An Algebraic Framework for Universal and Updatable SNARKs

Carla Ràfols[⋆,1,2] and Arantxa Zapico[⋆⋆,1]

[1] Universitat Pompeu Fabra
[2] Cybercat
carla.rafols@upf.edu, arantxa.zapico@upf.edu

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 tradeoffs. Our most efficient construction, Basilisk, seems to have the smallest proof size in the literature, although it pays a price in terms of structure reference string for the number of multiplicative gates whose fan-out exceeds a certain bound.

1 Introduction

Zero-Knowledge proofs [GMR89], and in particular, non-interactive ones [BFM88] have played a central role in both the theory and practice of cryptography. A long line of research [Kil92, Mic00, Gro10, GGPR13, Gro16] 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 [Gro16]).

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) [BBB$^+$18, BCC$^+$16, BBHR18, AHIV17, BBHR19, COS20, XZZ$^+$19, WTs$^+$18], pairing-based SNARKs such as [Gro16] 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 [BGG19, BGM17].

As an alternative to a trusted SRS, Groth et al. [GKM$^+$18] 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 [GGPR13]. On the other

⋆ 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.

hand, affine constraints are circuit-dependent, and it is a challenging task to efficiently prove them with a universal SRS [MBKM19, CHM+20, GWC19, CFF+20, DRZ20, Set20, Gab19, SZ20].

In Groth et al. [GKM+18] 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 [MBKM19], 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 [CHM+20], Plonk [GWC19], Lunar [CFF+20]) 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 [Ish20] 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 [KZG10, BFS20, KPV19]. 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 techniques. 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 sampled with the verifier's coins, the latter does not need to perform a linear (in Q, the number of rows) number of operations to verify that $D(X)$ is correct. Instead, this must be demonstrated 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 row vectors 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 [BCR+19]. However, we contribute a

[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.

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 [BCC$^+$16], of membership in linear spaces [JR13], or for linear algebra relations [Gro09].

Finally, we give several constructions of CSS arguments. In Section 5, we start from the representation of a matrix $\mathbf{W}$ as bivariate polynomial introduced in [CHM$^+$20] to construct an alternative CSS argument for sparse matrices. Our contribution is to introduce a new linearization step that allows us to build it modularly from an argument for what we refer to as *simple matrices*, i.e., those with at most one non-zero element per column. Compared to [CHM$^+$20, CFF$^+$20], at a minimal increase in communication cost, our argument significantly reduces the size of the derived SRS. We generalize these arguments to sums of simple and sparse matrices, without increasing the communication complexity. In Section 7, we show the helped Sonic mode works as a CSS argument in the amortized setting. In Section 6, we introduce a CSS argument that works for arbitrary matrices and, even though it requires quadratic indexer work and linear verifier memory, can be combined with other schemes to increase efficiency or expressivity, as we show in Section 4.4.

In the appendix, we introduce a generalized universal relation that captures the one considered in Plonk [GWC19] and we study the performance trade-offs resulting from the different possible choices of CSS argument of Section 5. In particular. we observe that the argument for simple matrices and sums of simple matrices is useful on its own, and not only to achieve modularity. We study the efficiency of our zkSNARK when: a) the CSS argument is our argument for sparse matrices of Section 5.3, b) when the Plonk constraint system is used and the matrix of constraints is a permutation, which is a special case of a simple matrix, and c) when the circuit has bounded fan-out, which results in a matrix of constraints that is a sum of simple matrices. To the best of our knowledge, the latter construction is the most efficient in proof size updatable and universal zkSNARK. Following the trend to give SNARKs the name of fabulous creatures, and given the importance of linear algebra and the Lagrange basis in our work, we call it BASILISK.

The efficiency of the zkSNARKs built from the different CSS arguments detailed can be found in Table 1. The details of the constructions are given in the Appendix.

