[\mathsf{n}(0)]}, (o_j)_{j\in[\mathsf{n}(0)]}$ for the `rel`-typed commitments and the relative statement. Formally, the adversary $\mathcal{A}_{\mathsf{php}}$ receives in input $\mathcal{R}_{\mathsf{php}}, \mathsf{ck}, \mathsf{vk}_{\mathsf{php}}$ and $\mathsf{aux}_Z^{\mathsf{PHP}} := \mathsf{aux}_Z, \mathsf{vk}_{\mathsf{opn}}$.

Let ϵ_{php} be the knowledge soundness error (with partial opening) of the CP-SNARK $\mathsf{CP}_{\mathsf{php}}$

Lemma C.1. $\Pr[b_W^0] \le \Pr[b_W^1] + \epsilon_{\mathsf{PHP}}$

Proof. Notice that $\Pr[b_W^0] - \Pr[b_W^1] = \Pr[b_V = 0]$. In particular, as described in the definition of the hybrid, the adversary $\mathcal{A}_{\mathsf{php}}$ on input the state st runs an execution of the universal argument between $\mathcal{A}_1$ and the honest verifier and then outputs $\big((i_j, d_j)_{j\in[\mathsf{n_p}]}, (G_j, \boldsymbol{v}_j)_{j\in[\mathsf{n_e}]}\big), (c_j)_{j\in[\mathsf{n}^*]}\big)$ and the proof π_{php}. Since $b = 1$ then it means that the proof π_{php} is valid, but $b_V = 0$, i.e., the extractor does not output a valid witness. □

Let $\mathbf{H}_2(1^\lambda, \mathcal{E}^*)$ be the same as $\mathbf{H}_1$ but where the variable b_W is computed differently. Specifically, let $\mathcal{E}'_i$ be the extractor for the adversary $\mathcal{A}^i$ that runs the same as the adversary $\mathcal{A}$ but that simply outputs the $(c_{i,j})_{j\in[\mathsf{n}(i)]}, \pi_{\mathsf{opn},i}$.

Let ϵ_{opn} be the knowledge soundness error of the CP-SNARK $\mathsf{CP}_{\mathsf{opn}}$.

Lemma C.2. $\Pr[b_W^1] \le \Pr[b_W^2] + \mathsf{r}\cdot\epsilon_{\mathsf{opn}}$

Proof. Similarly to the previous lemma, $\Pr[b_W^1] - \Pr[b_W^2] = \Pr[b_W^1 \wedge \exists i : b_V^i = 0] \le \mathsf{r}\cdot\max_i \Pr[b_W^1 \wedge b_V^i = 0] \le \mathsf{r}\cdot\epsilon_{\mathsf{opn}}$. Since $b = 1$ then it means that $\mathsf{Verify}_{\mathsf{opn}}(\mathsf{vk}_{\mathsf{opn}}, (c_{i,j})_{j\in\mathsf{n}(i)}, \pi_{\mathsf{opn},i}) = 1$, but $b_V^i = 0$ thus the extractor does not extract valid openings the commitments. □

Let $\mathbf{H}_3(1^\lambda, \mathcal{E}^*)$ be the same as $\mathbf{H}_2$ but where the variable b_W is computed differently. Specifically, we check that the extractions of the CP-SNARKs agree. Let ϵ_{CS} be the advantage against the binding of CS.

Lemma C.3. $\Pr[b_W^2] \le \Pr[b_W^3] + \epsilon_{\mathsf{CS}}$

Proof. Notice that $\Pr[b_W^2] \le \Pr[b_W^3] + \Pr[b_{\mathsf{CS}} = 0]$. We reduce to the binding of CS. In particular consider the adversary that runs $\mathbf{H}_3$ and if $b_C = 0$ it outputs the values (p_j, o_j, p'_j, o'_j) for the index j that make $b_{\mathsf{CS}} = 0$. Thus $\Pr[b_{\mathsf{CS}} = 0] \le \epsilon_{\mathsf{CS}}$. □

Consider the following PPT sampler algorithm:

Sampler $\mathcal{D}(1^\lambda)$:

$\mathsf{pp} \leftarrow \mathsf{ParGen}(1^\lambda)$; $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{N})$; $\mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathsf{srs})$;
$(\mathsf{R}, \mathsf{x}, st) \leftarrow \mathcal{A}_0(\mathsf{srs}, \mathsf{aux}_Z)$; $\mathsf{aux}'_Z := (\mathsf{srs}, \mathsf{aux}_Z)$; **return** $(\mathsf{R}, \mathsf{x}, \mathsf{aux}'_Z)$

