An Algebraic Framework for Universal and Updatable SNARKs

Carla Ràfols^{*,1,2} and Arantxa Zapico^{**,1}

[1] Universitat Pompeu Fabra
[2] Cybercat

Abstract. We introduce Checkable Subspace Sampling Arguments, a new information theoretic interactive proof system in which the prover shows that a vector has been sampled in a subspace according to the verifier's coins. We show that this primitive provides a unifying view that explains the technical core of most of the constructions of universal and updatable pairing-based (zk)SNARKs. This characterization is extended to a fully algebraic framework for designing such SNARKs in a modular way. We propose new constructions of CSS arguments that lead to SNARKs with different performance trade-offs.

1 Introduction

Zero-Knowledge proofs [23], and in particular, non-interactive ones [7] have played a central role in both the theory and practice of cryptography. A long line of research [32,34,25,22,26] has led to efficient pairing-based zero-knowledge *Succinct Non-interactive ARguments of Knowledge* or SNARKs. These arguments are *succinct*, in fact, they allow to prove that circuits of arbitrary size are satisfied with a constant-size proof. They are also extremely efficient concretely (3 group elements in the best construction for arithmetic circuits [26]).

Despite this impressive performance, some aspects of these constructions of SNARKs are still unsatisfactory. Probably the most problematic and not fully solved issue is their reliance on long trusted, structured, and circuit dependent parameters (a circuit dependent SRS, for *structured reference string*).

Albeit the significant research effort in finding alternatives to bypass the need of a trusted third party by constructing *transparent* arguments, i.e in the uniform random string model (URS) [11,8,3,2,4,15,40,39], pairing-based SNARKs such as [26] still seem the most practical alternative in many settings due to their very fast verification, which is a must in many blockchain applications. On the other hand, multiparty solutions for the problem are not fully scalable [9,10].

* This paper is part of a project that has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 856879.

** The project that gave rise to these results received the support of a fellowship from la Caixa Foundation (ID100010434). The fellowship code is LCF/BQ/DI18/11660052. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skodowska-Curie grant agreement No.71367.

As an alternative to a trusted SRS, Groth et al. [27] define the updatable model, in which the SRS can be updated by any party, non-interactively, and in a verifiable way, resulting in a properly generated structured reference string where the simulation trapdoor is unknown to all parties if at least one is honest. Further, they propose a construction where the SRS is universal and can be used for arbitrary circuits up to a maximum given size.

Arithmetic Circuit Satisfiability can be reduced to a set of quadratic and affine constraints over a finite field. The quadratic ones are universal and can be easily proven in the pairing-based setting with a Hadamard product argument, the basic core of most zkSNARKS constructions starting from [22]. On the other hand, affine constraints are circuit-dependent, and it is a challenging task to efficiently prove them with a universal SRS [33,14,20,13,16,37,19,38].

In Groth et al. [27] they are proven via a very expensive subspace argument that requires a SRS quadratic in the circuit size and a preprocessing step that is cubic. Sonic [33], the first efficient, universal, and updatable SNARK, gives two different ways to prove the affine constraints, a fully succinct one (not so efficient) and another one in the amortized setting (very efficient). Follow-up work (most notably, Marlin [14], Plonk [20], Lunar [13]) has significantly improved the efficiency in the fully succinct mode.

There is an important trend in cryptography, that advocates for constructing protocols in a modular way. One reason for doing so is the fact that, by breaking complicated protocols into simpler steps, they become easier to analyze. Ishai [28] mentions comparability as another fundamental motive. Specially in the area of zero-knowledge, given the surge of interest in practical constructions, it is hard not to lose sight of what each proposal achieves. As Ishai puts it: "*one reason such comparisons are difficult is the multitude of application scenarios, implementation details, efficiency desiderata, cryptographic assumptions, and trust models*".

Starting from Sonic, all the aforementioned works on universal and updatable zkSNARKs follow this trend. More concretely, they first build an information-theoretic proof system, that is then compiled into a full argument under some computational assumptions in bilinear groups. The main ingredient of the compiler is a polynomial commitment [30,12,31]. However, the information theoretic component is still very complex and comparison among these works remains difficult, for precisely the same reasons stated by Ishai. In particular, it is hard to extract the new ideas in each of them in the complex description of the arguments, that use sophisticated tricks for improving efficiency, as well as advanced properties of multiplicative subgroups of a finite field or bivariate Lagrange interpolation. Further, it is striking that all fully succinct arguments are for restricted types of constraints (sums of permutations in Sonic, sparse matrices in Marlin, and Lunar[3]) or pay a price for additive gates (Plonk). A modular, unified view of these important works seems essential for a clearer understanding of the tech-

[3] The number of non-zero entries of the matrices that encode linear constraints cannot exceed the size of some multiplicative group of the field of definition.

niques. In turn, this should allow for a better comparison, more flexibility in combining the different methods, and give insights on current limitations.

Our Contributions. We propose an algebraic framework that takes a step further in achieving modular constructions of universal and updatable SNARKs. We identify the technical core of previous work as instances of a *Checkable Subspace Sampling* (CSS) Argument. In this information-theoretic proof system, two parties, prover and verifier, on input a field $\mathbb{F}$ and a matrix $\mathbf{M} \in \mathbb{F}^{Q \times m}$, agree on a polynomial $D(X)$ encoding a vector $\boldsymbol{d}$ in the row space of $\mathbf{M}$. The interesting part is that, even though the coefficients of the linear combination that define $\boldsymbol{d}$ are chosen by the verifier's coins, the latter does not need to perform a linear (in Q, the number of rows) number of operations in order to verify that $D(X)$ is correct. Instead, this must be demostrated by the prover.

With this algebraic formulation, it is immediate to see that a CSS argument can be used as a building block for an argument of membership in linear spaces. Basically, given a matrix $\mathbf{M}$, we can prove that some vector $\boldsymbol{y}$ is orthogonal to the rows of $\mathbf{M}$ by sampling, after $\boldsymbol{y}$ is declared, a *sufficiently random* vector $\boldsymbol{d}$ in the row space of $\mathbf{M}$ and checking an inner product relation, namely, whether $\boldsymbol{d} \cdot \boldsymbol{y} = 0$. The purpose of a CSS argument is to guarantee that the sampling process can be checked by the verifier in sublinear time without sacrificing soundness.

Naturally, for building succinct proofs, instead of $\boldsymbol{y}, \boldsymbol{d}$, the argument uses polynomial encodings $Y(X)$ and $D(X)$ (which are group elements after the compilation step). To compute the inner product of this encoded vectors, we introduce a new argument in Section 3, which is specific to the case where the polynomials are encoded in the Lagrange polynomial basis, but can be easily generalized to the monomial basis. The argument is a straightforward application of the univariate sumcheck of Aurora [5]. However, we contribute a generalized sumcheck (that works not only for multiplicative subgroups of finite fields), with a completely new proof that relates it with polynomial evaluation at some fixed point v.

These building blocks can be put together as an argument for the language of Rank1 constraint systems. For efficiency, we stick to R1CS-lite, a variant recently proposed by Lunar, which is slightly simpler but still NP-complete. Our final construction can be instantiated with any possible choice of CSS scheme, so in particular it can essentially recover the construction of Marlin and Lunar by isolating the CSS argument implicit in these works, or the amortized construction of Sonic. We hope that this serves to better identify the challenge behind building updatable and universal SNARKs, and allow for new steps in improving efficiency, as well as more easily combining the techniques.

In summary, we reduce R1CS constraint systems to three algebraic relations: an inner product, a Hadamard product and a CSS argument. We think this algebraic formulation is very clear, and also makes it easier to relate advances in universal and updatable SNARKS with other works that have used a similar language, for example, the arguments for inner product of [8], of membership in linear spaces [29], or for linear algebra relations [24].

Finally, we give several constructions of CSS arguments. In Section 5.3, we start from the representation of a matrix $\mathbf{W}$ as bivariate polynomial introduced in [14], and present an alternative that comes from applying a linearization step to it. The result is a CSS for sparse matrices, that compared to [14,13], at a minimial increase in communication cost, significantly reduces the SRS. We study several extensions of this argument, for example, to sums of sparse matrices. We also identify a simple building block that allows for a modular construction. In the full version we discuss how these CSS arguments result in zkSNARKs with different performance trade-offs.

1.1 Related Work

Bivariate Polynomial Evaluation Arguments. As mentioned before, the complexity of building updatable and universal zkSNARKs protocols is mainly caused by proving affine constraints. A natural way to encode them is through a bivariate public polynomial $P(X, Y)$; in order to avoid having a quadratic SRS, this polynomial can only be given to the verifier evaluated or partially evaluated in the field. The common approach is to let the verifier chose arbitrary field elements x, y and having the prover evaluate and send $\sigma = P(y, x)$. The challenge is to prove that the evaluation has been performed correctly. In Sonic [33], this last step is called a *signature of correct computation* [36] and can be performed by the prover or by the verifier with some help from an untrusted third party. The drawback of the first construction is that, while still linear, prover's work is considerably costly; also, linear constraints are assumed to be sparse and the protocol works exclusively for a very particular polynomial $P(X, Y)$. The second construction is interesting only in some restricted settings where the same verifier checks a linear amount of proofs for one circuit. Marlin [14] bases its construction on the univariate sum-check protocol of Aurora [5] and presents a novel way to navigate from the naive quadratic representation $P(X, Y)$ to a linear one. This approach results in succinct prover and verifier work, but restricts their protocol to the case where the number of non-zero entries of matrix $\mathbf{W}$ is bounded by the size of some multiplicative subgroup of the field of definition. Lunar [13] uses the same representation as Marlin but improves on it, among other tweaks by introducing a new language (R1CS-lite) that can also represent arithmetic circuit satisfiability, but has a lighter representation than other constraint systems. Plonk [20] does not use bivariate polynomials or require sparse matrices but the SRS size depends on the number of both multiplicative and additive gates. Plonk, Marlin and Lunar use the Lagrange interpolation basis to commit to vectors. Claymore [38] presents a modular construction for zkSNARKs based on similar algebraic building blocks but in the monomial basis: inner product, Hadamard product and matrix-vector product arguments. The latter also uses an implicit CSS argument.

Information Theoretic Proof Systems. These previous works all follow the two step process described in the introduction and build their succinct argument by compiling an information theoretically secure one. Marlin introduces

Algebraic Holographic proofs, that are variation of interactive oracle proofs (IOPs) [6]. Holographic refers to the fact that the verifier never receives the input explicitly (otherwise, succinctness would be impossible), but rather its encoding as an oracle computed by an indexer or encoder. The term algebraic refers to the fact that oracles are low degree polynomials, and malicious provers are also bound to output low degree polynomials. This is similar to the notion of Idealised Low Degree protocols of Plonk. Lunar refines this model by introducing Polynomial Holographic IOPs, which generalize these works mostly by allowing for a fine grained analysis of the zero-knowledge property, including degree checks, and letting prover and verifier send field elements.

Polynomial Commitments. Polynomial commitments allow to commit to a polynomial $p(X) \in \mathbb{F}[X]$, and open it at any point $x \in \mathbb{F}$. As it is common, we will use a polynomial commitment based on the one by Kate et al. [30]. Sonic gave a proof of extractability of the latter in the Algebraic Group Model [18], and Marlin completed the proof to make the commitments usable as a standalone primitive, and also have an alternative construction under knowledge assumptions. Both Marlin and Plonk considered versions of polynomial commitments where queries in the same point can be batched together. For this work, we use the definitions presented in [14].

Work		$\|\mathsf{srs}_\mathsf{u}\|$	$\|\mathsf{srs}_\mathbf{W}\|$	$\|\pi\|$	KeyGen	Derive	Prove	Verifier
Sonic [33]	$\mathbb{G}_1$	$4N$	-	20	$4N$	$36n$	$273n$	7P
	$\mathbb{G}_2$	$4N$	3	-	$4N$	-	-	
	$\mathbb{F}$	-	-	16	-	$O(m \log m)$	$O(m \log m)$	$O(l + \log m)$
Plonk [20]	$\mathbb{G}_1$	$3N^*$	8	7	$3N^*$	$8n + 8a$	$11n + 11a$	2P
	$\mathbb{G}_2$	1	1	-	-	-	-	
	$\mathbb{F}$	-	-	7	-	$O((n+a)\log(n+a))$	$O((n+a)\log(n+a))$	$O(l + \log(n+a))$
Marlin [14]	$\mathbb{G}_1$	$3M$	12	13	$3M$	$12m$	$14n + 8m$	2P
	$\mathbb{G}_2$	2	2	-	-	-	-	
	$\mathbb{F}$	-	-	8	-	$O(m \log m)$	$O(m \log m)$	$O(l + \log m)$
Lunar [13]	$\mathbb{G}_1$	M	-	10	M	-	$8n + 3m$	7P
	$\mathbb{G}_2$	M	27	-	M	$24m$	-	
	$\mathbb{F}$	-	-	2	-	$O(m \log m)$	$O(m \log m)$	$O(l + \log m)$
This work	$\mathbb{G}_1$	M	4	11	M	$6m$	$8n + 4m$	2P
	$\mathbb{G}_2$	1	1	-	-	-	-	
	$\mathbb{F}$	-	-	4	-	$O(m \log m)$	$O(m \log m)$	$O(l + \log m)$

Comparison with state of the art universal and updatable zkSNARKs. n: number of multiplicative gates, a: number of additive gates, $m = |\mathbf{F}| + |\mathbf{G}|$, where $\mathbf{F}, \mathbf{G}$ are the matrices that describe the linear relations for the left and right inputs, respectively. N, A, M: maximum supported values for n, a, m. $N^* = M + A$.

Untrusted Setup. The original constructions of pairing-based zkSNARKs crucially depend for soundness on a trusted setup, although, as was shown in [1,17], the zero-knowledge property is still easy to achieve when the setup is subverted. Groth et al. introduced the updatable SRS model in [27] to address the issue of trust in SRS generation. There are several alternatives to achieve transparent setup and constant-size proofs, but all of them have either linear verifier [8,11,5,2], or work only for very structured types of computation [3,39]. An exception is the work of Setty [37]. Concretely, its approach is less efficient in terms of proof size and verification complexity compared to recent constructions of updatable and universal pairing-based SNARKs.

2 Preliminaries

A bilinear group gk is a tuple $gk = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathcal{P}_1, \mathcal{P}_2)$ where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order q, the elements $\mathcal{P}_1, \mathcal{P}_2$ are generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable, non-degenerate bilinear map, and there is an efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. Elements in $\mathbb{G}_\gamma$, are denoted implicitly as $[a]_\gamma = a\mathcal{P}_\gamma$, where $\gamma \in \{1, 2, T\}$ and $\mathcal{P}_T = e(\mathcal{P}_1, \mathcal{P}_2)$. With this notation, $e([a]_1, [b]_2) = [ab]_T$.

For $n \in \mathbb{N}$, $[n]$ is the set of integers $\{1, \ldots, n\}$. Vectors and matrices are denoted in boldface. Given two vectors $\boldsymbol{a}, \boldsymbol{b}$, their Hadamard product is denoted as $\boldsymbol{a} \circ \boldsymbol{b}$, and their inner product as $\boldsymbol{a} \cdot \boldsymbol{b}$. The subspace of polynomials of degree at most d in $\mathbb{F}[X]$ is denoted as $\mathbb{F}_{\leq d}[X]$. Given a matrix $\mathbf{M}$, $|\mathbf{M}|$ refers to the number of its non-zero entries.