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 [MBKM19], this last step is called a *signature of correct computation* [PST13] 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 [CHM$^+$20] bases its construction on the univariate sum-check protocol of Aurora [BCR$^+$19] 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 [CFF+20] uses the same representation as Marlin but improves on it, among other tweaks by introducing a new language (R1CS-lite) that can also encode arithmetic circuit satisfiability, but has a lighter representation than other constraint systems. Plonk [GWC19] does not use bivariate polynomials or require sparse matrices but the SRS size depends on the number of both multiplicative and additive gates. As we do, Plonk, Marlin and Lunar use the Lagrange interpolation basis to commit to vectors, while Claymore [SZ20] 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. Notably, it also uses implicitly a CSS argument.

Work		$\lvert\mathsf{srs}_\mathsf{u}\rvert$	$\lvert\mathsf{srs}_\mathbf{W}\rvert$	$\lvert\pi\rvert$	KeyGen	Derive	Prove	Verifier
Sonic [MBKM19]	$\mathbb{G}_1$	$4M$	-	20	$4M$	$36m$	$273m$	7P
	$\mathbb{G}_2$	$4M$	3	-	$4M$	-	-	
	$\mathbb{F}$	-	-	16	-	$O(K \log K)$	$O(K \log K)$	$O(l + \log K)$
Marlin [CHM+20]	$\mathbb{G}_1$	$3\hat{K}$	12	13	$3\hat{K}$	$12K$	$14m + 8K$	2P
	$\mathbb{G}_2$	2	2	-	-	-	-	
	$\mathbb{F}$	-	-	8	-	$O(K \log K)$	$O(K \log K)$	$O(l + \log K)$
Plonk [GWC19]	$\mathbb{G}_1$	$3N$	8	7	$3N$	$8n$	$11n$	2P
	$\mathbb{G}_2$	1	1	-	-	-	-	
	$\mathbb{F}$	-	-	7	-	$O(n \log n)$	$O(n \log n)$	$O(l + \log n)$
LunarLite2x[4] [CFF+20]	$\mathbb{G}_1$	$\hat{K}$	16	11	$\hat{K}$	$16K$	$8m + 4K$	2P
	$\mathbb{G}_2$	1	1	-	1	-	-	
	$\mathbb{F}$	-	-	3	-	$O(K \log K)$	$O(K \log K)$	$O(l + \log K)$
This work Sec. 5.3	$\mathbb{G}_1$	$\hat{K}$	4	10	$\hat{K}$	$6K$	$6m + 4K$	2P
	$\mathbb{G}_2$	1	1	-	-	-	-	
	$\mathbb{F}$	-	-	3	-	$O(K \log K)$	$O(K \log K)$	$O(l + \log K)$
This work Fig. 11	$\mathbb{G}_1$	N	11	8	N	$11n$	$8n$	2P
	$\mathbb{G}_2$	1	1	-	-	-	-	
	$\mathbb{F}$	-	-	4	-	$O(n \log n)$	$O(n \log n)$	$O(l + \log n)$
Basilisk App. F	$\mathbb{G}_1$	M	$3V + 1$	6	M	$(3v + 1)m$	$6m$	2P
	$\mathbb{G}_2$	1	1	-	-	-	-	
	$\mathbb{F}$	-	-	2	-	$O(m \log m)$	$O(m \log m)$	$O(l + \log m)$

Table 1. Comparison with state of the art universal and updatable zkSNARKs. m: number of multiplicative gates, n: number of total gates, v : bounded fan-out, K: non-zero elements of the matrix that describe the circuit ($|\mathbf{F} + \mathbf{G}|$ in our case). $N, \hat{K}, V, M$: maximum supported values for n, K, m, v. $N^* = M + A$. The numbers for our work take into account the trick to eliminate non-trivial degree checks in App.E.

Information Theoretic Proof Systems. All these previous works 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) [BCS16].

[3] Among all schemes with different trade-offs presented in Lunar, we highlight this one as it is the most directly comparable to our scheme.

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 notion is similar to the one 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. [KZG10]. Sonic gave a proof of extractability of the latter in the Algebraic Group Model [FKL18], 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 same definitions and construction as these works. The formal definition is given in Section 2.4.