Lemma C.4. *There exists a prover $\mathcal{P}^*$ for the protocol* PHP *such that*

$$\Pr[\langle \mathcal{P}^*(\mathbb{F}, \mathsf{R}, \mathsf{x}, z), \mathcal{V}^{\mathcal{RE}(\mathbb{F},\mathsf{R})}(\mathbb{F}, \mathsf{x})\rangle = 1 : (\mathsf{R}, \mathsf{x}, \mathsf{aux}'_Z) \leftarrow_{\$} \mathcal{D}(1^\lambda)] = \Pr[b_W^3] \tag{36}$$

Proof. We define the prover $\mathcal{P}^*$. For any i, j, let $ind(i,j) := \sum_{i'<i} \mathsf{n}^*(i') + j$ and let ind^{-1} its inverse in the domain $[\mathsf{n}^*]$. Namely, the function ind re-indexes the j-th polynomial sent at the i-th round as the $ind(i,j)$-th polynomial sent by the prover.

Prover $\mathcal{P}^*(\mathsf{R}, \mathsf{x}, \mathsf{aux}'_Z, \rho_1, \ldots, \rho_i))$:

$(\mathsf{R}, \mathsf{x}, st) \leftarrow \mathcal{A}_0(\mathsf{aux}'_Z)$;
if $i = \mathsf{r}+1$ **then** :
  $(c_{i,j})_{j\in[\mathsf{n}(i)]}, \{\pi_{i,j}\}_{j\in[\mathsf{r}(i)]}, \pi_{\mathsf{php}} \leftarrow \mathcal{A}_1(st, \rho_1, \ldots, \rho_{\mathsf{r}})$;
  $\hat{\mathsf{x}}_{\mathsf{PHP}} := \big((i_j, d_j)_{j\in[\mathsf{n_p}]}, (G'_j, \boldsymbol{v}_j)_{j\in[\mathsf{n_e}]}, (c_j)_{j\in[\mathsf{n}^*]}\big)$;
  if $\mathsf{Verify_{php}}(\mathsf{vk_{php}}, \hat{\mathsf{x}}_{\mathsf{PHP}}, \pi) = 0$ **then return** $\perp$;
  $(p'_k, o'_k)_{k\in[\mathsf{n}^*]} \leftarrow \mathcal{E}_{\mathsf{php}}(\mathcal{R}_{\mathsf{php}}, \mathsf{ck}, \mathsf{srs}, \mathsf{aux}_Z)$;
  for $k \in [\mathsf{n}^*]$ **let** $p_{i,j} := p'_{ind^{-1}(k)}$;
  if $\exists i, j : p'_{i,j} \neq p_{i,j}$ **then return** $\perp$;
else :
  $(c_{i,j})_{j\in[\mathsf{n}(i)]}, \{\pi_{i,j}\}_{j\in[\mathsf{r}(i)]}, \pi_j \leftarrow \mathcal{A}_1(st, \rho_1, \ldots, \rho_i))$;
  if $\mathsf{Verify_{opn}}(\mathsf{vk_{php}}, , \pi_j) = 0$ **then return** $\perp$;
  $(p_{i,j}, o_{i,j})_{j\in[\mathsf{n}(i)]} \leftarrow \mathcal{E}'_j(\mathsf{ek_{opn}}, \mathsf{aux}_Z)$
if $\exists j : \mathsf{VerCom}(\mathsf{ck}, c_{i,j}, p_{i,j}, o_{i,j}) = 0$ **then return** $\perp$;
return $(p_{i,j})_{j\in[\mathsf{n}(i)]}, \{\pi_{i,j}\}_{j\in[\mathsf{r}(i)]}$;

By inspection, if $\mathcal{P}^*$ does not output $\perp$ then the output of $\mathcal{P}^*$ is computed exactly the same as $\mathcal{A}$ does. Moreover, the prover $\mathcal{P}^*$ outputs $\perp$ only when either the verification of the $\mathsf{CP_{opn}}$ fails (if round $i \leq \mathsf{r}$) or the verification of the $\mathsf{CP_{php}}$ fails (if last round) or exists index j s.t. $\mathsf{VerCom}(\mathsf{ck}, c_{i,j}, p_{i,j}, o_{i,j}) = 0$ or the binding property of CS is violated. Notice that if b^3_W holds then none of the previous events can happen.