2.1 Constraint Systems

Formally, we will construct an argument for the universal relation $\mathcal{R}'_{\text{R1CS-}\mathsf{lite}}$, an equivalent of the relation $\mathcal{R}_{\text{R1CS-}\mathsf{lite}}$ introduced in Lunar [13]. The latter is a simpler version of Rank 1 Constraint Systems, it is still NP complete and encodes circuit satisfiability in a natural way:

Definition 1. *(R1CS-*lite*) Let $\mathbb{F}$ be a finite field and $m, l, s \in \mathbb{N}$. We define the universal relation R1CS-*lite *as:*

$$\mathcal{R}_{\text{R1CS-}\mathsf{lite}} = \left\{ \begin{array}{c} (\mathsf{R}, \mathsf{x}, \mathsf{w}) := \big((\mathbb{F}, s, m, l, \mathbf{F}, \mathbf{G}), \boldsymbol{x}, \boldsymbol{w}\big) : \\ \mathbf{F}, \mathbf{G} \in \mathbb{F}^{m \times m}, \boldsymbol{x} \in \mathbb{F}^{l-1}, \boldsymbol{w} \in \mathbb{F}^{m-l}, s = \max\{|\mathbf{F}|, |\mathbf{G}|\}, \\ \textit{and for } \boldsymbol{c} := (1, \boldsymbol{x}, \boldsymbol{w}), (\mathbf{F}\boldsymbol{c}) \circ (\mathbf{G}\boldsymbol{c}) = \boldsymbol{c} \end{array} \right\}.$$

As an equivalent formulation of this relation, we use the following:

$$\mathcal{R}'_{\text{R1CS-}\mathsf{lite}} = \left\{ \begin{array}{c} (\mathsf{R}, \mathsf{x}, \mathsf{w}) := \big((\mathbb{F}, s, m, l, \mathbf{F}, \mathbf{G}), \boldsymbol{x}, (\boldsymbol{a}', \boldsymbol{b}')\big) : \mathbf{F}, \mathbf{G} \in \mathbb{F}^{m \times m}, \boldsymbol{x} \in \mathbb{F}^{l-1}, \\ \boldsymbol{a}', \boldsymbol{b}' \in \mathbb{F}^{m-l}, s = \max\{|\mathbf{F}|, |\mathbf{G}|\}, \text{ and for } \boldsymbol{a} := (1, \boldsymbol{x}, \boldsymbol{a}'), \boldsymbol{b} := (\mathbf{1}, \boldsymbol{b}') \\ \begin{pmatrix} \mathbf{I} & \mathbf{0} & -\mathbf{F} \\ \mathbf{0} & \mathbf{I} & -\mathbf{G} \end{pmatrix} \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{b} \\ \boldsymbol{a} \circ \boldsymbol{b} \end{pmatrix} = \mathbf{0} \end{array} \right\}.$$

To see they are equivalent, observe that, if in $\mathcal{R}'_{\text{R1CS-}\mathsf{lite}}$ we define the vector $\boldsymbol{c} = \boldsymbol{a} \circ \boldsymbol{b}$, the linear equation reads as $\boldsymbol{a} = \mathbf{F}\boldsymbol{c}$ and $\boldsymbol{b} = \mathbf{G}\boldsymbol{c}$. A formal proof is a direct consequence of the proof that arithmetic circuit satisfiability reduces to R1CS-lite found in Lunar([13]).

2.2 zkSNARKs

Let $\mathcal{R}$ be a family of universal relations. Given a relation $\mathsf{R} \in \mathcal{R}$ and an instance x we call w a *witness* for x if $(\mathsf{x}, \mathsf{w}) \in \mathsf{R}$, $\mathcal{L}(\mathsf{R}) = \{\mathsf{x} | \ \exists \mathsf{w} : (\mathsf{x}, \mathsf{w}) \in \mathsf{R}\}$ is the language of all the x that have a witness w in the relation R, while $\mathcal{L}(\mathcal{R})$is the language of all the pairs (x, R) such that $\mathsf{x} \in \mathcal{L}(\mathsf{R})$.

Definition 2. *A Universal Succinct Non-Interactive Argument of Knowledge is a tuple of PPT algorithms* $(\mathsf{KeyGen}, \mathsf{KeyGenD}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Simulate})$ *such that:*

- $(\mathsf{srs_u}, \tau) \leftarrow \mathsf{KeyGen}(\mathcal{R})$*: On input a family of relations* $\mathcal{R}$*,* KeyGen *outputs a universal structured common reference string* $\mathsf{srs_u}$ *and a trapdoor* τ*;*
- $\mathsf{srs_R} \leftarrow \mathsf{KeyGenD}(\mathsf{srs_u}, \mathsf{R})$*: On input* $\mathsf{R} \in \mathcal{R}$*, this algorithm outputs a relation dependent SRS that includes* $\mathsf{srs_u}$*;*
- $\pi \leftarrow \mathsf{Prove}(\mathsf{R}, \mathsf{srs_R}, (\mathsf{x}, \mathsf{w}))$*: On input the relation,* $\mathsf{srs_R}$ *and a pair* $(\mathsf{x}, \mathsf{w}) \in \mathsf{R}$*, it outputs a proof* π*;*
- $1/0 \leftarrow \mathsf{Verify}(\mathsf{srs_R}, \mathsf{x}, \pi)$*:* Verify *takes as input* $\mathsf{srs_R}$*, the instance* x *and the proof and produces a bit expressing acceptance* (1)*, or rejection* (0)*;*
- $\pi_{\text{sim}} \leftarrow \mathsf{Simulate}(\mathsf{R}, \tau, \mathsf{x})$*: The simulator has the relation* R*, the trapdoor* τ *and the instance* x *as inputs and it generates a simulated proof* π_{sim}*,*

and that satisfies completeness, succinctness and ϵ*-knowledge soundness as defined below.*

Definition 3. *Completeness holds if an honest prover will always convince an honest verifier. Formally,* $\forall\ \mathsf{R} \in \mathcal{R}, (\mathsf{x}, \mathsf{w}) \in \mathsf{R}$*,*

$$\Pr\left[\mathsf{Verify}(\mathsf{srs_R}, \mathsf{x}, \pi) = 1 \;\middle|\; \begin{array}{l} (\mathsf{srs_u}, \tau) \leftarrow \mathsf{KeyGen}(\mathcal{R}) \\ \mathsf{srs_R} \leftarrow \mathsf{KeyGenD}(\mathsf{srs_u}, \mathsf{R}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{R}, \mathsf{srs_R}, (\mathsf{x}, \mathsf{w})) \end{array}\right] = 1.$$

Definition 4. *Succinctness holds if the size of the proof* π *is poly*$(\lambda + \log |\mathsf{w}|)$ *and* Verify *runs in time poly*$(\lambda + |\mathsf{x}| + \log |\mathsf{w}|)$*.*

Definition 5. ϵ*-knowledge soundness captures the fact that a cheating prover cannot, except with probability at most* ϵ*, create a proof* π *accepted by the verification algorithm unless it has a witness* w *such that* $(\mathsf{x}, \mathsf{w}) \in \mathsf{R}$*. Formally, for all PPT adversaries* $\mathcal{A}$*, there exists a PPT extractor* $\mathcal{E}$ *such that:*

$$\Pr\left[(\mathsf{x}, \mathsf{w}) \notin \mathsf{R} \wedge \mathsf{Verify}(\mathsf{srs_R}, \mathsf{x}, \pi) = 1 \;\middle|\; \begin{array}{l} (\mathsf{srs_u}, \tau) \leftarrow \mathsf{KeyGen}(\mathcal{R}) \\ \mathsf{R} \leftarrow \mathcal{A}(\mathsf{srs_u}) \\ \mathsf{srs_R} \leftarrow \mathsf{KeyGenD}(\mathsf{srs_u}, \mathsf{R}) \\ (\mathsf{x}, \pi) \leftarrow \mathcal{A}(\mathsf{R}, \mathsf{srs_R}) \\ \mathsf{w} \leftarrow \mathcal{E}(\mathsf{srs_R}, \mathsf{x}, \pi) \end{array}\right] \leq \epsilon.$$

Definition 6. $(\mathsf{KeyGen}, \mathsf{KeyGenD}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Simulate})$ *is zero-knowledge (a zk-SNARK) if for all* $\mathsf{R} \in \mathcal{R}$*, instances* x *and PPT adversaries* $\mathcal{A}$*.*

$$\Pr\left[\mathcal{A}(\mathsf{R}, \mathsf{srs_R}, \pi) = 1 \;\middle|\; \begin{array}{l} (\mathsf{srs_u}, \tau) \leftarrow \mathsf{KeyGen}(\mathcal{R}) \\ \mathsf{srs_R} \leftarrow \mathsf{KeyGenD}(\mathsf{srs_u}, \mathsf{R}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{R}, \mathsf{srs_R}, (\mathsf{x}, \mathsf{w})) \end{array}\right] \approx$$

$$\Pr\left[\mathcal{A}(\mathsf{R},\mathsf{srs}_\mathsf{R},\pi_{sim}) = 1 \;\middle|\; \begin{array}{l} (\mathsf{srs}_\mathsf{u},\tau) \leftarrow \mathsf{KeyGen}(\mathcal{R}) \\ \mathsf{srs}_\mathsf{R} \leftarrow \mathsf{KeyGenD}(\mathsf{srs}_\mathsf{u},\mathsf{R}) \\ \pi_{sim} \leftarrow \mathsf{Simulate}(\mathsf{R},\tau,\mathsf{x}) \end{array}\right].$$

Updatability. We will say a universal zkSNARK is *updatable* if srs_u is updatable as defined in [21]. We remark their result states that this is the case if srs_u consists solely of monomials.

2.3 Polynomial Holographic Proofs

In this paper, we use the notion of Polynomial Holographic Interactive Oracle Proofs (PHP), recently introduced by Campanelli et al. [13]. It is a refinement and quite similar to other notions used in the literature to construct SNARKs in a modular way, such as Algebraic Holographic Proofs (AHP) [14] or idealized polynomial protocols [20].

A proof system for a relation R is holographic if the verifier does not read the full description of the relation, but rather has access to an encoding of the statement produced by some holographic relation encoder, also called indexer, that outputs oracle polynomials. In all these models, the prover is restricted to send oracle polynomials or field elements, except that, for additional flexibility, the PHP model of [13] also lets the prover send arbitrary messages. In PHPs, the queries of the verifier are algebraic checks over the polynomials sent by the verifier, as opposed to being limited to polynomial evaluations as in AHPs.

The following definitions are taken almost verbatim from [13].

Definition 7. *A family of polynomial time computable relations $\mathcal{R}$ is field dependent if each relation $\mathsf{R} \in \mathcal{R}$, specifies a unique finite field. More precisely, for any pair $(\mathsf{x},\mathsf{w}) \in \mathsf{R}$, x specifies the same finite field $\mathbb{F}_\mathsf{R}$ (simply denoted as $\mathbb{F}$ if there is no ambiguity).*

Definition 8 (Polynomial Holographic IOPs (PHP)). *A Polynomial Holographic IOP for a family of field-dependent relations $\mathcal{R}$ is a tuple $\mathsf{PHP} = (\mathsf{rnd}, \mathsf{n}, \mathsf{m}, \mathsf{d}, \mathsf{n}_e, \mathcal{I}, \mathcal{P}, \mathcal{V})$, where $\mathsf{rnd}, \mathsf{n}, \mathsf{m}, \mathsf{d}, \mathsf{n}_e : \{0,1\}^* \rightarrow \mathbb{N}$ are polynomial-time computable functions, and $\mathcal{I}, \mathcal{P}, \mathcal{V}$ are three algorithms that work as follows:*

- ***Offline phase:*** *The encoder or indexer $\mathcal{I}(\mathsf{R})$ is executed on a relation description R, and it returns $\mathsf{n}(0)$ polynomials $\{p_{0,j}\}_{j=1}^{\mathsf{n}(0)} \in \mathbb{F}[X]$ encoding the relation R and where $\mathbb{F}$ is the field specified by R.*
- ***Online phase:*** *The prover $\mathcal{P}(\mathsf{R},\mathsf{x},\mathsf{w})$ and the verifier $\mathcal{V}^{\mathcal{I}(\mathsf{R})}(\mathsf{x})$ are executed for $\mathsf{rnd}(|\mathsf{R}|)$ 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 $\{\pi_{i,j} \in \mathbb{F}\}_{j=1}^{\mathsf{m}(i)}$, and $\mathsf{n}(i)$ oracle polynomials $\{p_{i,j} \in \mathbb{F}[X]\}_{j=1}^{\mathsf{n}(i)}$, such that $\deg(p_{i,j}) < \mathsf{d}(|\mathsf{R}|, i, j)$.*
- ***Decision phase:*** *After the $\mathsf{rnd}(|\mathsf{R}|)$-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* $\mathsf{n}_p = \sum_{k=1}^{\mathsf{rnd}(|\mathsf{R}|)} \mathsf{n}(k)$ *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 satisfies the following condition*

$$\forall k \in [\mathsf{n}_p] : \deg(p_k) \leq d_k.$$

- *Polynomial checks: to verify that certain polynomial identities hold between the oracle polynomials and the messages sent by the prover. Let* $\mathsf{n}^* = \sum_{k=0}^{\mathsf{rnd}(|\mathsf{R}|)} \mathsf{n}(k)$ *and* $\mathsf{m}^* = \sum_{k=0}^{\mathsf{rnd}(|\mathsf{R}|)} \mathsf{m}(k)$*, and denote by* $(p_1, \ldots, p_{\mathsf{n}^*})$ *and* $(\pi_1, \ldots, \pi_{\mathsf{n}^*})$ *all the oracle polynomials (including the* $\mathsf{n}(0)$ *ones frrom the encoder) and all the messages sent by the prover. The verifier can specify a list of* n_e *tuples, each of the form* $(G, v_1, \ldots, v_{\mathsf{n}^*})$*, where* $G \in \mathbb{F}[X, X_1, \ldots, X_{\mathsf{n}^*}, Y_1, \ldots, Y_{\mathsf{m}^*}]$ *and every* $v_k \in \mathbb{F}[X]$*. Then a tuple* $(G, v_1, \ldots, v_{\mathsf{n}^*})$ *is satisfied if and only if* $F(X) \equiv 0$ *where*

$$F(X) := G\big(X, \{p_k(v_k(X))\}_{k=1,\ldots,\mathsf{n}^*}, \{\pi_k\}_{k=1,\ldots,\mathsf{m}^*}\big).$$

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

Definition 9. *A* PHP *is complete if for any triple* $(\mathsf{R}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$*, the checks returned by* $\mathcal{V}^{\mathcal{I}(\mathsf{R})}$ *after interacting with the honest prover* $\mathcal{P}(\mathsf{R}, \mathsf{x}, \mathsf{w})$*, are satisfied with probability 1.*

Definition 10. *A* PHP *is* ϵ*-sound if for every relation-instance tuple* $(\mathsf{R}, \mathsf{x}) \notin \mathcal{L}(\mathcal{R})$ *and polynomial time prover* $\mathcal{P}^*$ *we have*

$$\Pr\left[\langle \mathcal{P}^*, \mathcal{V}^{\mathcal{I}(\mathsf{R})}(\mathsf{x})\rangle = 1\right] \leq \epsilon.$$

Definition 11. *A* PHP *is* ϵ*-knowledge sound if there exists a polynomial time knowledge extractor* $\mathcal{E}$ *such that for any prover* $\mathcal{P}^*$*, relation* R*, instance* x *and auxiliary input* z *we have*

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

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

Definition 12. *A* PHP *is* ϵ*-zero-knowledge if there exists a PPT simulator* $\mathcal{S}$ *such that for 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:*

$$\mathsf{View}\left(\mathcal{P}(\mathsf{R}, \mathsf{x}, \mathsf{w}), \mathcal{V}^*\right) \approx_c \mathsf{View}\left(\mathcal{S}^{\mathcal{V}^*}(\mathsf{R}, \mathsf{x})\right),$$

where $\mathsf{View}\left(\mathcal{P}(\mathsf{R}, \mathsf{x}, \mathsf{w}), \mathcal{V}^*\right)$ *consists of* $\mathcal{V}^*$*'s randomness,* $\mathcal{P}$*'s messages (which do not include the oracles) and* $\mathcal{V}^*$*'s list of checks, while* $\mathsf{View}(\mathcal{S}^{\mathcal{V}^*}(\mathsf{R}, \mathsf{x}))$ *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.*

We assume that in every PHP scheme there is an implicit maximum degree for all the polynomials used in the scheme. Thus, we include only degree checks that differ from this maximum. In all our PHPs, the verifier is public coin.