Untrusted Setup. The original constructions of pairing-based zkSNARKs crucially depend for soundness on a trusted setup, although, as was shown in [ABLZ17, Fuc18], the zero-knowledge property is still easy to achieve when the setup is subverted. Groth et al. introduced the updatable SRS model in [GKM$^+$18] to address the issue of trust in SRS generation. There are several alternatives to achieve transparent setup and constant-size proofs, but all of them have either linear verifier [BCC$^+$16, BBB$^+$18, BCR$^+$19, AHIV17], or work only for very structured types of computation [BBHR18, WTs$^+$18]. An exception is the work of Setty [Set20]. Concretely, the 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 $\mathcal{R}'_{\text{R1CS-lite}}$, an equivalent of the relation $\mathcal{R}_{\text{R1CS-lite}}$ introduced in Lunar [CFF$^+$20]. 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) Let $\mathbb{F}$ be a finite field and $m, l, s \in \mathbb{N}$. We define the universal relation R1CS as:*

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

Definition 2. *(R1CS-*lite*) Let $\mathbb{F}$ be a finite field and $m, l, s \in \mathbb{N}$. We define the universal relation R1CS-*lite *as:*

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

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

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

To see they are equivalent, observe that, if in $\mathcal{R}'_{\text{R1CS-lite}}$ 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([CFF+20]).

In Appendix A we introduce $\mathcal{R}_{\text{W-R1CS}}$, a generalization of these relations that can also express the relation considered in Plonk.

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 3. *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:*

- $(\mathsf{srs}_\mathsf{u}, \tau) \leftarrow \mathsf{KeyGen}(\mathcal{R})$*: On input a family of relations $\mathcal{R}$,* KeyGen *outputs a universal structured common reference string* srs_u *and a trapdoor τ;*
- $\mathsf{srs}_\mathsf{R} \leftarrow \mathsf{KeyGenD}(\mathsf{srs}_\mathsf{u}, \mathsf{R})$*: On input $\mathsf{R} \in \mathcal{R}$, this algorithm outputs a relation dependent SRS that includes* srs_u*;*
- $\pi \leftarrow \mathsf{Prove}(\mathsf{R}, \mathsf{srs}_\mathsf{R}, (\mathsf{x}, \mathsf{w}))$*: On input the relation,* srs_R *and a pair* $(\mathsf{x}, \mathsf{w}) \in \mathsf{R}$*, it outputs a proof π;*
- $1/0 \leftarrow \mathsf{Verify}(\mathsf{srs}_\mathsf{R}, \mathsf{x}, \pi)$*:* Verify *takes as input* 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 4. *Completeness holds if an honest prover will always convince an honest verifier. Formally,* $\forall\ \mathsf{R} \in \mathcal{R}, (\mathsf{x}, \mathsf{w}) \in \mathsf{R}$*,*

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

Definition 5. *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 6. *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:*

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

Definition 7. $(\mathsf{KeyGen}, \mathsf{KeyGenD}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Simulate})$ *is zero-knowledge (a zkSNARK) if for all* $\mathsf{R} \in \mathcal{R}$*, instances* x *and PPT adversaries* $\mathcal{A}$*.*

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

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

Updatability. We will say a universal zkSNARK is *updatable* if srs_u is updatable as defined in [GMMM18]. 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. [CFF+20]. 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) [CHM+20] or idealized polynomial protocols [GWC19].

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 [CFF+20] 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 [CFF+20].

Definition 8. *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 9 (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* $\mathsf{rnd}, \mathsf{n}, \mathsf{m}, \mathsf{d}, \mathsf{n}_e : \{0,1\}^* \to \mathbb{N}$ *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 10. *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 11. *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 12. *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 13. *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* $\mathsf{View}\big(\mathcal{S}^{\mathcal{V}^*}(\mathsf{R}, \mathsf{x})\big)$ *consists of* $\mathcal{V}^*$*'s randomness followed by* $\mathcal{S}$*'s output, obtained after having straightline access to* $\mathcal{V}^*$*, and* $\mathcal{V}^*$*'s list of checks.*

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 14. *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 15. *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 Polynomial Commitments

Definition 16 (Polynomial Commitment Scheme). *A* Polynomial Commitment Scheme *is a tuple of algorithms* $(\mathsf{PC.KeyGen}, \mathsf{PC.Commit}, \mathsf{PC.Open}, \mathsf{PC.Verify})$ *such that:*