□

Notice that we can rewrite the Eq. (36) as:

$$\mathbb{E}_{(\mathsf{R},\mathsf{x},\mathsf{aux}'_Z) \leftarrow\$ \mathcal{D}(1^\lambda)} \left[\Pr[\langle \mathcal{P}^*(\mathbb{F}, \mathsf{R}, \mathsf{x}, z), \mathcal{V}^{\mathcal{RE}(\mathbb{F},\mathsf{R})}(\mathbb{F}, \mathsf{x})\rangle = 1]\right] = \Pr[b^3_W]$$

Thus by applying the knowledge soundness of PHP there exists an extractor $\mathcal{E}$:

$$\mathbb{E}_{(\mathsf{R},\mathsf{x},\mathsf{aux}'_Z) \leftarrow\$ \mathcal{D}(1^\lambda)} \left[\Pr[\mathcal{E}^{\mathcal{P}^*}(\mathsf{R}, \mathsf{x}, \mathsf{aux}'_Z)] + \epsilon_{\mathsf{PHP}}\right] \geq \Pr[b^3_W]$$

Equivalently we can rewrite the equation above:

$$\Pr[\mathcal{E}^{\mathcal{P}^*}(\mathsf{R}, \mathsf{x}, \mathsf{aux}'_Z) : (\mathsf{R}, \mathsf{x}, \mathsf{aux}'_Z) \leftarrow\$ \mathcal{D}(1^\lambda)] + \epsilon_{\mathsf{PHP}} \geq \Pr[b^3_W].$$

Finally, we define the extractor $\mathsf{Ext}^* := \mathsf{Ext}^{\mathcal{P}^*}$. It is easy to see that:

$$\Pr[\mathsf{Ext}^{\mathcal{P}^*}(\mathsf{R}, \mathsf{x}, \mathsf{aux}'_Z) : (\mathsf{R}, \mathsf{x}, \mathsf{aux}'_Z) \leftarrow\$ \mathcal{D}(1^\lambda)] = \Pr\left[b^0_E\left[\mathcal{E}^*\right]\right]$$

The equation above holds by definition of $\mathcal{D}$ and $\mathcal{E}^*$. Thus we can conclude:

$$\begin{aligned}\Pr\left[\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{Z},\mathcal{A},\mathcal{E}^*}\right] &\leq \Pr[b^0_W] - \Pr[b^0_E] \;\leq\; \Pr[b^3_W] - \Pr[b^0_E] + N \cdot \epsilon_{\mathsf{snark}} + (\mathsf{n}^*+1)\epsilon_{\mathsf{CS}} \\ &\leq \epsilon_{\mathsf{PHP}} + N \cdot \epsilon_{\mathsf{snark}} + (\mathsf{n}^*+1)\epsilon_{\mathsf{CS}}.\end{aligned}$$

Zero Knowledge. We now prove that under the condition of the statement of the theorem the UIA is trapdoor commitment honest-verifier zero-knowledge.

Let C and $\mathcal{S}_{\mathsf{PHP}}$ be the checker and the simulator for the claimed $(\mathsf{C}, \mathsf{b_{PHP}})$-bounded zero-knowledge of PHP. Let $\mathcal{S} = (\mathcal{S}'_{\mathsf{leak}}, \mathcal{S}_{\mathsf{prv}}{}')$ be the simulator of $\mathsf{CP_{php}}$. Let $\mathcal{S}''$ be the simulator of $\mathsf{CP_{opn}}$. Consider the simulator $\mathcal{S}_{\mathsf{UIA}} = (\mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{prv}})$:

Simulator $\mathcal{S}_{\mathsf{kg}}(\mathsf{ck}, \mathsf{td}_{\mathsf{ck}}, \mathsf{N})$:

1. Sample $\mathsf{ek}_{\mathsf{opn}}, \mathsf{vk}_{\mathsf{opn}}, \mathsf{td}_{\mathsf{k}}^{\mathsf{opn}} \leftarrow\$ \mathcal{S}_{\mathsf{kg}}{}''(\mathsf{ck}, \mathsf{td}_{\mathsf{ck}})$ and output $\mathsf{ek}_{\mathsf{php}}, \mathsf{vk}_{\mathsf{php}}, \mathsf{td}_{\mathsf{k}}^{\mathsf{php}}$ and output $\mathsf{vk} = \mathsf{ck}, \mathsf{ek}_{\mathsf{opn}}, \mathsf{ek}_{\mathsf{php}}$ and $\mathsf{vk} = \mathsf{ck}, \mathsf{vk}_{\mathsf{opn}}, \mathsf{vk}_{\mathsf{php}}$ and $\mathsf{td}_{\mathsf{k}} := \mathsf{td}_{\mathsf{k}}^{\mathsf{opn}}, \mathsf{td}_{\mathsf{k}}^{\mathsf{php}}, \mathsf{td}_{\mathsf{ck}}$.