The following definition captures de fact that zero-knowledge should hold even when the verifier has access to *a bounded amount* of evaluations of the polynomials that contain information about the witness. Let $\mathcal{Q}$ be a list of queries; we say that $\mathcal{Q}$ is (b, C)-bounded for $\mathsf{b} \in \mathbb{N}^{\mathsf{n}_p}$ and C a PT algorithm, if for every $i \in [\mathsf{n}_p]$, $|\{(i, z) : (i, z) \in \mathcal{Q}\}| \leq \mathsf{b}_i$, and for all $(i, z) \in \mathcal{Q}$, $\mathsf{C}(i, z) = 1$.

Definition 13. *A* 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* $\mathcal{Q}$*, the follow random variables are within* ϵ *statistical distance:*

$$\big(\mathsf{View}\big(\mathcal{P}(\mathbb{F}, \mathsf{R}, \mathsf{x}, \mathsf{w}), \mathcal{V}\big), (p_i(z))_{(i,z)\in\mathcal{Q}}\big) \approx_\epsilon \mathcal{S}\left(\mathbb{F}, \mathsf{R}, \mathsf{x}, \mathcal{V}(\mathbb{F}, \mathsf{x}), \mathcal{Q}\right),$$

where the $p_i(X)$ *are the polynomials returned by the prover.*

Definition 14. *A* PHP *is honest-verifier zero-knowledge with query bound* b *if there exists a PT algorithm* C *such that* PHP *is* (b, C)*-zero-knowledge and for all* $i \in \mathbb{N}$*,* $Pr[\mathsf{C}(i, z) = 0]$ *is negligible, where* z *is uniformly sampled over* $\mathbb{F}$*.*

2.4 Cryptographic Assumptions

Once we compile the PHP through a polynomial commitment into a zkSNARK, the latter will achieve its security properties in the Algebraic Group Model of Fuchsbauer et al. ([18]). In this model adversaries are restricted to be *algebraic*, namely, when an adversary $\mathcal{A}$ gets some group elements as input and outputs another group element, it can provide some algebraic representation of the latter in terms of the former.

Definition 15 (Algebraic Adversary). *Let* $\mathbb{G}$ *be a cyclic group of order* p*. We say that a PPT adversary* $\mathcal{A}$ *is algebraic if there exists an efficient extractor* $\mathcal{E}_\mathcal{A}$ *that, given the inputs* $([x_1], \ldots, [x_m])$ *of* $\mathcal{A}$*, outputs a representation* $\mathbf{z} = (z_1, \ldots, z_m)^\top \in \mathbb{F}^m$*, where* $\mathbb{F}$ *is the finite field of* p *elements, for every group element* $[y]$ *in the output of* $\mathcal{A}$ *such that:*

$$Adv^{alg}_{\mathbb{G},\mathcal{A}}(\lambda) = \begin{bmatrix} [y] \leftarrow \mathcal{A}([x_1], \ldots, [x_m]), \mathbf{z} \leftarrow \mathcal{E}_\mathcal{A}([y], [x_1], \ldots, [x_m]), \\ \text{and } [y] \neq \sum_{j=1}^{m} z_j [x_j] \end{bmatrix} = \mathsf{negl}(\lambda).$$

The security of our final argument for R1CS-lite (after compilation) is proven in the algebraic group model under the following assumption:

Definition 16 (q-dlog Asymmetric Assumption). *The* $q(\lambda)$*-discrete logarithm assumption holds for* $gk \leftarrow \mathcal{G}(1^\lambda)$ *if for all PPT algorithm* $\mathcal{A}$

$$Adv^{q-dlog}_{gk,\mathcal{A}}(\lambda) = \Pr\left[x \leftarrow \mathcal{A}(gk, [x]_{1,2}, \ldots, [x^q]_{1,2})\right] = \mathsf{negl}(\lambda).$$

3 Generalized Univariate Sumcheck

In this section, we revisit the sumcheck of Aurora [5]. As presented there, this argument allows to prove that the sum of the evaluations of a polynomial in some multiplicative[4] set $\mathbb{H}$ of a finite field $\mathbb{F}$ sum to 0. We generalize the argument to arbitrary sets $\mathbb{H} \subset \mathbb{F}$, solving an open problem posed there. Additionally, we give a simpler proof of the same result by connecting the sumcheck to polynomial evaluation and other basic properties of polynomials.

Given some finite field $\mathbb{F}$, let $\mathbb{H}$ be an arbitrary set of cardinal m, with some predefined canonical order, and h_i refers to the ith element in this order. The ith Lagrange basis polynomial associated to $\mathbb{H}$ is denoted by $\lambda_i(X)$. The vector $\boldsymbol{\lambda}(X)$ is defined as $\boldsymbol{\lambda}(X)^\top = (\lambda_1(X), \ldots, \lambda_m(X))$. The vanishing polynomial of $\mathbb{H}$ will be denoted by $t(X)$. When $\mathbb{H}$ is a multiplicative subgroup, the following properties are known to hold:

$$t(X) = X^m - 1, \qquad \lambda_i(X) = \frac{\mathsf{h}_i}{m} \frac{(X^m - 1)}{(X - \mathsf{h}_i)}, \qquad \lambda_i(0) = \frac{1}{m},$$

for any $i = 1, \ldots, m$. This representation makes their computation particularly efficient: both $t(X)$ and $\lambda_i(X)$ can be evaluated in $O(\log m)$ field operations.

We prove a generalized sumcheck theorem below, and derive the sumcheck of Aurora as a corollary for the special case where $\mathbb{H}$ is a multiplicative subgroup. The intuition is simple: let $P_1(X)$ be a polynomial of arbitrary degree in $\mathbb{F}[X]$, and $P_2(X) = \sum_{i=1}^m \lambda_i(X) P_1(\mathsf{h}_i)$. Note that $P_1(X), P_2(X)$ are congruent modulo $t(X)$, and the degree of $P_2(X)$ is at most $m-1$. Then, when $P_2(X)$ is evaluated at an arbitrary point $v \in \mathbb{F}$, $v \notin \mathbb{H}$, $P_2(v) = \sum_{i=1}^m \lambda_i(v) P_1(\mathsf{h}_i)$. Thus, $P_2(v)$ is "almost" (except for the constants $\lambda_i(v)$) the sum of the evaluations of $P_1(\mathsf{h}_i)$. Multiplying by a normalizing polynomial, we get rid of the constants and obtain a polynomial that evaluated at v is the sum of any set of evaluations of interest. The sum will be zero if this product polynomial has a root at v.

Theorem 1 (Generalized Sumcheck). *Let $\mathbb{H}$ be an arbitrary subset of some finite field $\mathbb{F}$ and $t(X)$ the vanishing polynomial at $\mathbb{H}$. For any $P(X) \in \mathbb{F}[X]$, $\mathcal{S} \subset \mathbb{H}$, and any $v \in \mathbb{F}, v \notin \mathbb{H}$, $\sum_{s \in \mathcal{S}} P(s) = \sigma$ if and only if there exist polynomials $H(X) \in \mathbb{F}[X]$, $R(X) \in \mathbb{F}_{\leq m-2}[X]$ such that*

$$P(X) N_{\mathcal{S},v}(X) - \sigma = (X - v) R(X) + t(X) H(X),$$

where $N_{\mathcal{S},v}(X) = \sum_{s \in \mathcal{S}} \lambda_s(v)^{-1} \lambda_s(X)$ and $\lambda_s(X)$ is the Lagrange polynomial associated to s and the set $\mathbb{H}$.

[4] In fact, the presentation is more general as they also consider additive cosets, but we stick to the multiplicative case which is the one that has been used in other constructions of zkSNARKs.

Proof. Observe that $P(X) = \sum_{\mathsf{h}\in\mathbb{H}} P(\mathsf{h})\lambda_{\mathsf{h}}(X) \mod t(X)$. Therefore,

$$\begin{aligned} P(X)N_{\mathcal{S},v}(X) - \sigma &= \Big(\sum_{\mathsf{h}\in\mathbb{H}} P(\mathsf{h})\lambda_{\mathsf{h}}(X)\Big)\Big(\sum_{s\in\mathcal{S}} \lambda_s(v)^{-1}\lambda_s(X)\Big) - \sigma \\ &= \Big(\sum_{s\in\mathcal{S}} P(s)\lambda_s(v)^{-1}\lambda_s(X)\Big) - \sigma \mod t(X). \end{aligned}$$

Let $Q(X) = \Big(\sum_{s\in\mathcal{S}} P(s)\lambda_s(v)^{-1}\lambda_s(X)\Big) - \sigma$. Note that $Q(v) = \sum_{s\in\mathcal{S}} P(s) - \sigma$. Thus, $\sum_{s\in\mathcal{S}} P(s) = \sigma$ if and only if $Q(X)$ is divisible by $X - v$. The claim follows from this observation together with the fact that $Q(X)$ is the unique polynomial of degree $m-1$ that is congruent with $P(X)N_{\mathcal{S},v}(X) - \sigma$. □

Lemma 1. *If $\mathcal{S} = \mathbb{H}$ is a multiplicative subgroup of $\mathbb{F}$, $N_{\mathbb{H},0}(X) = m$.*

Proof. Recall that, as $\mathbb{H}$ is a multiplicative subgroup, $\lambda_i(0) = 1/m$ for all $i = 1,\ldots,m$. Therefore, $N_{\mathbb{H},0}(X) = \sum_{i=1}^m \lambda_i(0)^{-1}\lambda_i(X) = m\sum_{i=1}^m \lambda_i(X) = m$. □

As a corollary of Lemma 1 and the Generalized Sumcheck, we recover the univariate sumcheck: if $\mathbb{H}$ is a multiplicative subgroup, $\sum_{\mathsf{h}\in\mathbb{H}} P(\mathsf{h}) = \sigma$ if and only if there exist polynomials $R(X), H(X)$ with $\deg(R(X)) \leq m-2$ such that $P(X)m - \sigma = XR(X) + t(X)H(X)$.

3.1 Application to Linear Algebra Arguments

Several works [5,14,13] have observed that R1CS languages can be reduced to proving a Hadamard product relation and a linear relation, where the latter consists on showing that two vectors $\boldsymbol{x}, \boldsymbol{y}$ are such that $\boldsymbol{y} = \mathbf{M}\boldsymbol{x}$, or equivalently, that the inner product of $(\boldsymbol{y}, \boldsymbol{x})$ with all the rows of $(\mathbf{I}, -\mathbf{M})$ is zero. When matrices and vectors are encoded as polynomials for succinctness, for constructing a PHP it is necessary to express these linear algebra operations as polynomial identities.

For the Hadamard product relation, the basic observation is that, for any polynomials $A(X), B(X), C(X)$, the equation

$$A(X)B(X) - C(X) = H(X)t(X), \tag{1}$$

holds for some $H(X)$ if and only if $(A(\mathsf{h}_1),\ldots,A(\mathsf{h}_m)) \circ (B(\mathsf{h}_1),\ldots,B(\mathsf{h}_m)) - (C(\mathsf{h}_1),\ldots,C(\mathsf{h}_m)) = 0$. In particular, $A(X) = \boldsymbol{a}^\top\boldsymbol{\lambda}(X)$, $B(X) = \boldsymbol{b}^\top\boldsymbol{\lambda}(X)$ encode vectors $\boldsymbol{a}, \boldsymbol{b}$, then $C(X) \mod t(X)$ encodes $\boldsymbol{a}\circ\boldsymbol{b}$. This Hadamard product argument is one of the main ideas behind the zkSNARK of Gentry et al. [22] and follow-up work.

For linear relations, the following Theorem explicitly derives a polynomial identity that encodes the inner product relation from the univariate sumcheck. This connection in a different formulation is implicit in previous works [5,14,13].

Theorem 2 (Inner Product Polynomial Relation). *For some $k \in \mathbb{N}$, let $\boldsymbol{y} = (\boldsymbol{y}_1, \dots, \boldsymbol{y}_k)$, $\boldsymbol{y}_i = (y_{ij})$, $\boldsymbol{d} = (\boldsymbol{d}_1, \dots, \boldsymbol{d}_k)$ be two vectors in $\mathbb{F}^{km}$, $\boldsymbol{y}_i, \boldsymbol{d}_i \in \mathbb{F}^m$, and $\mathbb{H}$ a multiplicative subgroup of $\mathbb{F}$ of order m. Then, $\boldsymbol{y} \cdot \boldsymbol{d} = 0$ if and only if there exist $H(X), R(X) \in \mathbb{F}[X]$, $R(X)$ of degree at most $m-2$ such that the following relation holds:*

$$\boldsymbol{Y}(X) \cdot \boldsymbol{D}(X) = XR(X) + t(X)H(X), \tag{2}$$

where $\boldsymbol{Y}(X) = (Y_1(X), \dots, Y_k(X))$ is a vector of polynomials of arbitrary degree such that $Y_i(\mathsf{h}_j) = y_{ij}$ for all $i = 1, \dots, k$, $j = 1, \dots, m$, and $\boldsymbol{D}(X) = (D_1(X), \dots, D_k(X))$ is such that $D_i(X) = \boldsymbol{d}_i^\top \boldsymbol{\lambda}(X)$.

Proof. Since $Y_i(\mathsf{h}_j) = y_{ij}$, for all i, j, $Y_i(X) = \boldsymbol{y}_i^\top \boldsymbol{\lambda}(X) \mod t(X)$. Therefore, $Y_i(X)D_i(X) = (\boldsymbol{y}_i^\top \boldsymbol{\lambda}(X))(\boldsymbol{d}_i^\top \boldsymbol{\lambda}(X)) \mod t(X)$, and by the aforementioned properties of the Lagrange basis, this is also congruent modulo $t(X)$ to $(\boldsymbol{y}_i \circ \boldsymbol{d}_i)^\top \boldsymbol{\lambda}(X)$. Therefore,

$$\begin{aligned}\boldsymbol{Y}(X) \cdot \boldsymbol{D}(X) = \sum_{i=1}^{k} Y_i(X)D_i(X) &= \sum_{i=1}^{k} (\boldsymbol{y}_i \circ \boldsymbol{d}_i)^\top \boldsymbol{\lambda}(X) \\ &= \left(\sum_{i=1}^{k} (\boldsymbol{y}_i \circ \boldsymbol{d}_i)^\top \right) \boldsymbol{\lambda}(X) \mod t(X).\end{aligned}$$

By Theorem 1, $\big(\big(\sum_{i=1}^{k} (\boldsymbol{y}_i \circ \boldsymbol{d}_i)^\top \big) \boldsymbol{\lambda}(X) \big) N_{\mathbb{H},0}(X)$ is divisible by X if and only if the sum of the coordinates of $\sum_{i=1}^{k} (\boldsymbol{y}_i \circ \boldsymbol{d}_i)$ is 0. The implication is also true after dividing by $N_{\mathbb{H},0}(X) = m$. The jth coordinate of $\sum_{i=1}^{k} (\boldsymbol{y}_i \circ \boldsymbol{d}_i)$ is $\sum_{i=1}^{k} y_{ij} d_{ij}$, thus the sum of all coordinates is $\sum_{j=1}^{m} \sum_{i=1}^{k} y_{ij} d_{ij} = \boldsymbol{y} \cdot \boldsymbol{d}$, which concludes the proof. □

In the rest of the paper $\mathbb{H}$ will always be a multiplicative subgroup, both for simplicity (as $N_{\mathbb{H},0} = m$), and efficiency (due to the properties that Lagrange and vanishing polynomials associated to multiplicative subgroups have). However, Theorem 2 can be easily generalized to arbitrary sets $\mathbb{H}$ (just multiplying the left side of Eq. (2) by $N_{\mathbb{H},0}$).