- $\mathsf{srs}_{\mathsf{PC}} \leftarrow \mathsf{PC.KeyGen}(\mathsf{pp}_{\mathsf{PC}}, d)$*: On input the system parameters and a degree bound* d*, it outputs a structured reference string.*
- $\mathbf{c} \leftarrow \mathsf{PC.Commit}(\mathsf{srs}_{\mathsf{PC}}, \boldsymbol{p}(X), \boldsymbol{d})$*: On input the srs and a vector of* t *polynomials* $p_i(X)$ *of degree up to* d_i*, it outputs a vector* $\mathbf{c}$ *where* c_i *is a commitment to* $p_i(X)$*.*
- $(\boldsymbol{s}, \pi_{\mathsf{PC}}) \leftarrow \mathsf{PC.Open}(\mathsf{srs}_{\mathsf{PC}}, \boldsymbol{p}(X), \boldsymbol{d}, \mathcal{Q}, \gamma)$*: On input the srs, the vector of polynomials, the degree bounds, a query set* $\mathcal{Q}$ *where each query is a tuple* $(i, z) \in [t] \times \mathbb{F}$*, and a opening challenge* γ*, it outputs a vector of evaluations* $\boldsymbol{s}$ *and an evaluation proof* π_{PC}*.*
- $1/0 \leftarrow \mathsf{PC.Verify}(\mathsf{srs}_{\mathsf{PC}}, \mathbf{c}, \boldsymbol{d}, \mathcal{Q}, \gamma, \boldsymbol{s}, \pi_{\mathsf{PC}})$*: On input the srs, the vector of commitments, the degree bounds, the query set, the opening challenge, a vector of evaluations* $\boldsymbol{s} = (s_{i,z})_{(i,z)\in\mathcal{Q}}$*, and the proof of correct evaluation, it outputs a bit indicating acceptance or rejection.*

A polynomial commitment scheme should satisfy the following properties:

Completeness: *It captures the fact that an honest prover will always convince an honest verifier. Formally, for any vector of polynomials* $\boldsymbol{p}(X)$ *such that* $\deg(p_i) \leq d_i$ *and set of queries* $\mathcal{Q}$ *the following probability is 1:*

$$\mathsf{Pr}\left[\begin{array}{c} \deg(p_i) \leq d_i \Rightarrow \\ \mathsf{PC.Verify}(\mathsf{srs}_{\mathsf{PC}}, \mathbf{c}, \boldsymbol{d}, \mathcal{Q}, \gamma, \boldsymbol{s}, \pi_{\mathsf{PC}}) = 1 \end{array} \middle| \begin{array}{r} \mathsf{srs}_{\mathsf{PC}} \leftarrow \mathsf{PC.KeyGen}(\mathsf{pp}_{\mathsf{PC}}, d) \\ \mathbf{c} \leftarrow \mathsf{PC.Commit}(\mathsf{srs}_{\mathsf{PC}}, \boldsymbol{p}(X), \boldsymbol{d}) \\ \boldsymbol{s}_{(i,z)} = p_i(z), (i,z) \in \mathcal{Q} \\ (\boldsymbol{s}, \pi_{\mathsf{PC}}) \leftarrow \mathsf{PC.Open}(\mathsf{srs}_{\mathsf{PC}}, \boldsymbol{p}(X), \boldsymbol{d}, \mathcal{Q}, \gamma) \end{array}\right]$$