Simulator $\mathcal{S}_{\mathsf{prv}}(\mathsf{td}_{\mathsf{k}}, \mathsf{R}, \mathsf{x})$:

1. **Init Phase.** Let $\mathsf{r} := \mathsf{r}(|\mathsf{R}|)$. Run the honest verifier on input vk_{R} and x, obtain a sequence of messages $\rho_1, \ldots, \rho_{\mathsf{r}+1}$ and the constraints $((G_j, \boldsymbol{v}_j))_{j\in[\mathsf{n_e}]}, (i_j, d_j)_{j\in[\mathsf{n_p}]}$. Set $\bar{\rho} := (\rho_j)_{j\in[\mathsf{r}+1]}$.
   Parse the trapdoor td_{k} as $(\mathsf{td}_{\mathsf{k}}^{\mathsf{opn}}, \mathsf{td}_{\mathsf{k}}^{\mathsf{php}}, \mathsf{td}, s)$ where $s \in \mathbb{F}$.
2. **Define Leakage.** Let $\mathcal{L}' \leftarrow \mathcal{S}'_{\mathsf{leak}}(\hat{\mathsf{x}}_{\mathsf{PHP}})$, Let $\mathcal{L} := \mathcal{L}' \cup \{(i, s) : i \in [\mathsf{n}^*]\}$.Assert that $\mathsf{C}(i, x)$ for all $(i, x) \in \mathcal{L}$;
3. **Create Transcript.** Compute the following:
   (a) Run the simulator $\mathcal{S}_{\mathsf{PHP}}(\mathbb{F}, \mathsf{R}, \mathsf{x}, \mathcal{L})$ and obtain a simulated transcript $\tilde{\tau} = (\boldsymbol{\pi}_1, \rho_1, \ldots, \boldsymbol{\pi}_{\mathsf{r}}, \rho_{\mathsf{r}})$, and a set of simulated evaluated points $\{\tilde{p}_i(y) : (i, y) \in \mathcal{L}\}$;
   (b) For $j \in [\mathsf{n}^*]$ compute the simulated commitment: set $\tilde{c}_j, st_j \leftarrow \mathsf{TdCom}(\mathsf{td}_{\mathsf{k}}, \tilde{p}_j(s))$;
   (c) Let $leak' := (\tilde{p}_i(y))_{(i,y)\in\mathcal{L}}$ and let $\hat{\mathsf{x}}_{\mathsf{PHP}} := \Big((i_j, d_j)_{j\in[\mathsf{n_p}]}, (G'_j, \boldsymbol{v}_j)_{j\in[\mathsf{n_e}]}, (c_j)_{j\in[\mathsf{n}^*]}\Big)$, compute the simulated proof $\widetilde{\pi_{\mathsf{php}}} \leftarrow \mathcal{S}_{\mathsf{prv}}{}'(\mathsf{td}_{\mathsf{k}}^{\mathsf{php}}, \hat{\mathsf{x}}_{\mathsf{PHP}}, leak')$;
   (d) For $i \in [\mathsf{r}]$ compute the simulated proof $\widetilde{\pi_{\mathsf{opn},i}} \leftarrow \mathcal{S}''(\mathsf{td}_{\mathsf{k}}^{\mathsf{opn}}, (c_{i,j})_{j\in\mathsf{r}(i)})$;

   Output the full transcript re-ordered according to the specification of the protocol.

We consider a sequence of hybrid experiments. The first hybrid $\mathbf{H}_0$ receives in input the trapdoor td_{k}, the specific relation R, the input x and the witness w, runs the same steps of simulator $\mathcal{S}_{\mathsf{prv}}$ defined above, and outputs the full view including the evaluation points.

The next hybrid $\mathbf{H}_1$ runs the same as $\mathbf{H}_0$ but instead of running $\mathcal{S}_{\mathsf{PHP}}$ at step 3, it runs the real protocol between $\mathcal{P}$ and $\mathcal{V}$ and computes the evaluation points using the polynomial oracles output by $\mathcal{P}$.