4 Checkable Subspace Sampling: Definition and Implications

In a *Checkable Subspace Sampling* (CSS) argument prover and verifier interactively agree on a polynomial $D(X)$ representing a vector $\boldsymbol{d}$ in the row space of a matrix $\mathbf{M}$. The fiber of the protocol is that $D(X)$ is calculated as a linear combination of encoding of the rows of $\mathbf{M}$ with some coefficients determined by the verifier, but the verifier does not need to calculate $D(X)$ itself (this would require the verifier to do linear work in the number of rows of $\mathbf{M}$). Instead, the prover can calculate this polynomial and then convince the verifier that it has been correctly computed.

Below we give the syntactical definition of Checkable Subspace Sampling. Essentially, a CSS scheme is similar to a PHP for a relation $\mathsf{R}_{\mathbf{M}}$, except that the statement $(\mathsf{cns}, D(X))$ is decided interactively, and the verifier has only oracle access to the polynomial $D(X)$. A CSS scheme can be used as a building block in a PHP, and the result is also a PHP.

Definition 17 (Checkable Subspace Sampling, CSS). *A checkable subspace sampling argument over a field $\mathbb{F}$ defines some $Q, m \in \mathbb{N}$, a set of admissible matrices $\mathcal{M}$, a vector of polynomials $\boldsymbol{\beta}(X) \in (\mathbb{F}[X])^m$, a coinspace $\mathcal{C}$, a sampling function $\mathsf{Smp} : \mathcal{C} \to \mathbb{F}^Q$, and a relation:*

$$\mathsf{R}_{\mathsf{CSS},\mathbb{F}} = \left\{ \begin{array}{rl} \big(\mathbf{M}, \mathsf{cns}, D(X)\big) : & \mathbf{M} \in \mathcal{M} \subset \mathbb{F}^{Q \times m}, D(X) \in \mathbb{F}[X], \mathsf{cns} \in \mathcal{C}, \\ & \boldsymbol{s} = \mathsf{Smp}(\mathsf{cns}), \text{ and } D(X) = \boldsymbol{s}^\top \mathbf{M} \boldsymbol{\beta}(X) \end{array} \right\}.$$

For any $\mathbf{M} \in \mathcal{M}$, it also defines:

$$\mathsf{R}_{\mathbf{M}} = \left\{ \big(\mathsf{cns}, D(X)\big) : \big(\mathbf{M}, \mathsf{cns}, D(X)\big) \in \mathsf{R}_{\mathsf{CSS},\mathbb{F}} \right\}.$$

It consists of three algorithms:

– *$\mathcal{I}_{\mathsf{CSS}}$ is the indexer: in an offline phase, on input $(\mathbb{F}, \mathbf{M})$ returns a set $\mathcal{W}_{\mathsf{CSS}}$ of $\mathsf{n}(0)$ polynomials $\{p_{0,j}(X)\}_{j=1}^{n(0)} \in \mathbb{F}[X]$. This algorithm is run once for each $\mathbf{M}$.*
– *Prover and Verifier proceed as in a PHP, namely, the verifier sends field elements to the prover and has oracle access to the polynomials outputted by both the indexer and the prover; this phase is run in two different stages:*
  - $\mathsf{Sampling}$*: $\mathcal{P}_{\mathsf{CSS}}$ and $\mathcal{V}_{\mathsf{CSS}}$ engage in an interactive protocol. In some round, the verifier sends $\mathsf{cns} \leftarrow \mathcal{C}$, and the prover replies with $D(X) = \boldsymbol{s}^\top \mathbf{M} \boldsymbol{\beta}(X)$, for $\boldsymbol{s} = \mathsf{Smp}(\mathsf{cns})$.*
  - $\mathsf{ProveSampling}$*: $\mathcal{P}_{\mathsf{CSS}}$ and $\mathcal{V}_{\mathsf{CSS}}$ engage in another interactive protocol to prove that $(\mathsf{cns}, D(X)) \in \mathsf{R}_{\mathbf{M}}$.*
– *When the proving phase is concluded, the verifier outputs a bit indicating acceptance or rejection.*

The vector $\boldsymbol{\beta}(X) = (\beta_1(X), \dots, \beta_m(X))$ defines an encoding of vectors as polynomials: vector $\boldsymbol{v}$ is mapped to the polynomial $\boldsymbol{v}^\top \boldsymbol{\beta}(X) = \sum_{i=1}^m v_i \beta_i(X)$. When using a CSS for constructing an argument of membership in linear spaces as in the next section, we choose a characterization of inner product that is compatible with Lagrange polynomials. Thus, in this work, $\beta_i(X)$ is defined as $\lambda_i(X)$, the ith Lagrange polynomial associated to some multiplicative subgroup $\mathbb{H}$ of $\mathbb{F}$. Still, it also makes sense to consider also CSS arguments for other polynomial encodings, e.g. the monomial basis or Laurent polynomials.

We require a CSS argument to satisfy the following security definitions:

Perfect Completeness. If both prover and verifier are honest the output of the protocol is 1:

$$\Pr\left[\langle \mathcal{P}_{\mathsf{CSS}}(\mathbb{F},\mathbf{M},\mathsf{cns}), \mathcal{V}_{\mathsf{CSS}}^{\mathcal{W}_{\mathsf{CSS}}}(\mathbb{F})\rangle = 1\right] = 1.$$

where the probability is taken over the random coins of prover and verifier.

Soundness. A checkable subspace sampling argument $(\mathcal{I}_{\mathsf{CSS}}, \mathcal{P}_{\mathsf{CSS}}, \mathcal{V}_{\mathsf{CSS}})$ is ϵ-sound if for all $\mathbf{M}$ and any polynomial time prover $\mathcal{P}^*_{\mathsf{CSS}}$:

$$\Pr\left[D^*(X) \neq \boldsymbol{s}^\top \mathbf{M}\boldsymbol{\beta}(X) \;\middle|\; \begin{array}{c} (\mathsf{cns}, D^*(X)) \leftarrow \mathsf{Sampling}\langle \mathcal{P}^*_{\mathsf{CSS}}(\mathbb{F},\mathbf{M},\mathsf{cns}), \mathcal{V}^{\mathcal{W}_{\mathsf{CSS}}}(\mathbb{F})\rangle; \\ \boldsymbol{s} = \mathsf{Smp}(\mathsf{cns});\; \langle \mathcal{P}^*_{\mathsf{CSS}}(\mathbb{F},\mathbf{M},\mathsf{cns}), \mathcal{V}_{\mathsf{CSS}}^{\mathcal{W}_{\mathsf{CSS}}}(\mathbb{F})\rangle = 1 \end{array} \right] \leq \epsilon.$$

The soundness of the CSS argument will ensure that the vector is sampled as specified by the coins of the verifier so the prover cannot influence its distribution. For a CSS argument to be useful, we additionally need that distribution induced by the sampling function is sufficiently "good". This is a geometric property that can be captured in the Elusive Kernel property defined below.

Definition 18. *A CSS argument is ϵ-elusive kernel*[5] *if*

$$\max_{\boldsymbol{t}\in\mathbb{F}^Q, \boldsymbol{t}\neq \mathbf{0}} \Pr\left[\, \boldsymbol{s}\cdot\boldsymbol{t} = 0 \;\middle|\; \boldsymbol{s} = \mathsf{Smp}(\mathsf{cns}); \mathsf{cns} \leftarrow \mathcal{C}\,\right] \leq \epsilon.$$

In practice, for most schemes, $\boldsymbol{s}$ is a vector of monomials or Lagrange basis polynomials evaluated at some point $x = \mathsf{cns}$, and this property is an immediate application of Schwartz-Zippel lemma, so we will not explicitly prove it for most of our CSS arguments.

4.1 Linear Arguments from Checkable Subspace Sampling

In this section we build a PHP for the universal relation of membership in linear subspaces:

$$\mathcal{R}_{\mathsf{LA}} = \left\{(\mathbb{F}, \mathbf{W}, \boldsymbol{y}) : \mathbf{W} \in \mathbb{F}^{Q\times km}, \boldsymbol{y} \in \mathbb{F}^{km} \text{ s.t. } \mathbf{W}\boldsymbol{y} = \mathbf{0}\right\},$$

using a CSS scheme as building block. That is, given a vector $\boldsymbol{y}$, the argument allows to prove membership in the linear space $\mathbf{W}^\perp = \{y \in \mathbb{F}^{km} : \mathbf{W}\boldsymbol{y} = \mathbf{0}\}$. Although relation $\mathcal{R}_{\mathsf{LA}}$ is polynomial-time decidable, it is not trivial to construct a polynomial holographic proof for it, as the verifier has only an encoding of $\mathbf{W}$ and $\boldsymbol{y}$.

A standard way to prove that some vector $\boldsymbol{y}$ is in $\mathbf{W}^\perp$ is to let the verifier sample a *sufficiently random* vector $\boldsymbol{d}$ in the row space of matrix $\mathbf{W}$, and prove $\boldsymbol{y}\cdot\boldsymbol{d} = 0$. Naturally, the vector $\boldsymbol{y}$ must be declared before $\boldsymbol{d}$ is chosen. We follow this strategy to construct a PHP for $\mathcal{R}_{\mathsf{LA}}$, except that the vector $\boldsymbol{d}$ is sampled by the prover itself on input the coins of the verifier through a CSS argument.

As we have seen in Section 2.1, it is natural in our application to proving R1CS to consider matrices in blocks. Thus, in this section we prove membership in $\mathbf{W}^\perp$ where the matrix is written in k blocks of columns, that is, $\mathbf{W} = (\mathbf{W}_1, \ldots, \mathbf{W}_k)$. The vectors $\boldsymbol{y}, \boldsymbol{d} \in \mathbb{F}^{km}$ are also written in blocks as $\boldsymbol{y}^\top = (\boldsymbol{y}_1^\top, \ldots, \boldsymbol{y}_k^\top)$ and $\boldsymbol{d}^\top = (\boldsymbol{d}_1^\top, \ldots, \boldsymbol{d}_k^\top)$.

[5] The name is inspired by the property of t-elusiveness of [35].

Each block of $\mathbf{W}$, as well as the vectors $\boldsymbol{y}, \boldsymbol{d}$ can be naturally encoded, respectively, as a vector of polynomials or a single polynomial multiplying on the right by $\boldsymbol{\lambda}(X)$. However, we allow for additional flexibility in the encoding of $\boldsymbol{y}$: our argument is parameterized by a set of valid witnesses W_Y and a function $\mathcal{E}_Y : W_Y \to (\mathbb{F}[X])^k$ that determines how $\boldsymbol{y}$ is encoded as a polynomial. Thanks to this generalization we can use the argument as a black-box in our R1CS-lite construction. There, valid witnesses are of the form $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a} \circ \boldsymbol{b})$ and, for efficiency, its encoding will be $(A(X) = \boldsymbol{a}^\top \boldsymbol{\lambda}(X), B(X) = \boldsymbol{b}^\top \boldsymbol{\lambda}(X), A(X)B(X))$, which means that the last element does not need to be sent.

The argument goes as follows. The prover sends a vector of polynomials $\boldsymbol{Y}(X)$ encoding $\boldsymbol{y}$. The CSS argument is used to delegate to the prover the sampling of $\boldsymbol{d}_i^\top$, $i = 1, \ldots, k$ in the row space of $\mathbf{W}_i$. Then, the prover sends $\boldsymbol{D}(X)$ together with a proof that $\boldsymbol{y} \cdot \boldsymbol{d} = 0$. For this inner product argument to work, we resort to Theorem 2 that guarantees that, if $\mathcal{E}_Y$ is an encoding such that if $\mathcal{E}_Y(\boldsymbol{y}) = \boldsymbol{Y}(X)$, then $Y_i(\mathsf{h}_j) = y_{ij}$, the inner product relation holds if and only if the verification equation is satisfied for some $H_t(X), R_t(X)$.

Because of the soundness property of the CSS argument, the prover cannot influence the distribution of $\boldsymbol{d}$, which is sampled according to the verifier's coins. Therefore, if $\boldsymbol{Y}(X)$ passes the test of the verifier, $\boldsymbol{y}$ is orthogonal to $\boldsymbol{d}$. By the Elusive Kernel property of the CSS argument, $\boldsymbol{d}$ will be sufficiently random. As it is sampled after $\boldsymbol{y}$ is declared, this will imply that $\boldsymbol{y}$ is in $\mathbf{W}^\perp$.

Offline Phase: $\mathcal{I}_{\mathsf{LA}}(\mathbb{F}, \mathbf{W})$: For $i = 1, \ldots, k$, run the indexer $\mathcal{I}_{\mathsf{CSS}}$ on input $(\mathbb{F}, \mathbf{W}_i)$ to obtain the set $\mathcal{W}_{\mathsf{CSS}i}$ and output $\mathcal{W}_{\mathsf{LA}} = \bigcup_{i=1}^k \mathcal{W}_{\mathsf{CSS}i}$.

Online Phase: $\mathcal{P}_{\mathsf{LA}}$: On input a witness $\boldsymbol{y} \in W_Y \subset (\mathbb{F}^m)^k$, output $\boldsymbol{Y}(X) = \mathcal{E}_Y(\boldsymbol{y})$.
$\mathcal{P}_{\mathsf{LA}}$ and $\mathcal{V}_{\mathsf{LA}}$ run in parallel k instances of the CSS argument, with inputs $(\mathbb{F}, \mathbf{W}_i)$ and $\mathbb{F}$, respectively, and where the verifier is given oracle access to $\mathcal{W}_{\mathsf{CSS}i}$. The output is a set $\{(\mathsf{cns}, D_i(X))\}_{i=1}^k$, where cns are the same for all k instances. Define $\boldsymbol{D}(X) = (D_1(X), \ldots, D_k(X))$.
$\mathcal{P}_{\mathsf{LA}}$: Outputs $R_t(X) \in \mathbb{F}_{\leq m-2}[X], H_t(X)$ such that

$$\boldsymbol{Y}(X) \cdot \boldsymbol{D}(X) = X R_t(X) + t(X) H_t(X). \tag{3}$$

Decision Phase: Accept if and only if (1) $\deg(R_t) \leq m - 2$, (2) $\mathcal{V}^i_{\mathsf{CSS}}$ accepts $(\mathsf{cns}, D_i(X))$, and (3) the following equation holds:

$$\boldsymbol{Y}(X) \cdot \boldsymbol{D}(X) = X R_t(X) + t(X) H_t(X).$$

Fig. 1. Argument for proving membership in $\mathbf{W}^\perp$, parameterized by the polynomial encoding $\mathcal{E}_Y : W_Y \to \mathbb{F}[X]^k$, and the set $W_Y \subset \mathbb{F}^{km}$.

Theorem 3. *When instantiated using a CSS scheme with perfect completeness, and when the encoding $\mathcal{E}_Y : W_Y \to \mathbb{F}[X]^k$ satisfies that, if $\mathcal{E}_Y(\boldsymbol{y}) = \boldsymbol{Y}(X)$, then $Y_i(\mathsf{h}_j) = y_{ij}$, the PHP of Fig. 1 has perfect completeness.*

Proof. By definition, $\boldsymbol{D}(X) = (\boldsymbol{s}^\top \mathbf{W}_1 \boldsymbol{\lambda}(X), \ldots, \boldsymbol{s}^\top \mathbf{W}_k \boldsymbol{\lambda}(X))$, for $\boldsymbol{s} = \mathsf{Samp}(\mathsf{cns})$. Note that this is because the k instances of the CSS scheme are run in parallel and the same coins are used to sample each of the $\boldsymbol{d}_i$. Thus, $\boldsymbol{D}(X)$ is the polynomial encoding of $\boldsymbol{d} = (\boldsymbol{s}^\top \mathbf{W}_1, \ldots, \boldsymbol{s}^\top \mathbf{W}_k) = \boldsymbol{s}^\top \mathbf{W}$. Therefore, if $\boldsymbol{y}$ is in $\mathbf{W}^\perp$, $\boldsymbol{d} \cdot \boldsymbol{y} = \boldsymbol{s}^\top \mathbf{W} \boldsymbol{y} = \mathbf{0}$. By the characterization of inner product, as explained in Section 3, this implies that polynomials $H_t(X), R_t(X)$ satisfying the verification equation exist. □

Theorem 4. *Let* CSS *be ϵ-sound and ϵ'-Elusive Kernel, and $\mathcal{E}_Y : W_Y \to \mathbb{F}[X]^k$ an encoding such that if $\mathcal{E}_Y(\boldsymbol{y}) = \boldsymbol{Y}(X)$, $Y_i(\mathsf{h}_j) = y_{ij}$. Then, for any polynomial time adversary $\mathcal{A}$ against the soundness of PHP of Fig. 1:*

$$\mathsf{Adv}(\mathcal{A}) \leq \epsilon' + k\epsilon.$$

Further, the PHP satisfies 0-knowledge soundness.

Proof. Let $\boldsymbol{Y}^*(X) = (Y_1^*(X), \ldots, Y_k^*(X))$ be the output of a cheating $\mathcal{P}^*_{\mathsf{LA}}$ and $\boldsymbol{y}^* = (\boldsymbol{y}_1^*, \ldots, \boldsymbol{y}_k^*)$ the vector such that $Y_i^*(\mathsf{h}_j) = y_{ij}^*$. As a direct consequence of Theorem 2, $\boldsymbol{Y}^*(X) \cdot \boldsymbol{D}(X) = X R_t(X) + t(X) H_t(X)$ only if $\boldsymbol{y}^* \cdot \boldsymbol{d} = 0$, where $\boldsymbol{d}$ is the unique vector $\boldsymbol{d}$ such that $\boldsymbol{D}(X) = (\boldsymbol{d}_1^\top \boldsymbol{\lambda}(X), \ldots, \boldsymbol{d}_k^\top \boldsymbol{\lambda}(X))$.

On the other hand, the soundness of the CSS scheme guarantees that, for each i, the result of sampling $D_i(X)$ corresponds to the sample coins sent by the verifier, except with probability ϵ. Thus, the chances that the prover can influence the distribution of $\boldsymbol{D}(X)$ so that so that $\boldsymbol{y}^* \cdot \boldsymbol{d} = 0$ are at most $k\epsilon$. Excluding this possibility, a cheating prover can try to craft $\boldsymbol{y}^*$ in the best possible way to maximize the chance that $\boldsymbol{y}^* \cdot \boldsymbol{d} = 0$. Since $\boldsymbol{d}^\top = \boldsymbol{s}^\top \mathbf{W}$, and in a successful attack $\boldsymbol{y}^* \notin \mathbf{W}^\perp$, we can see that this possibility is bounded by the probability:

$$\max_{\boldsymbol{y}^* \notin \mathbf{W}^\perp} \Pr\left[\boldsymbol{d} \cdot \boldsymbol{y}^* = 0 \;\middle|\; \begin{array}{r} \mathsf{cns} \leftarrow \mathcal{C}; \\ \boldsymbol{s} = \mathsf{Smp}(\mathsf{cns}); \\ \boldsymbol{d} = \boldsymbol{s}^\top \mathbf{W} \end{array} \right] = \max_{\boldsymbol{y}^* \notin \mathbf{W}^\perp} \Pr\left[\boldsymbol{s}^\top \mathbf{W} \boldsymbol{y}^* = 0 \;\middle|\; \begin{array}{r} \mathsf{cns} \leftarrow \mathcal{C}; \\ \boldsymbol{s} = \mathsf{Smp}(\mathsf{cns}) \end{array} \right]$$

Since $\boldsymbol{s}^\top \mathbf{W} \boldsymbol{y}^* = \boldsymbol{s} \cdot (\mathbf{W} \boldsymbol{y}^*)$, and $\mathbf{W} \boldsymbol{y}^* \neq \mathbf{0}$, this can be bounded by ϵ', by the elusive kernel property of the CSS scheme.

For knowledge soundness, define the extractor $\mathcal{E}$ as the algorithm that runs the prover and, by evaluating $Y_i(X)$ in $\{\mathsf{h}_j\}_{j=1}^m$ for all $i \in [k]$, recovers $\boldsymbol{y}$. If the verifier accepts with probability greater than $\epsilon' + k\epsilon$, then $\boldsymbol{y}$ is such that $\mathbf{W}\boldsymbol{y} = \mathbf{0}$ with the same probability. □

Extension to other polynomial encodings. As mentioned, the construction is specific to the polynomial encoding defined by interpolation. However, the only place where this plays a role is in the check of equation (3). Now, if the polynomial encoding $\boldsymbol{\beta}(X)^\top$ associated to the CSS argument for $\mathbf{W}$ was set to be for instance the monomial basis, i.e. $\boldsymbol{\beta}(X)^\top = (1, X, \ldots, X^{m-1})$, the argument can be easily modified to still work. It suffices to choose the "reverse" polynomial encoding for $\boldsymbol{y}$, that is define $\boldsymbol{Y}(X) = (\boldsymbol{y}_1^\top \tilde{\boldsymbol{\beta}}(X), \ldots, \boldsymbol{y}_k^\top \tilde{\boldsymbol{\beta}}(X))$, where $\tilde{\boldsymbol{\beta}}(X)^\top = (X^{m-1}, \ldots, X, 1)$, and require the prover to find $R_t(X), H_t(X)$, with $R_t(X)$ of degree at most $m-2$ such that:

$$\boldsymbol{Y}(X) \cdot \boldsymbol{D}(X) = R_t(X) + X^m H_t(X). \tag{4}$$

Indeed, observe that this check guarantees that $\boldsymbol{Y}(X) \cdot \boldsymbol{D}(X)$ does not have any term of degree exactly $m-1$, and the term of degree $m-1$ is exactly $\sum_{i=1}^k \boldsymbol{y}_i \cdot \boldsymbol{d}_i = \boldsymbol{y} \cdot \boldsymbol{d}$.

4.2 R1CS-lite from Linear Arguments

In this section we give a PHP for R1CS-lite by combining our linear argument with other well known techniques. In this section, $\mathbf{W}$ is the block matrix defined in Section 2.1.

Offline Phase: $\mathcal{I}_{\mathsf{lite}}(\mathbf{W}, \mathbb{F})$ runs $\mathcal{I}_{\mathsf{LA}}(\mathbf{W}, \mathbb{F})$ to obtain a list of polynomials $\mathcal{W}_{\mathsf{LA}}$ and outputs $\mathcal{W}_{\mathsf{lite}} = \mathcal{W}_{\mathsf{LA}}$.
Online Phase: $\mathcal{P}_{\mathsf{lite}}(\mathbb{F}, \mathbf{W}, \boldsymbol{x}, (\boldsymbol{a}', \boldsymbol{b}'))$ defines $\boldsymbol{a} = (1, \boldsymbol{x}, \boldsymbol{a}'), \boldsymbol{b} = (\mathbf{1}_l, \boldsymbol{b}')$, and computes

$$A'(X) = \left(\sum_{j=l+1}^{m} a_j \lambda_j(X) \right) / t_l(X),\ B'(X) = \left(\left(\sum_{j=1}^{m} b_j \lambda_j(X) \right) - 1 \right) / t_l(X),$$

for $t_l(X) = \prod_{i=1}^{\ell} (X - \mathsf{h}_i)$. It outputs $\big(A'(X), B'(X)\big)$.
$\mathcal{V}_{\mathsf{lite}}$ and $\mathcal{P}_{\mathsf{lite}}$ instantiate $\mathcal{V}_{\mathsf{LA}}^{\mathcal{W}_{\mathsf{LA}}}(\mathbb{F})$ and $\mathcal{P}_{\mathsf{LA}}(\mathbb{F}, \mathbf{W}, (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a} \circ \boldsymbol{b}))$. Let $\boldsymbol{Y}(X) = (A(X), B(X), A(X)B(X))$ be the polynomials outputted by $\mathcal{P}_{\mathsf{LA}}$ in the first round.

Decision Phase: Define $C_l(X) = \lambda_1(X) + \sum_{j=1}^{l-1} x_j \lambda_{j+1}(X)$ and accept if and only if (1) $A(X) = A'(X)t_l(X) + C_l(X)$, (2) $B(X) = B'(X)t_l(X) + 1$, and (3) $\mathcal{V}_{\mathsf{LA}}$ accepts.

Fig. 2. PHP for $\mathcal{R}'_{\text{R1CS-lite}}$ from PHP for $\mathcal{R}_{\mathsf{LA}}$. The PHP for $\mathcal{R}_{\mathsf{LA}}$ should be instantiated for $W_Y = \{(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a} \circ \boldsymbol{b}) : \boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^m\}$, $\mathcal{E}(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a} \circ \boldsymbol{b}) = (\boldsymbol{a}^\top \boldsymbol{\lambda}(X), \boldsymbol{b}^\top \boldsymbol{\lambda}(X), (\boldsymbol{a}^\top \boldsymbol{\lambda}(X))(\boldsymbol{b}^\top \boldsymbol{\lambda}(X)))$.

Theorem 5. *When instantiated with a complete, sound and knowledge sound linear argument, the PHP of Fig. 2 satisfies completeness, soundness and knowledge-soundness.*

Proof. Completeness follows directly from the definition of $A'(X), B'(X), A(X), B(X)$ and completeness of the linear argument. Soundness and knowledge soundness hold if the linear argument is sound as well, because $\mathcal{V}_{\mathsf{lite}}$ accepts if $\mathcal{V}_{\mathsf{LA}}$ accepts, meaning $\mathbf{W}(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a} \circ \boldsymbol{b})^\top = 0$ and $\mathcal{R}'_{\text{R1CS-lite}}$ holds, and for extraction it suffices to use the extractor of the linear argument. ◻

4.3 Adding Zero Knowledge

To achieve zero-knowledge, it is common to several works on pairing-based zkSNARKS [13,14,22] to randomize the polynomial commitment to the witness with a polynomial that is a multiple of the vanishing polynomial. That is, the commitment to a vector $\boldsymbol{a}$ is $A(X) = \sum a_i \lambda_i(X) + t(X)h(X)$, where $t(X), \lambda_i(X)$ are defined as usual, and the coefficients of $h(X)$ are the randomness. In [22], $h(X)$ can be constant, since the commitment $A(X)$ in the final argument is evaluated at a single point. In other works where the commitment needs to support queries at several point values, $h(X)$ needs to be of higher degree. In Marlin, it is suggested to choose the degree according to the number of oracle queries to maximimize efficiency, and in Lunar this idea is developed into a fine-grained analysis and a vector with query bounds is specified for the compiler. Additionally, for this technique, the prover needs to send a masking polynomial to randomize the polynomial $R(X)$ of the inner product check. The reason is that this polynomial leaks information about $(A(X), B(X), A(X)B(X)) \cdot \boldsymbol{D}(X) \mod t(X)$.

In this section, we show how to add zero-knowledge to the PHP for R1CS-lite of Section 4.2 without sending additional polynomials. The approach is natural and a similar technique has also been used in [38]. Let $(\mathsf{b}_A, \mathsf{b}_B, \mathsf{b}_{R_t}, \mathsf{b}_{H_t})$ be the tuple of bounds on the number of polynomial evaluations seen by the verifier after compiling for the polynomials $A(X), B(X), R_t(X), H_t(X)$. To commit to a vector $\boldsymbol{y} \in \mathbb{F}^m$, we sample some randomness $\boldsymbol{r} \in \mathbb{F}^n$, where n is a function of $(\mathsf{b}_A, \mathsf{b}_B, \mathsf{b}_{R_t}, \mathsf{b}_{H_t})$ to be specified (a small constant when compiling). The cardinal of $\mathbb{H}$ is denoted by $\tilde{m}$ in this section. A commitment is defined in the usual way for the vector $(\boldsymbol{y}, \boldsymbol{r})$, i.e. $\sum_{i=1}^{m} y_i \lambda_i(X) + \sum_{i=m+1}^{m+n} r_i \lambda_i(X)$, and, naturally, we require $m + n \leq \tilde{m}$. Our idea is to consider related randomness for $A(X), B(X)$ so that the additional randomness sums to 0 and does not interfere with the inner product argument. The novel approach is to enforce this relation of the randomness by adding one additional constraint to $\mathbf{W}$. The marginal cost of this for the prover is minimal. Starting from the PHP of Fig. 2 we introduce the changes described in Fig. 3.

Offline Phase: For $\tilde{m} = m + n$, the matrix of constraints is:

$$\tilde{\mathbf{W}} = \begin{pmatrix} \mathbf{I}_m & \mathbf{0}_{m\times n} & \mathbf{0}_{m\times m} & \mathbf{0}_{m\times n} & -\mathbf{F} & \mathbf{0}_{m\times n} \\ \mathbf{0}_{m\times m} & \mathbf{0}_{m\times n} & \mathbf{I}_m & \mathbf{0}_{m\times n} & -\mathbf{G} & \mathbf{0}_{m\times n} \\ \mathbf{0}_m^\top & \mathbf{1}_n^\top & \mathbf{0}_m^\top & \mathbf{1}_n^\top & \mathbf{0}_m^\top & \mathbf{0}_n^\top \end{pmatrix}$$

Online Phase: $\mathcal{P}_{\mathsf{lite}}$ samples $\boldsymbol{r}_a \leftarrow \mathbb{F}^n, \boldsymbol{r}_b \leftarrow \mathbb{F}^n$ conditioned on $\sum_{i=1}^{n} r_{a,i} + r_{b,i} = 0$ and uses $\tilde{\boldsymbol{a}} := (1, \boldsymbol{x}, \boldsymbol{a}', \boldsymbol{r}_a)$, $\tilde{\boldsymbol{b}} := (\mathbf{1}_l, \boldsymbol{b}', \boldsymbol{r}_b)$, to construct $\tilde{A}(X)$ and $\tilde{B}(X)$, $\tilde{A}'(X)$ and $\tilde{B}'(X)$ as before.

Fig. 3. Modification of the PHP for $\mathcal{R}'_{\text{R1CS-}\mathsf{lite}}$ to achieve zero-knowledge. The omitted parts are identical.