Lemma C.5. *For all* $\mathsf{ck}, \mathsf{R}, \mathsf{x}, \mathsf{w}$ *and for any adversary* $\mathcal{A}$*:*

$$|\Pr[\mathcal{A}(\mathsf{srs}, \mathbf{H}_0(\mathsf{td}_{\mathsf{k}}, \mathsf{R}, \mathsf{x}, \mathsf{w})) = 1] - |\Pr[\mathcal{A}(\mathsf{srs}, \mathbf{H}_1(\mathsf{td}_{\mathsf{k}}, \mathsf{R}, \mathsf{x}, \mathsf{w})) = 1]| \in \mathsf{negl}(\lambda)$$

Proof. Notice that if the assertion in step 2 does not hold the two hybrids are equivalent. Thus we can assume the assertion holds, in this case the list $\mathcal{L}$ is $(\mathsf{C}, \mathsf{b} + \mu(\boldsymbol{t}))$-bounded. The proof of the lemma follows straightforwardly from the $(\mathsf{C}, \mathsf{b} + \mu(\boldsymbol{t}))$-bounded zero-knowledge of PHP. □

The next hybrid $\mathbf{H}_2$ is the same as $\mathbf{H}_1$ but the assertion in step 2 is not executed.

Lemma C.6. *For all* $\mathsf{ck}, \mathsf{R}, \mathsf{x}, \mathsf{w}$ *and for any adversary* $\mathcal{A}$*:*

$$|\Pr[\mathcal{A}(\mathsf{srs}, \mathbf{H}_1(\mathsf{td}_{\mathsf{k}}, \mathsf{R}, \mathsf{x}, \mathsf{w})) = 1] - |\Pr[\mathcal{A}(\mathsf{srs}, \mathbf{H}_2(\mathsf{td}_{\mathsf{k}}, \mathsf{R}, \mathsf{x}, \mathsf{w})) = 1]| \in \mathsf{negl}(\lambda)$$

Proof. The two hybrids diverge if there is tuple $(i, x) \in \mathcal{L}$ such that $\mathsf{C}(i, x) = 0$. Notice that, by our assumption on C (Definition 3.2), a tuple (i, s) does not pass the checker with negligible probability (since the trapdoor element s is chosen uniformly at random). Moreover, by the bounded leakage property of the (b, C)-leaky zero knowledge of $\mathsf{CP_{php}}$, a tuple in $\mathcal{L}$ does not pass the check with negligible probability. We can conclude applying union bound. □

The next hybrid $\mathbf{H}_3$ is the same as $\mathbf{H}_2$ but the commitments are computed as in the real protocol, specifically for any i, j where $i > 1$ we compute $c_{i,j}, o_{i,j} \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathtt{swh}, p_{i,j})$.

Lemma C.7. *For all* $\mathsf{ck}, \mathsf{R}, \mathsf{x}, \mathsf{w}$ *and for any adversary* $\mathcal{A}$*:*

$$|\Pr[\mathcal{A}(\mathsf{srs}, \mathbf{H}_2(\mathsf{td_k}, \mathsf{R}, \mathsf{x}, \mathsf{w})) = 1] - |\Pr[\mathcal{A}(\mathsf{srs}, \mathbf{H}_3(\mathsf{td_k}, \mathsf{R}, \mathsf{x}, \mathsf{w})) = 1]| \in \mathsf{negl}(\lambda)$$

The lemma easily follows by the `swh`-typed somewhat-hiding property of CS.
The next hybrid $\mathbf{H}_4$ is the same as $\mathbf{H}_3$ but at step (c), the proof is computed with the algorithm $\mathsf{Prove_{php}}$ and the polynomial oracles $p_1, \ldots, p_{\mathsf{n}^*}$.

Lemma C.8. *For all* $\mathsf{ck}, \mathsf{td_k}, \mathsf{R}, \mathsf{x}, \mathsf{w}$ *and for any adversary* $\mathcal{A}$*:*

$$|\Pr[\mathcal{A}(\mathsf{srs}, \mathbf{H}_3(\mathsf{td_k}, \mathsf{R}, \mathsf{x}, \mathsf{w})) = 1] - |\Pr[\mathcal{A}(\mathsf{srs}, \mathbf{H}_4(\mathsf{td_k}, \mathsf{R}, \mathsf{x}, \mathsf{w})) = 1]| \in \mathsf{negl}(\lambda)$$

The lemma easily follows by the leaky zero-knowledge of the CP-SNARK $\mathsf{CP_{php}}$.

The next hybrid $\mathbf{H}_5$ is the same as $\mathbf{H}_4$ but at step (d), for any round $i \in [\mathsf{r}]$, the proofs are computed with the algorithm $\mathsf{Prove_{opn}}$ and the polynomial oracles $p_{i,j}$.

Lemma C.9. *For all* $\mathsf{ck}, \mathsf{td_k}, \mathsf{R}, \mathsf{x}, \mathsf{w}$ *and for any adversary* $\mathcal{A}$*:*

$$|\Pr[\mathcal{A}(\mathsf{srs}, \mathbf{H}_4(\mathsf{td_k}, \mathsf{R}, \mathsf{x}, \mathsf{w})) = 1] - |\Pr[\mathcal{A}(\mathsf{srs}, \mathbf{H}_5(\mathsf{td_k}, \mathsf{R}, \mathsf{x}, \mathsf{w})) = 1]| \in \mathsf{negl}(\lambda)$$

The lemma easily follows by the zero-knowledge of the CP-SNARK $\mathsf{CP_{opn}}$.

C.3 Proof of Theorem 8.1

The proof of zero-knowledge is almost the same as in the proof of Thm. 6.1, for knowledge soundness there are some differences that we highlight next.

Consider the hybrid $\mathbf{H}_3$ as in the proof of Thm. 6.1 and the event b_W. For clarity we rewrite the hybrid below. The only difference with the hybrid from Thm. 6.1 is that they hybrid below does not run the extractor $\mathcal{E}^*$ since we are interested only in the event b_W.

$\mathbf{H}_3(1^\lambda)$

$\mathsf{pp} \leftarrow \mathsf{ParGen}(1^\lambda)$; $\mathsf{ck} \leftarrow \mathsf{CS}^*.\mathsf{Setup}(\mathsf{pp})$; $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{ck}, \mathsf{N})$; $\mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathsf{srs})$;
$\big(\mathsf{R}, \hat{\mathsf{x}} = (\mathsf{x}, (c_j)_{j\in[\ell]}), st\big) \leftarrow \mathcal{A}_0(\mathsf{srs}, \mathsf{aux}_Z)$; $(\tau, b) \leftarrow \langle \mathcal{A}_1(st), \mathbb{V}(\mathsf{Derive}(\mathsf{vk}, \mathsf{R}), \mathsf{x}, (c_j)_{j\in[\ell]})\rangle$;
$(p'_j, o'_j)_{j\in[\mathsf{n}^*]} \leftarrow \mathcal{E}(\mathcal{R}_{\mathsf{php}}, \mathsf{srs}, \mathsf{aux}_Z)$;
$b_V \leftarrow \bigwedge_k \big(G_k(X, \{p'_j(v_{k,j}(X))\}_{j\in[\mathsf{n}^*]}, \{\pi_j\}_{j\in[\mathsf{m}^*]}) \equiv 0\big) \wedge \bigwedge_k \big(\mathsf{deg}(p'_{i_k}) \le d_k\big) \wedge \bigwedge_k \mathsf{VerCom}(\mathsf{ck}, c_k, p'_k, o'_k)$;
for $j \in [\mathsf{r}] : (p_{i,j}, o_{i,j})_{j\in[\mathsf{n}(i)]} \leftarrow \mathcal{E}'_j(\mathsf{vk_{opn}}, \mathsf{aux}_Z)$; $b^j_V \leftarrow \bigwedge_{j\in[\mathsf{n}(i)]} \mathsf{VerCom}(\mathsf{ck}, c_{i,j}, p_{i,j}, o_{i,j})$
let $(p_j)_{j\in[\mathsf{n}^*]} := (p_{i,j})_{i\in[\mathsf{r}], j\in[\mathsf{n}(i)]}$; $b_{\mathsf{CS}} \leftarrow \forall j : p'_j = p_j$;
$b_W \leftarrow b \wedge b_V \wedge (\wedge_j b^j_V) \wedge b_{\mathsf{CS}}$;

Consider the hybrid $\mathbf{H}_4$ which is the same of $\mathbf{H}_3$ but additionally check that the linking relation holds. Recall that for a PHP has a straight-line extractor $\mathsf{WitExtract}$.