Theorem 6. *With the modification described in Fig. 3 the PHP of Fig. 2 is perfectly complete, sound, knowledge-sound, perfect zero-knowledge and* $(\mathsf{b}_A, \mathsf{b}_B, \mathsf{b}_{R_t}, \mathsf{b}_{H_t})$*-bounded honest-verifier zero-knowledge if* $n \geq \big(\mathsf{b}_A + \mathsf{b}_B + \mathsf{b}_{R_t} + \mathsf{b}_{H_t} + 1\big)/2$*, and* $n \geq \max(\mathsf{b}_A, \mathsf{b}_B)$.

Proof. The only difference with the previous argument is the fact that the matrix of constraints has changed, which is now $\tilde{\mathbf{W}}$. For completeness, observe that the additional constraint makes sure that $\sum_{i=1}^{n} r_{a,i} + r_{b,i} = 0$, and an honest prover chooses the randomness such that this holds. On the other hand, the sumcheck theorem together with this equation guarantee that the randomness does not affect the divisibility at 0 of $(\tilde{A}(X), \tilde{B}(X), \tilde{A}(X)\tilde{B}(X)) \cdot \boldsymbol{D}(X) \mod t(X)$.

For soundness, note that $\tilde{\mathbf{W}}\left(\tilde{\boldsymbol{a}}^\top, \tilde{\boldsymbol{b}}^\top, (\tilde{\boldsymbol{a}} \circ \tilde{\boldsymbol{b}})^\top\right)$, is equivalent to 1) $\boldsymbol{a} = \mathbf{F}(\boldsymbol{a} \circ \boldsymbol{b})$, 2) $\boldsymbol{b} = \mathbf{G}(\boldsymbol{a} \circ \boldsymbol{b})$, and 3) $\sum_{i=1}^n r_{a,i} + r_{b,i} = 0$, for $\boldsymbol{a} := (1, \boldsymbol{x}, \boldsymbol{a}')$ $\boldsymbol{b} := (\mathbf{1}_l, \boldsymbol{b}')$. This is because the first two blocks of constraints have 0s in the columns corresponding to $\boldsymbol{r}_a, \boldsymbol{r}_b$, and the other way around for the last constraint. Therefore, by the soundness of the linear argument $\sum_{i=1}^n r_{a,i} + r_{b,i} = 0$, and the randomness does not affect divisibility at 0 of $(A(X), B(X), A(X)B(X))^\top \cdot \boldsymbol{D}(X) \mod t(X)$, so the same reasoning used for the argument of Fig. 2 applies.

Perfect zero-knowledge of the PHP is immediate, as all the messages in the CSS procedure contain only public information and the rest of the information exchanged are oracle polynomials.

We now prove honest-verifier bounded zero-knowledge. The simulator is similar to [13](Th. 4.7), but generalized to the distribution of $\boldsymbol{D}(X)$ induced by the underlying CSS scheme. The simulator gets access to the random tape of the honest verifier and receives x and the coins of the CSS scheme, as well as a list of its checks. It creates honestly all the polynomials of the CSS argument, since these are independent of the witness.

For an oracle query at point γ, the simulator samples uniform random values $A'_\gamma, B'_\gamma, R_{\gamma,t}$ in $\mathbb{F}$ and declares them, respectively, as $A'(\gamma), B'(\gamma), R_t(\gamma)$. It then defines the rest of the values to be consistent with them. More precisely, let $\boldsymbol{D}(X)^\top = \boldsymbol{s}^\top \mathbf{W} \boldsymbol{\lambda}(X) = (D_a(X), D_b(X), D_{ab}(X))$ be the output of the CSS argument, which the simulator can compute with the CSS coins. Then, the simulator sets:

$$A_\gamma = A'_\gamma t_l(\gamma) + \sum_{i=1}^{l} x_i \lambda_i(\gamma), \qquad B_\gamma = B'_\gamma t_l(\gamma) + 1,$$

$$p_\gamma = D_a(\gamma) A_\gamma + D_b(\gamma) B_\gamma + D_{ab}(\gamma) A_\gamma B_\gamma \qquad H_{t\gamma} = (p_\gamma - \gamma R_{t,\gamma})/t(\gamma),$$

where Q_γ for $Q \in \{A', B', R_t, H_t\}$ is declared as $Q(\gamma)$. The simulator keeps a table of the computed values to answer consistently the oracle queries.

We now argue that the queries have the same distribution as the evaluations of the prover's polynomials if all the queries γ are in $\mathbb{F} \backslash \mathbb{H}$. Since the verifier is honest, and $|\mathbb{H}|$ is assumed to be a negligible fraction of the field elements, we can always assume this is the case. In this case, the polynomial encoding of $\boldsymbol{r}_a, \boldsymbol{r}_b$ acts as a masking polynomial for $A'(X), B'(X), R_t(X), H_t(X)$ and taking into account that $\sum_{i=1}^n r_{a,i} + r_{b,i} = 0$ to have the same distribution it is sufficient that $2n - 1 \geq \mathsf{b}_A + \mathsf{b}_B + \mathsf{b}_{R_t} + \mathsf{b}_{H_t}$, and $n \geq \max(\mathsf{b}_A + \mathsf{b}_B)$, as stated in the theorem. Therefore, bounded zero-knowledge is proven. $\square$

4.4 Combining CSS schemes

Since a CSS scheme outputs a linear combination of the rows of a matrix $\mathbf{M}$, different instances of a CSS scheme can be easily combined with linear operations. More precisely, given a matrix $\mathbf{M}$ that can be written as $\begin{pmatrix} \mathbf{M}_1 \\ \mathbf{M}_2 \end{pmatrix}$, we can use a different CSS arguments for each $\mathbf{M}_i$[6] Since all current constructions of CSS arguments have limitations in terms of the types of matrices they apply to, this opens the door to decomposing the matrix of constraints into different blocks that admit efficient CSS

[6] The naive approach would run both CSS arguments in parallel, but savings might be possible batching the proofs.

arguments. For instance, matrices with a few very dense constraints (i.e. with very few rows with a lot of non-zero entries) and otherwise sparse could be split to use the scheme for sparse matrices of Section 3 for one part, and the trivial approach (where one polynomial for each row is computed by the indexer, and the verifier can sample the polynomial $D(X)$ computing the linear combination itself) for the rest. That is, one reason to divide the matrix $\mathbf{M}$ into blocks is to have a broader class of admissible matrices. Another reason is efficiency, since if a block that is either $\mathbf{0}$ or the identity matrix, the verifier can open the polynomial $D(X)$ itself, saving on the number of polynomials that need to be sent. More specifically, for our final construction, we will often split a matrix into two blocks of m rows, $\mathbf{M} = \begin{pmatrix}\mathbf{M}_1\\ \mathbf{M}_2\end{pmatrix}$, use the same CSS argument for each matrix with the same coins, and combine them to save on communication. More precisely, if $\boldsymbol{s} = \mathsf{Smp}(\mathsf{cns})$, and $D_1(X) = \boldsymbol{s}^\top \mathbf{M}_1 \boldsymbol{\lambda}(X)$ and $D_2(X) = \boldsymbol{s}^\top \mathbf{M}_2 \boldsymbol{\lambda}(X)$ are the polynomials associated to $\mathbf{M}_1, \mathbf{M}_2$, we will modify the CSS argument so that it sends $D_1(X) + zD_2(X)$ for some challenge z chosen by the verifier, instead of $D_1(X)$ and $D_2(X)$ individually. Note that $D_1(X) + zD_2(X) = (\boldsymbol{s}^\top, z\boldsymbol{s}^\top)\mathbf{M}\boldsymbol{\lambda}(X)$, that is, this corresponds to a CSS argument where the sampling coefficients depend on z also.

This cannot be done generically, it depends on the underlying CSS argument and the type of admissible matrices. Intuitively, this modification corresponds to implicitly constructing a CSS argument for the matrix $\mathbf{M}_1 + z\mathbf{M}_2$, so it is necessary that: a) the polynomials computed by the indexer of the CSS argument for $\mathbf{M}_1, \mathbf{M}_2$ can be combined, upon receiving the challenge z, to the CSS indexer polynomials of $\mathbf{M}_1 + z\mathbf{M}_2$, and b) that $\mathbf{M}_1 + z\mathbf{M}_2$ is an admissible matrix for this CSS argument. For instance, if $\mathbf{M}_1, \mathbf{M}_2$ has K non-zero entries each, and the admissible matrices of a CSS instance must have at most K non-zero entries, then $\mathbf{M}_1 + z\mathbf{M}_2$ is not generally an admissible matrix. We will be using this optimization for our final PHP for sparse matrices, and we will see there that these conditions are met in this case.

5 Constructions of Checkable Subspace Sampling Arguments

Given the results of the previous sections, for our R1CS-lite argument it is sufficient to design a CSS argument for matrices $\mathbf{M} \in \mathbb{F}^{m\times m}$ and then use it on all the blocks of $\mathbf{W}$. In this section, we give several novel CSS arguments for different types of square matrices.

We consider two disjoint sets of roots of unity, $\mathbb{H}, \mathbb{K}$ of degree m and K, respectively. For $\mathbb{H}$ we use the notation defined in Section 3. The elements of $\mathbb{K}$ are assumed to have some canonical order, and we use k_ℓ for the ℓth element in $\mathbb{K}$, $\mu_\ell(X)$ for the ℓth Lagrangian interpolation polynomial associated to $\mathbb{K}$, and $u(X)$ for the vanishing polynomial.

Matrices $\mathbf{M} \in \mathbb{F}^{m\times m}$ can be naturally encoded as a bivariate polynomial as $P(X,Y) = \boldsymbol{\alpha}(Y)^\top \mathbf{M}\boldsymbol{\beta}(X)$, for some $\boldsymbol{\alpha}(Y) \in \mathbb{F}[Y]^m, \boldsymbol{\beta}(X) \in \mathbb{F}[X]^m$. Let $\boldsymbol{m}_i^\top$ be the ith row of $\mathbf{M}$, and $P_i(X) = \boldsymbol{m}_i^\top \boldsymbol{\beta}(X)$. Then,

$$P(X,x) = \boldsymbol{\alpha}(x)^\top \mathbf{M}\boldsymbol{\beta}(X) = \sum_{i=1}^{m} \alpha_i(x) P_i(X).$$

That is, the polynomial $P(X,x)$ is a linear combination of the polynomials associated to the rows of $\mathbf{M}$ via the encoding defined by $\boldsymbol{\beta}(X)$, with coefficients $\alpha_i(x)$. This suggests to define a CSS scheme where, in the sampling phase, the verifier sends the challenge x and the prover replies with $D(X) = P(X, x)$, and, in the proving phase, the prover convinces the verifier that $D(X)$ is correctly sampled from coins x. This approach appears, implicitly or explicitly, in Sonic and most follow-up work we are aware of.

In Sonic, $\boldsymbol{\alpha}(Y), \boldsymbol{\beta}(X)$ are vectors of Laurent polynomials. In Marlin, Lunar and in this work, we set $\boldsymbol{\alpha}(Y) = \boldsymbol{\lambda}(Y)$, and $\boldsymbol{\beta}(X) = \boldsymbol{\lambda}(X)$. The choice of $\boldsymbol{\beta}(X)$ is to make the encoding compatible with the inner product defined by the sumcheck, and the choice of $\alpha(Y)$ is necessary for the techniques used in the proving phase of the CSS scheme that will be detailed in this Section.

For the proving phase, the common strategy is to follow the general template introduced in Sonic: the verifier samples a challenge $y \in \mathbb{F}$, checks that $D(y)$ is equal to a value σ sent by the prover, and that $\sigma = P(y, x)$ (through what is called a signature of correct computation, as in [36]). This proves that $D(X) = P(X, x)$. The last one is the challenging step, and is in fact, the main technical novelty of each of the mentioned previous works. In all of them, this is achieved by restricting the sets of matrices $\mathbf{M}$ to have a special structure: in Sonic they need to be sums of permutation matrices, and in Marlin, as later also Lunar, arbitrary matrices with at most K non-zero entries.

This section is organized as follows. We start by giving an overview of our new techniques in Section 5.1. In Section 5.2, we explain our basic CSS scheme, that works only for matrices with at most one non-zero element per column. In Section 5.3, we see how to compose these checks to achieve a CSS argument for arbitrary sparse matrices $\mathbf{M}$. In Section 5.4, we give an extension of the basic construction that can be used to generalize the CSS argument from basic matrices to sum of basic matrices without increasing the communication complexity. In the full version we explain how this can be used to extend the CSS argument for sparse matrices to matrices that are sums of sparse matrices without increasing the communication complexity.

5.1 Overview of New Techniques

Our main result of this section is a CSS scheme for any matrix $\mathbf{M} = (m_{i,j}) \in \mathbb{F}^{m \times m}$ of at most K non-zero entries. Assuming the non-zero entries are ordered, this matrix can be represented, as proposed in Marlin, by three functions $\mathsf{v} : \mathbb{K} \to \mathbb{F}$, $\mathsf{r} : \mathbb{K} \to [m]$, $\mathsf{c} : \mathbb{K} \to [m]$ such that $P(X, Y) = \sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell) \lambda_{\mathsf{r}(\mathsf{k}_\ell)}(Y) \lambda_{\mathsf{c}(\mathsf{k}_\ell)}(X)$, where the ℓth non-zero entry is $\mathsf{v}(\mathsf{k}_\ell) = m_{\mathsf{r}(\mathsf{k}_\ell), \mathsf{c}(\mathsf{k}_\ell)}$. If the matrix has less than K non-zero entries $\mathsf{v}(\mathsf{k}_\ell) = 0$, for $\ell = |\mathbf{M}|+1, \ldots, K$, and $\mathsf{r}(\mathsf{k}_\ell), \mathsf{c}(\mathsf{k}_\ell)$ are defined arbitrarily. We borrow this representation but design our own CSS scheme by following a "linearization strategy".

To see that $P(y, x)$ is correctly evaluated, we observe that it can be written as:

$$P(y, x) = \left(\lambda_{\mathsf{r}(\mathsf{k}_1)}(x), \ldots, \lambda_{\mathsf{r}(\mathsf{k}_K)}(x)\right) \cdot \left(\mathsf{v}(\mathsf{k}_1) \lambda_{\mathsf{c}(\mathsf{k}_1)}(y), \ldots, \mathsf{v}(\mathsf{k}_K) \lambda_{\mathsf{c}(\mathsf{k}_K)}(y)\right).$$

We define low degree extensions of each of these vectors respectively as:

$$e_x(X) = \sum_{\ell=1}^{K} \lambda_{\mathsf{r}(\mathsf{k}_\ell)}(x) \mu_\ell(X), \qquad e_y(X) = \sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell) \lambda_{\mathsf{c}(\mathsf{k}_\ell)}(y) \mu_\ell(X).$$

If the prover can convince the verifier that $e_x(X), e_y(X)$ are correctly computed, then it can show that $P(y, x) = \sigma$ by using the inner product argument to prove that the sum of $e_x(X) e_y(X) \mod t(X)$ at $\mathbb{K}$ is σ.