Let $\mathcal{E}_{\mathsf{link}}$ be the extractor for the CP-SNARK for the adversary $\mathcal{A}_{\mathsf{link}}$ that runs the same as the adversary $\mathcal{A}$ but that simply outputs the proof π_{link} and the statement $\big((\hat{c}_j)_{j\in[\ell]}, (c_j)_{j\in[\mathsf{n}^*]}\big)$. Formally, the adversary $\mathcal{A}_{\mathsf{link}}$ receives in input $\mathsf{R_{link}}, \mathsf{ck}, \mathsf{srs_{link}}$ and $\mathsf{aux}^{\mathsf{link}}_Z$ where $\mathsf{aux}^{\mathsf{link}}_Z$ contains all the other values, namely the elements of srs different than $\mathsf{srs_{link}}$, and $\mathsf{n_e}$ and R. In particular:

$\mathbf{H}_4(1^\lambda)$

$\mathsf{pp} \leftarrow \mathsf{ParGen}(1^\lambda)$; $\mathsf{ck} \leftarrow \mathsf{Setup}(\mathsf{pp})$; $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{ck}, \mathsf{N})$; $\mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathsf{srs})$;

$\big(\mathsf{R}, \hat{\mathsf{x}} = (\mathsf{x}, (c_j)_{j\in[\ell]}), st\big) \leftarrow \mathcal{A}_0(\mathsf{srs}, \mathsf{aux}_Z)$; $(\tau, b) \leftarrow \langle \mathcal{A}_1(st), \mathbb{V}(\mathsf{Derive}(\mathsf{vk}, \mathsf{R}), \mathsf{x}, (c_j)_{j\in[\ell]})\rangle$;

$\boxed{((\mathsf{u}_j)_{j\in[\ell]}, (\hat{o}_j)_{j\in[\ell]},\ (p''_j)_{j\in[\mathsf{n}^*]}, (o''_j)_{j\in[\mathsf{n}^*]}) \leftarrow \mathcal{E}_{\mathsf{link}}(\mathsf{R}_{\mathsf{link}}, \mathsf{ck}, \mathsf{srs}_{\mathsf{link}}, \mathsf{aux}_Z^{\mathsf{link}})}$;

$\boxed{\mathsf{w} = \bar{\mathsf{u}}, \omega \leftarrow \mathsf{WitExtract}((p''_j)_{j\in[\mathsf{n}^*]})}$;

$\boxed{b_V^{\mathsf{link}} \leftarrow \wedge_{j\in[\ell]}(\mathsf{VerCom}(\mathsf{ck}, \hat{c}_j, \hat{o}_j, \mathsf{u}_j) = 1) \wedge_{l\in[\kappa]} (\mathsf{VerCom}(\mathsf{ck}, c_{i_l, j_l}, o_l, p_l) = 1) \wedge \mathsf{Decode}((\mathsf{u}_j)_{j\in[\ell]}) = \bar{\mathsf{u}}}$;

$(p'_j, o'_j)_{j\in[\mathsf{n}^*]} \leftarrow \mathcal{E}(\mathcal{R}_{\mathsf{php}}, \mathsf{ck}, \mathsf{srs}, \mathsf{aux}_Z)$;

$b_V \leftarrow \bigwedge_k \big(G_k(X, \{p'_j(v_{k,j}(X))\}_{j\in[\mathsf{n}^*]}, \{\pi_j\}_{j\in[\mathsf{m}^*]}) \equiv 0\big) \wedge \bigwedge_k \big(\mathsf{deg}(p'_{i_k}) \le d_k\big) \wedge \bigwedge_k \mathsf{VerCom}(\mathsf{ck}, c_k, p'_k, o'_k)$;

for $j \in [\mathsf{r}] : (p_{i,j}, o_{i,j})_{j\in[\mathsf{n}(i)]} \leftarrow \mathcal{E}'_j(\mathsf{vk}_{\mathsf{opn}}, \mathsf{ck}, \mathsf{aux}_Z)$; $b_V^j \leftarrow \bigwedge_{j\in[\mathsf{n}(i)]} \mathsf{VerCom}(\mathsf{ck}, c_{i,j}, p_{i,j}, o_{i,j})$

let $(p_j)_{j\in[\mathsf{n}^*]} := (p_{i,j})_{i\in[\mathsf{r}], j\in[\mathsf{n}(i)]}$; $b_{\mathsf{CS}} \leftarrow \forall j : p'_j = p_j \boxed{= p''_j}$;

$b_W \leftarrow b \wedge b_V \wedge (\wedge_j b_V^j) \wedge b_{\mathsf{CS}} \boxed{\wedge b_V^{\mathsf{link}}}$;

Let $\epsilon_{\mathsf{snark}}^{\mathsf{link}}$ be the knowledge soundness error of $\mathsf{CP}_{\mathsf{link}}$.