Observe that $e_x(X) = \boldsymbol{\lambda}(x)^\top \mathbf{M}_x \boldsymbol{\mu}(X)$, $e_y(X) = \boldsymbol{\lambda}(y)^\top \mathbf{M}_y \boldsymbol{\mu}(X)$, for some matrices $\mathbf{M}_x, \mathbf{M}_y$ with at most one non-zero element per column. To prove they are correctly computed it suffices to design a CSS argument for these simple matrices. This can be done in a much simpler way than in Marlin (and as in Lunar, that uses a similar technique), who prove directly that a low degree extension of $e_x(X)e_y(X)$ is correctly computed (intuitively, theirs is a quadratic check that requires the indexer to publish more information, as verifiers can only do linear operations in the polynomials output by it). Still, our technique is similar to theirs: given an arbitrary polynomial $e_x(X) = \sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{f}(\mathsf{k}_\ell)}(x)\mu_\ell(X)$, for some function $\mathsf{f} : \mathbb{K} \to [m]$, we can "complete" the Lagrange $\lambda_{\mathsf{f}(\mathsf{k}_\ell)}(x)$ with the missing term $(x - \mathsf{h}_{\mathsf{f}(\mathsf{k}_\ell)})$ to get the vanishing polynomial $t(x)$. The key insight is that the low degree extension of these "completing terms" is $x - v_1(X)$, where $v_1(X) = \sum_{\ell=1}^{K} \mathsf{h}_{\mathsf{f}(\mathsf{k}_\ell)}\mu_\ell(X)$ can be computed by the indexer.

The encoding for sparse matrices requires K to be at least $|\mathbf{M}|$, and generating a field with this large multiplicative subgroup can be a problem. In the full version, we consider a generalization to matrices $\mathbf{M}$ of a special form with sparsity KV, for any $V \in \mathbb{N}$. The interesting point is that communication complexity does not grow with V, and only the number of indexer polynomials grows (as $2V + 2$). This generalization is constructed from the argument for sums of basic matrices presented in Section 5.4.

We stress the importance of the linearization step: it not only allows for a simple explanation of underlying techniques for the proving phase, but also for generalizations such as the one in Section 5.4.

5.2 CSS Argument for Simple Matrices

Our basic building block is a CSS argument for matrices $\mathbf{M} = (m_{ij}) \in \mathbb{F}^{m \times K}$ with at most one non-zero value in each column, in particular, $|\mathbf{M}| \le K$. We define two functions associated to $\mathbf{M}$, $\mathsf{v} : \mathbb{K} \to \mathbb{F}$, $\mathsf{f} : \mathbb{K} \to [m]$. Given an element $\mathsf{k}_\ell \in \mathbb{K}$, $\mathsf{v}(\mathsf{k}_\ell) = m_{\mathsf{f}(\mathsf{k}_\ell),\ell} \neq 0$, i.e., function v outputs the only non zero value of column ℓ and f the corresponding row; if such a value does not exist set $\mathsf{v}(\mathsf{k}_\ell) = 0$ and $\mathsf{f}(\mathsf{k}_\ell)$ arbitrarily. We define the polynomial $P(X, Y)$ such that $D(X) = P(X, x)$ as $P(X, Y) = \boldsymbol{\lambda}(Y)^\top \mathbf{M}\boldsymbol{\mu}(X)$. Observe that, by definition of v and f, $P(X, Y) = \sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{f}(\mathsf{k}_\ell)}(Y)\mu_\ell(X)$.

Offline Phase: $\mathcal{I}_{\mathsf{CSS}}(\mathbb{F}, \mathbf{M})$ outputs $\mathcal{W}_{\mathsf{CSS}} = \{v_1(X), v_2(X)\}$, where

$$v_1(X) = \sum_{\ell=1}^{K} \mathsf{h}_{\mathsf{f}(\mathsf{k}_\ell)}\mu_\ell(X), \qquad v_2(X) = m^{-1}\sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell)\mathsf{h}_{\mathsf{f}(\mathsf{k}_\ell)}\mu_\ell(X).$$

Online Phase: Sampling: $\mathcal{V}_{\mathsf{CSS}}$ outputs $x \leftarrow \mathbb{F}$ and $\mathcal{P}_{\mathsf{CSS}}$ sends $D(X) = P(X, x)$. ProveSampling: $\mathcal{P}_{\mathsf{CSS}}$ finds and outputs $H_u(X)$ such that

$$D(X)\big(x - v_1(X)\big) = t(x)v_2(X) + H_u(X)u(X)$$

Decision Phase: Accept if and only if (1) $\deg D(X) \le K - 1$, and (2) $D(X)\big(x - v_1(X)\big) = t(x)v_2(X) + H_u(X)u(X)$.

Fig. 4. A simple CSS scheme for matrices with at most one non-zero element per column.

Theorem 7. *The argument of Figure 4 satisfies completeness and perfect soundness.*

Proof. When evaluated in any $\mathsf{k}_\ell \in \mathbb{K}$, the right side of the verification equation is $t(x)v_2(\mathsf{k}_\ell) = t(x)\mathsf{v}(\mathsf{k}_\ell)\mathsf{h}_{\mathsf{f}(\mathsf{k}_\ell)}m^{-1}$. Completeness follows from the fact that the left side is:

$$D(\mathsf{k}_\ell)(x - v_1(\mathsf{k}_\ell)) = \big(\mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{f}(\mathsf{k}_\ell)}(x)\big)\big(x - \mathsf{h}_{\mathsf{f}(\mathsf{k}_\ell)}\big) = t(x)\mathsf{v}(\mathsf{k}_\ell)m^{-1}\mathsf{h}_{\mathsf{f}(\mathsf{k}_\ell)}.$$

For soundness, note that the degree of $D(X)$ is at most $K-1$ and that the left side of the verification is $D(\mathsf{k}_\ell)(x - v_1(\mathsf{k}_\ell))$, so $D(\mathsf{k}_\ell) = t(x)\mathsf{v}(\mathsf{k}_\ell)m^{-1}\mathsf{h}_{\mathsf{f}(\mathsf{k}_\ell)}(x - \mathsf{h}_{\mathsf{f}(\mathsf{k}_\ell)})^{-1} = \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{f}(\mathsf{k}_\ell)}$, for all $\mathsf{k}_\ell \in \mathbb{K}$. Thus, $D(X) = \sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{f}(\mathsf{k}_\ell)}\mu_\ell(X)$. □

5.3 CSS argument for Sparse Matrices

In this section, we present a CSS argument for matrices $\mathbf{M}$ that are sparse without any restriction on the non-zero entries per column. We assume a set of roots of unity $\mathbb{K}$ such that $|\mathbf{M}| \leq K$ and define $P(X,Y) = \sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{r}(\mathsf{k}_\ell)}(Y)\lambda_{\mathsf{c}(\mathsf{k}_\ell)}(X)$. As explained in the overview, $P(y,x)$ can be written as the inner product of two vectors that depend only on x and y, and the low degree extensions of these vectors, $e_x(X), e_y(X)$, are nothing but the encodings of new matrices $\mathbf{M}_x$ and $\mathbf{M}_y$ in $\mathbb{F}^{m\times K}$ that have at most one non-zero element per column, so the basic CSS of Section 5.2 can be used to prove correctness.

Theorem 8. *The argument of Figure 5 satisfies completeness and* $(2K{+}1)/|\mathbb{F}|$*-soundness.*

Proof. Completeness follows immediately and thus we only prove soundness. Although it does so in a batched form, the prover is showing that the following equations are satisfied,

$$e_x(X)(x - v_{\mathsf{r}}(X)) = t(x)m^{-1}v_{\mathsf{r}}(X) + H_{u,x}(X)u(X)$$
$$e_y(X)(y - v_{1,\mathsf{c}}(X)) = t(y)v_{2,\mathsf{c}}(X) + H_{u,y}(X)u(X)$$
$$Ke_x(X)e_y(X) - \sigma = XR_u(X) + u(X)H_{u,x,y}(X),$$

Now, since all the left terms of the equations are defined before the verifier sends z, by the Schwartz-Zippel lemma, with all but probability $3/|\mathbb{F}|$, the verifier accepts if and only such $H_{u,x}(X), H_{u,y}(X), H_{u,x,y}(X), R_u(X)$ exist.

Assuming they do, the rest of the proof is a consequence of (1) soundness of the protocol in Fig. 4, which implies that $e_x(X), e_y(X)$ correspond to the correct polynomials modulo $u(X)$, and (2) Lemma 2 (see below) shows that if the last equation is satisfied, and $e_x(X), e_y(X)$ coincide with the honest polynomials modulo $u(X)$, then $\sigma = P(y,x)$. Because the prover sends $D(X)$ before receiving y and $D(y) = \sigma$, from the Schwartz-Zippel lemma we have that, except with negligible probability, $P(X,x) = D(X)$ and the argument is sound. □

Lemma 2. *Given* $e_x(X)$*,* $e_y(X)$ *such that* $e_x(X) = \sum_{\ell=1}^{K} \lambda_{\mathsf{r}(\mathsf{k}_\ell)}(x)\mu_\ell(X)$ *and* $e_y(X) = \sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{c}(\mathsf{k}_\ell)}(y)\mu_\ell(X)$*,* $P(y,x) = \sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{c}(\mathsf{k}_\ell)}(y)\lambda_{\mathsf{r}(\mathsf{k}_\ell)}(x) = \sigma$ *if and only if there exist polynomials* $R_u(X) \in \mathbb{F}_{\leq m-2}[X], H_{u,x,y}(X)$ *such that:*

$$e_x(X)e_y(X) - \sigma/K = XR_u(X) + H_{u,x,y}(X)u(X).$$

Proof. Note that $e_x(X)e_y(X) = \sum\limits_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{c}(\mathsf{k}_\ell)}(y)\lambda_{\mathsf{r}(\mathsf{k}_\ell)}(x)\mu_\ell(X) \mod u(X)$. By the univariate sumcheck (Lemma 1), $e_x(X)e_y(X) - \sigma/K$ is divisible by X if and only if $P(y,x) = \sigma$, which concludes the proof. □

Offline Phase: $\mathcal{I}_{\mathsf{CSS}}$ outputs $\mathcal{W}_{\mathsf{CSS}} = \big(v_{\mathsf{r}}(X), v_{1,\mathsf{c}}(X), v_{2,\mathsf{c}}(X)\big)$, where:

$$v_r(X) = \sum_{\ell=1}^{K} \mathsf{h}_{\mathsf{r}(\mathsf{k}_\ell)}\mu_\ell(X),$$

$$v_{1,\mathsf{c}}(X) = \sum_{\ell=1}^{K} \mathsf{h}_{\mathsf{c}(\mathsf{k}_\ell)}\mu_\ell(X), \qquad v_{2,\mathsf{c}}(X) = m^{-1}\sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell)\mathsf{h}_{\mathsf{c}(\mathsf{k}_\ell)}\mu_\ell(X).$$

Online Phase: $\mathsf{Sampling}$: $\mathcal{V}_{\mathsf{CSS}}$ sends $x \leftarrow \mathbb{F}$, and $\mathcal{P}$ outputs $D(X) = P(X, x)$, for $P(X,Y) = \sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{r}(\mathsf{k}_\ell)}(Y)\lambda_{\mathsf{c}(\mathsf{k}_\ell)}(X)$.

$\mathsf{ProveSampling}$: $\mathcal{V}_{\mathsf{CSS}}$ sends $y \leftarrow \mathbb{F}$ and $\mathcal{P}_{\mathsf{CSS}}$ outputs $\sigma = D(y)$ and $e_x(X), e_y(X)$, where $e_x(X) = \sum_{\ell=1}^{K} \lambda_{\mathsf{r}(\mathsf{k}_\ell)}(x)\mu_\ell(X)$, $e_y(X) = \sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{c}(\mathsf{k}_\ell)}(y)\mu_\ell(X)$, $\mathcal{V}_{\mathsf{CSS}}$ sends $z \leftarrow \mathbb{F}$ and $\mathcal{P}_{\mathsf{CSS}}$ computes $H_{u,x}(X)$, $H_{u,y}(X)$, $R_u(X)$, $H_{u,x,y}(X)$ such that:

$$e_x(X)(x - v_{\mathsf{r}}(X)) = m^{-1}t(x)v_{\mathsf{r}}(X) + H_{u,x}(X)u(X)$$

$$e_y(X)(y - v_{1,\mathsf{c}}(X)) = t(y)v_{2,\mathsf{c}}(X) + H_{u,y}(X)u(X)$$
$$Ke_x(X)e_y(X) - \sigma = XR_u(X) + u(X)H_{u,x,y}(X),$$

It also defines $H_u(X) = H_{u,x,y}(X) + zH_{u,x}(X) + z^2H_{u,y}(X)$, and outputs $\big(R_u(X), H_u(X)\big)$.

Decision Phase: Accept if and only if (1) $\deg(R_u) \le K - 2$, (2) $D(y) = \sigma$, and (3) for $i_x(X) = (x - v_{\mathsf{r}}(X))$, $i_y(X) = (y - v_{1,\mathsf{c}}(X))$

$$(e_x(X) + z^2 i_y(X))(e_y(X) + zi_x(X)) - z^3 i_x(X)i_y(X)$$
$$- z^2 t(y)v_{2,\mathsf{c}}(X) - \sigma/K - zt(x)m^{-1}v_{\mathsf{r}}(X) = XR_u(X) + H_u(X)u(X).$$

Fig. 5. CSS argument for $\mathbf{M}$, with $\mathbb{K}$ such that $|\mathbf{M}| \le |\mathbb{K}|$.

5.4 CSS Argument for Sums of Basic Matrices

In this section, we use $\mathbf{M}$ for a matrix in $\mathbb{F}^{m\times K}$ that can be written as $\sum_{i=1}^{V}\mathbf{M}_i$, with each $\mathbf{M}_i$ having at most one non-zero element in each column. We define two functions associated to each $\mathbf{M}_i$, $\mathsf{v}_i : \mathbb{K} \to \mathbb{F}$, $\mathsf{f}_i : \mathbb{K} \to [m]$ as in Section 5.2. This type of matrices will be used to design a generalization of the CSS argument for sums of sparse matrices in the full version.

Define $P(X,Y) = \boldsymbol{\lambda}(Y)^\top \mathbf{M}\boldsymbol{\mu}(X)$, and $D(X) = P(X, x)$. Observe that $P(X,Y) = \sum_{i=1}^{V}\sum_{\ell=1}^{K} \mathsf{v}_i(\mathsf{k}_\ell)\lambda_{\mathsf{f}_i(\mathsf{k}_\ell)}(Y)\mu_\ell(X)$. Let $S_\ell = \{\mathsf{f}_i(\mathsf{k}_\ell) : i \in [V]\}$, and $S_\ell^c = [K] - S_\ell$. The intuition is that, since there are at most V non zero $\mathsf{v}_i(\mathsf{k}_\ell)$ for each ℓ, we can factor as:

$$P(\mathsf{k}_\ell, x) = \sum_{i=1}^{V} \mathsf{v}_i(\mathsf{k}_\ell)\lambda_{\mathsf{f}_i(\mathsf{k}_\ell)}(x) = \prod_{s\in S_\ell^c}(x - \mathsf{h}_s)R_\ell(x),$$

where $R_\ell(X)$ is a polynomial of degree V. So, to "complete" $P(\mathsf{k}_\ell, x)$ to be a multiple of $t(x)$, we need to multiply it by $\prod_{s\in S_\ell}(x - \mathsf{h}_s)$, and the result will be $t(x)R_\ell(x)$. The trick is that $\hat{I}_\ell(Y) = \prod_{s\in S_\ell}(Y - \mathsf{h}_s)$, and $R_\ell(X)$ are polynomials of degrees V, $V - 1$, respectively. Thus, if the indexer publishes the coefficients of these polynomials in the monomial basis, they can be reconstructed by the verifier with coefficients $1, x, \dots, x^V$.

Offline Phase: $\mathcal{I}_{\mathsf{CSS}}(\mathbb{F}, \mathbf{M})$: Define the polynomials $\hat{R}_\ell(Y), \hat{I}_\ell(Y)$, and its coefficients $\hat{R}_{\ell j}, \hat{I}_{\ell j}$:

$$\hat{R}_\ell(Y) = \frac{1}{m}\sum_{i=1}^{V} \mathsf{v}_i(\mathsf{k}_\ell)\mathsf{h}_{f_i(\mathsf{k}_\ell)} \prod_{s \in S_\ell - \{f_i(\mathsf{k}_\ell)\}} (Y - \mathsf{h}_s) = \sum_{j=0}^{V-1} \hat{R}_{\ell j} Y^j,$$

$$\hat{I}_\ell(Y) = \prod_{s \in S_\ell} (Y - \mathsf{h}_s) = \sum_{j=0}^{V} \hat{I}_{\ell j} Y^j.$$

Define

$$v_j^{\hat{R}}(X) = \sum_{\ell=1}^{K} \hat{R}_{\ell j}\mu_\ell(X), \qquad v_j^{\hat{I}}(X) = \sum_{\ell=1}^{K} \hat{I}_{\ell j}\mu_\ell(X).$$

Output $\mathcal{W}_{\mathsf{CSS}} = \Big\{ \{v_j^{\hat{I}}(X)\}_{j=0}^{V}, \{v_j^{\hat{R}}(X)\}_{j=0}^{V-1} \Big\}$.

Online Phase: $\mathsf{Sampling}$: $\mathcal{V}_{\mathsf{CSS}}$ outputs $x \leftarrow \mathbb{F}$ and $\mathcal{P}_{\mathsf{CSS}}$ computes $D(X) = P(X, x)$.

$\mathsf{ProveSampling}$: $\mathcal{P}_{\mathsf{CSS}}$ finds and outputs $H_u(X)$ such that, if $\hat{R}_x(X) = \sum_{j=0}^{V-1} x^j v_j^{\hat{R}}(X)$, and $\hat{I}_x(X) = \sum_{j=0}^{V} x^j v_j^{\hat{I}}(X)$,

$$D(X)\hat{I}_x(X) = t(x)\hat{R}_x(X) + H_u(X)u(X).$$

Decision Phase: Accept if and only if (1) $\deg(D) \leq K-1$, and (2) $D(X)\hat{I}_x(X) = t(x)\hat{R}_x(X) + H_u(X)u(X)$.

Fig. 6. A CSS scheme for matrices with at most V non-zero elements per column.

Theorem 9. *The argument of Figure 6 satisfies completeness and perfect soundness.*

Proof. When evaluated in any $\mathsf{k}_\ell \in \mathbb{K}$, the right side of the verification equation is:

$$\begin{aligned} t(x)\hat{R}_x(x) &= \frac{t(x)}{m}\sum_{i=1}^{V} \mathsf{v}_i(\mathsf{k}_\ell)\mathsf{h}_{\mathsf{f}_i(\mathsf{k}_\ell)} \prod_{s \in S_\ell - \{\mathsf{f}_i(\mathsf{k}_\ell)\}} (x - \mathsf{h}_s) \\ &= \sum_{i=1}^{V} \mathsf{v}_i(\mathsf{k}_\ell)\frac{\mathsf{h}_{\mathsf{f}_i(\mathsf{k}_\ell)}}{m}\frac{t(x)}{x - \mathsf{h}_{\mathsf{f}_i(\mathsf{k}_\ell)}} \prod_{s \in S_\ell}(x - \mathsf{h}_s) = \prod_{s \in S_\ell}(x - \mathsf{h}_s)\sum_{i=1}^{V} \mathsf{v}_i(\mathsf{k}_\ell)\lambda_{\mathsf{f}_i(\mathsf{k}_\ell)}(x). \end{aligned}$$

The left side of the equation is $D(\mathsf{k}_\ell)\hat{I}_x(\mathsf{k}_\ell) = \big(\sum_{i=1}^{V} \mathsf{v}_i(\mathsf{k}_\ell)\lambda_{\mathsf{f}_i(\mathsf{k}_\ell)}(x)\big)\big(\prod_{s \in S_\ell}(x - \mathsf{h}_s)\big)$, so completeness is immediate. For soundness, if the verifier accepts $D(X)$, then $D(\mathsf{k}_\ell)\hat{I}_x(\mathsf{k}_\ell) = t(x)\hat{R}_x(\mathsf{k}_\ell)$ and $\hat{I}_x(\mathsf{k}_\ell) = \hat{I}_\ell(x)$, therefore:

$$D(\mathsf{k}_\ell) = \hat{I}_\ell(x)^{-1}t(x)\hat{R}_\ell(x) = \big(\prod_{s \in S_\ell^c}(x - \mathsf{h}_s)\big)\hat{R}_x(x) = \sum_{i=1}^{V} \mathsf{v}_i(\mathsf{k}_\ell)\lambda_{\mathsf{f}_i(\mathsf{k}_\ell)}(x).$$

We conclude that $D(X) = P(X, x) \mod u(X)$. Since both have degree at most $K - 1$, soundness is proven. □

6 A zkSNARK for R1CS-lite

The PHP for R1CS-lite can be compiled to a (zk)SNARK for this relation via standard techniques. Formally, since we have used the model of PHPs, this follows from Theorem 6.1 in [13]. Concretely, when using for compilation the polynomial commitment presented in Marlin (the variant secure in the AGM) and our PHP for R1CS-lite, the theorem states that it is sufficient to prove that the PHP is honest-verifier bounded zero-knowledge, where the bound for each oracle polynomial is the number of oracle queries plus one.

The universal SRS of the zkSNARK will be $\mathsf{srs_u} = \left(\{[\tau^i]_1\}_{i=1}^{\rho}, [\tau]_2\right)$, and the derived one $\mathsf{srs}_{\mathbf{W}}$ consists of the evaluation in x of the polynomials that $\mathcal{I}_{\mathsf{CSS}}$ outputs. Prover and Verifier instantiate $\mathcal{P}_{\mathsf{lite}}$ and $\mathcal{V}_{\mathsf{lite}}$ (for the PHP of Fig. 2 that achieves zero-knowledge through the changes presented in Fig. 3), and all oracle polynomials output by $\mathcal{P}_{\mathsf{lite}}$ are translated into polynomials evaluated (in the source group) at τ. For all degree checks with $\deg(p) < \mathsf{dg}$, $\mathsf{dg} < \rho$, the prover sends a single extra polynomial and field element, while checks for $\mathsf{dg} \geq \rho$ are for free. For each polynomial equation, prover sends extra field elements corresponding to evaluations (or openings) of some of the polynomials involved on it (maximum one per quadratic term, due to the procedure stated in [20] attributed to M. Maller). There are several ways to do this compilation check, but to optimize efficiency the choices are quite standard (for instance, only $A'(X)$ or $B'(X)$, should be opened). All the openings at one point as well as the degrees of the opened polynomials can be proven with one group element and verified with one pairing. Prover's work includes running $\mathcal{P}_{\mathsf{lite}}$ as well as the computation of the polynomial commitment opening procedures. Verifier work is also $\mathcal{V}_{\mathsf{lite}}$ plus the (batched) verification procedure of the polynomial commitments. The vector of queries is $(\mathsf{b}_A, \mathsf{b}_B, \mathsf{b}_{R_t}, \mathsf{b}_{H_t}) = (1, 0, 1, 0)$.

On the other hand, we write the matrix $\mathbf{W}$ that expresses the constraints as:

$$\mathbf{W} = \begin{pmatrix} \mathbf{I}_m & \mathbf{0}_{m\times n} & \mathbf{0}_{m\times m} & \mathbf{0}_{m\times n} & -\mathbf{F} & \mathbf{0}_{m\times n} \\ \mathbf{0}_{m\times m} & \mathbf{0}_{m\times n} & \mathbf{I}_m & \mathbf{0}_{m\times n} & -\mathbf{G} & \mathbf{0}_{m\times n} \\ \mathbf{0}_m^\top & \mathbf{1}_n^\top & \mathbf{0}_m^\top & \mathbf{1}_n^\top & \mathbf{0}_{m\times m}^\top & \mathbf{0}_{m\times n}^\top \end{pmatrix} = \begin{pmatrix} \mathbf{I}' & \mathbf{0} & \mathbf{F}' \\ \mathbf{0} & \mathbf{I}' & \mathbf{G}' \\ \boldsymbol{w} & \boldsymbol{w} & \mathbf{0} \end{pmatrix},$$

where $\mathbf{I}', \mathbf{F}', \mathbf{G}'$ are of size $m \times (m+n)$, $\boldsymbol{w}$ is a row vector of length $m + n$.

Our PHP is built generically for any CSS scheme, but concrete efficiency depends on the specifics of the latter and also how the blocks of rows of $\mathbf{W}$ are combined into it. The last constraint will always be treated separately (to exploit the symmetry of the other blocks), and because of its simple form, the verifier can compute the corresponding $\boldsymbol{D}(X) = (\sum_{i=m+1}^{m+n} \lambda_i(x), \sum_{i=m+1}^{m+n} \lambda_i(x), 0)$ itself, and combine it with the rest by adding (see Section 4.4). Below we discuss concrete costs of each of the CSS arguments for the other two blocks.

For the sparse matrice construction of Fig. 5, we assume that $K \geq 2m$, which sets $\rho = K - 1$. This eliminates the degree checks for $e_x(X), e_y(X), R_u(X)$. Assuming $K \geq |\mathbf{F}| + |\mathbf{G}|$, the indexer is run for a matrix $\mathbf{F} + Z\mathbf{G}$, where Z is a variable and thus outputs one polynomial $v_r(X)$, one polynomial $v_{1,c}(X)$ but two polynomials $v_{2,c}^F(X), v_{2,c}^G(X)$ that will let the verifier construct $v_{2,c}(X) = v_{2,c}^F(X) + z v_{2,c}^G(X)$ after choosing challenge z. For $\mathbf{I}'$ it is not necessary to run a CSS argument, as for this block the corresponding polynomial $\boldsymbol{D}(X)$ is $D_{\mathbf{I}'}(X) = \sum_{i=1}^{m} \lambda_i(x)\lambda_i(X)$ and thus $D_{\mathbf{I}'}(y)$ can be calculated by the verifier in $\log m$ time as $(xt(y) - yt(x))/(x - y) - \sum_{i=m+1}^{m+n} \lambda_i(x)\lambda_i(y)$.

References

1. B. Abdolmaleki, K. Baghery, H. Lipmaa, and M. Zajac. A subversion-resistant SNARK. In T. Takagi and T. Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 3–33, Hong Kong, China, Dec. 3–7, 2017. Springer, Heidelberg, Germany. 5
2. S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 2087–2104, Dallas, TX, USA, Oct. 31 – Nov. 2, 2017. ACM Press. 1, 6
3. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. `https://eprint.iacr.org/2018/046`. 1, 6
4. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable zero knowledge with no trusted setup. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 701–732, Santa Barbara, CA, USA, Aug. 18–22, 2019. Springer, Heidelberg, Germany. 1
5. E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. Aurora: Transparent succinct arguments for R1CS. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. 3, 4, 6, 11, 13
6. E. Ben-Sasson, A. Chiesa, and N. Spooner. Interactive oracle proofs. In M. Hirt and A. D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60, Beijing, China, Oct. 31 – Nov. 3, 2016. Springer, Heidelberg, Germany. 5
7. M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112, Chicago, IL, USA, May 2–4, 1988. ACM Press. 1
8. J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. 1, 3, 6
9. S. Bowe, A. Gabizon, and M. D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK. In A. Zohar, I. Eyal, V. Teague, J. Clark, A. Bracciali, F. Pintore, and M. Sala, editors, *FC 2018 Workshops*, volume 10958 of *LNCS*, pages 64–77, Nieuwpoort, Curaçao, Mar. 2, 2019. Springer, Heidelberg, Germany. 1
10. S. Bowe, A. Gabizon, and I. 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`. 1
11. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press. 1, 6
12. B. Bünz, B. Fisch, and A. Szepieniec. Transparent SNARKs from DARK compilers. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. 2

13. M. Campanelli, A. Faonio, D. Fiore, A. Querol, and H. Rodríguez. Lunar: a toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. Cryptology ePrint Archive, Report 2020/1069, 2020. `https://eprint.iacr.org/2020/1069`. 2, 4, 5, 6, 8, 13, 19, 20, 27
14. A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. 2, 4, 5, 8, 13, 19
15. A. Chiesa, D. Ojha, and N. Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 769–793, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. 1
16. V. Daza, C. Ràfols, and A. Zacharakis. Updateable inner product argument with logarithmic verifier and applications. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 527–557, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany. 2
17. G. Fuchsbauer. Subversion-zero-knowledge SNARKs. In M. Abdalla and R. Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347, Rio de Janeiro, Brazil, Mar. 25–29, 2018. Springer, Heidelberg, Germany. 5
18. G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Heidelberg, Germany. 5, 10, 11
19. A. 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`. 2
20. A. Gabizon, Z. J. Williamson, and O. 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`. 2, 4, 5, 8, 27
21. S. Garg, M. Mahmoody, D. Masny, and I. Meckler. On the round complexity of OT extension. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 545–574, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Heidelberg, Germany. 8
22. R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany. 1, 2, 13, 19
23. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proofs. In *SIAM Journal on Computing*, pages 186–208, 1989. 1
24. J. Groth. Linear algebra with sub-linear zero-knowledge arguments. In S. Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 192–208, Santa Barbara, CA, USA, Aug. 16–20, 2009. Springer, Heidelberg, Germany. 3
25. J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340, Singapore, Dec. 5–9, 2010. Springer, Heidelberg, Germany. 1
26. J. Groth. On the size of pairing-based non-interactive arguments. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. 1

27. J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Heidelberg, Germany. 2, 5
28. Y. Ishai. Zero-knowledge proofs from information theoretic proof systems. In *Zkproofs Blog, https://zkproof.org/2020/08/12/information-theoretic-proof-systems/*, 2020. 2
29. C. S. Jutla and A. Roy. Shorter quasi-adaptive NIZK proofs for linear subspaces. In K. Sako and P. Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 1–20, Bengalore, India, Dec. 1–5, 2013. Springer, Heidelberg, Germany. 3
30. A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194, Singapore, Dec. 5–9, 2010. Springer, Heidelberg, Germany. 2, 5
31. A. Kattis, K. Panarin, and A. Vlasov. RedShift: Transparent SNARKs from list polynomial commitment IOPs. Cryptology ePrint Archive, Report 2019/1400, 2019. `https://eprint.iacr.org/2019/1400`. 2
32. J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732, Victoria, BC, Canada, May 4–6, 1992. ACM Press. 1
33. M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, Nov. 11–15, 2019. 2, 4, 5
34. S. Micali. The knowledge complexity of interactive proofs. In *SIAM Journal on Computing 30 (4)*, pages 1253–1298, 2000. 1
35. P. Morillo, C. Ràfols, and J. L. Villar. The kernel matrix Diffie-Hellman assumption. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 729–758, Hanoi, Vietnam, Dec. 4–8, 2016. Springer, Heidelberg, Germany. 15
36. C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 222–242, Tokyo, Japan, Mar. 3–6, 2013. Springer, Heidelberg, Germany. 4, 22
37. S. Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737, Santa Barbara, CA, USA, Aug. 17–21, 2020. Springer, Heidelberg, Germany. 2, 6
38. A. Szepieniec and Y. Zhang. Polynomial iops for linear algebra relations. Cryptology ePrint Archive, Report 2020/1022, 2020. `https://eprint.iacr.org/2020/1022`. 2, 4, 19
39. R. S. Wahby, I. Tzialla, a. shelat, J. Thaler, and M. Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press. 1, 6
40. T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 733–764, Santa Barbara, CA, USA, Aug. 18–22, 2019. Springer, Heidelberg, Germany. 1