Lemma C.10. $\Pr[b_W^4] \le \Pr[b_W^3] + \epsilon_{\mathsf{snark}}^{\mathsf{link}} + \epsilon_{\mathsf{CS}}$.

The proof follows almost identically to Lemma C.1 and Lemma C.3, specifically we can reduce to the knowledge soundness of $\mathsf{CP}_{\mathsf{link}}$ or the binding property of CS, the proof of the lemma is therefore omitted. Consider the following PPT sampler algorithm:

Sampler $\mathcal{D}(1^\lambda)$:

$\mathsf{pp} \leftarrow \mathsf{ParGen}(1^\lambda)$; $\mathsf{ck} \leftarrow \mathsf{Setup}(\mathsf{pp})$; $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{ck}, \mathsf{N})$; $\mathsf{aux}_Z \leftarrow \mathcal{Z}(\mathsf{srs})$;

$\big(\mathsf{R}, \mathsf{x}, st\big) \leftarrow \mathcal{A}_0(\mathsf{R}, \mathsf{ck}, \mathsf{srs}, \mathsf{aux}_R, \mathsf{aux}_Z)$; $\mathsf{aux}'_Z := (\mathsf{R}, \mathsf{ck}, \mathsf{srs}, \mathsf{aux}_R, \mathsf{aux}_Z)$; **return** $(\mathsf{R}, \mathsf{x}, \mathsf{aux}'_Z)$

Similarly to the proof of Thm. 6.1.

Lemma C.11. *There exists a prover $\mathcal{P}^*$ such that*

$$\Pr[\langle \mathcal{P}^*(\mathbb{F}, \mathsf{R}, \mathsf{x}, z), \mathcal{V}^{\mathcal{RE}(\mathbb{F}, \mathsf{R})}(\mathbb{F}, \mathsf{x})\rangle = 1 : (\mathsf{R}, \mathsf{x}, \mathsf{aux}'_Z) \leftarrow_\$ \mathcal{D}(1^\lambda)] = \Pr[b_W^4] \tag{37}$$

We define $\mathcal{P}^*$ to be the prover that emulates $\mathbf{H}_4$ almost identically as done in Thm. 6.1, the proof follows similarly thus is omitted. By the knowledge soundness of the PHP with straight-line extractor we have that

$$\Pr[(\mathsf{R}, \mathsf{x}, \mathsf{WitExtract}((p_j)_{j\in[\mathsf{n}^*]})) \in \mathsf{R} : (\mathsf{R}, \mathsf{x}, \mathsf{aux}'_Z) \leftarrow_\$ \mathcal{D}(1^\lambda)] + \epsilon_{\mathsf{PHP}} \ge \Pr[b_W^4].$$

We are ready to define the extractor $\mathcal{E}^*$. Let $\mathcal{E}^*(\mathsf{R}, \mathsf{srs}, \mathsf{aux}_R, \mathsf{aux}_Z)$ be the algorithm that:

1. computes $((\hat{p}_j)_{j\in[\ell]}, (\hat{o}_j)_{j\in[\ell]}, (p_j)_{j\in[\mathsf{n}^*]}, (o_j)_{j\in[\mathsf{n}^*]}) \leftarrow \mathcal{E}_{\mathsf{link}}(\mathsf{R}_{\mathsf{link}}, \mathsf{ck}, \mathsf{srs}_{\mathsf{link}}, \mathsf{aux}_R, \mathsf{aux}_Z^{\mathsf{link}})$;
2. outputs $((\hat{p}_j)_{j\in[\ell]}, (\hat{o}_j)_{j\in[\ell]}, \mathsf{WitExtract}((p_j)_{j\in[\mathsf{n}^*]}))$.

Assuming that b_W^4 is true then we have $(p_j)_{j\in[\mathsf{n}^*]} = (p''_j)_{j\in[\mathsf{n}^*]}$ in $\mathbf{H}_4$ (thus by definition of the extractor the polynomial extracted by $\mathcal{E}_{\mathsf{link}}$ are the same as the one sent by $\mathcal{P}^*$) and that for any j the opening $\hat{o}_j$ is a valid opening for the polynomial $\hat{p}_j$, moreover by straight-line extractability we have that $\mathsf{WitExtract}((p_j)_{j\in[\mathsf{n}^*]})$ is a valid witness.