Soundness: *Captures the fact that a cheating prover should not be able to convince the verifier of a false opening. Formally, for all PPT adversaries* $\mathcal{A}$*:*

$$\mathsf{Pr}\left[\begin{array}{c} \exists\ (i,z) \in \mathcal{Q} \text{ s.t. } p_i(z) \neq \boldsymbol{s}_{(i,z)}, \quad \text{or} \\ \exists i \in [t] \text{ s.t. } deg(p_i) > d_i, \text{ and} \\ \mathsf{PC.Verify}(\mathsf{srs}_{\mathsf{PC}}, \mathbf{c}, \boldsymbol{d}, \mathcal{Q}, \gamma, \boldsymbol{s}, \pi_{\mathsf{PC}}) = 1 \end{array} \middle| \begin{array}{r} \mathsf{srs}_{\mathsf{PC}} \leftarrow \mathsf{PC.KeyGen}(\mathsf{pp}_{\mathsf{PC}}, d) \\ \mathbf{c} \leftarrow \mathcal{A}(\mathsf{srs}_{\mathsf{PC}}, \boldsymbol{p}(X), \boldsymbol{d}) \\ (\boldsymbol{s}, \pi_{\mathsf{PC}}) \leftarrow \mathcal{A}(\mathsf{srs}_{\mathsf{PC}}, \boldsymbol{p}(X), \boldsymbol{d}, \mathcal{Q}, \gamma) \end{array}\right] \approx 0$$

Extractability: *Captures the fact that whenever the prover provides a valid opening, it knows a valid pair* $(p_i(X), p_i(z)) \in \mathbb{F}[X] \times \mathbb{F}$*, where* $\deg(p_i) \leq d_i$*. Formally, for all PPT adversaries* $\mathcal{A}$ *there exists an efficient extractor* $\mathcal{E}$ *such that:*

$$\mathsf{Pr}\left[\begin{array}{c} \mathsf{PC.Verify}(\mathsf{srs}_{\mathsf{PC}}, \mathbf{c}, \boldsymbol{d}, \mathcal{Q}, \gamma, \boldsymbol{s}, \pi_{\mathsf{PC}}) = 1 \\ \wedge \\ \exists\ (i,z) \in \mathcal{Q} \text{ s.t. } p_i(z) \neq \boldsymbol{s}_{(i,z)}, \quad \text{or} \\ \exists i \in [t] \text{ s.t. } deg(p_i) > d_i \end{array} \middle| \begin{array}{r} \mathsf{srs}_{\mathsf{PC}} \leftarrow \mathsf{PC.KeyGen}(\mathsf{pp}_{\mathsf{PC}}, d) \\ \mathbf{c} \leftarrow \mathcal{A}(\mathsf{srs}_{\mathsf{PC}}) \\ \boldsymbol{p}(X) \leftarrow \mathcal{E}(\mathsf{srs}_{\mathsf{PC}}, \mathbf{c}, \boldsymbol{d}) \\ (\mathcal{Q}, \gamma) \leftarrow \mathcal{A}(\mathsf{srs}_{\mathsf{PC}}, \mathbf{c}, \boldsymbol{d}) \\ (\boldsymbol{s}, \pi_{\mathsf{PC}}) \leftarrow \mathcal{A}(\mathsf{srs}_{\mathsf{PC}}, \boldsymbol{p}(X), \boldsymbol{d}, \mathcal{Q}, \gamma) \end{array}\right] \approx 0$$

2.5 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. ([FKL18]). 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 17 (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:*

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

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

Definition 18 (q-dlog Asymmetric Assumption). *The $q(\lambda)$-discrete logarithm assumption holds for $gk \leftarrow \mathcal{G}(1^\lambda)$ if for all PPT algorithm $\mathcal{A}$*

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

3 Generalized Univariate Sumcheck

In this section, we revisit the sumcheck of Aurora [BCR+19]. As presented there, this argument allows to prove that the sum of the evaluations of a polynomial in some multiplicative subgroup[5] $\mathbb{H}$ of a finite field $\mathbb{F}$ sum to some value σ. 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 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 $\mathbb{H}$.

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 $P(X)N_{\mathcal{S},v}(X) - \sigma$. □

[5] 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.

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 [BCR$^+$19, CHM$^+$20, CFF$^+$20] 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, if $A(X) = \boldsymbol{a}^\top\boldsymbol{\lambda}(X)$, $B(X) = \boldsymbol{b}^\top\boldsymbol{\lambda}(X)$ 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. [GGPR13] 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 [BCR$^+$19, CHM$^+$20, CFF$^+$20].

Theorem 2 (Inner Product Polynomial Relation). *For some $k \in \mathbb{N}$, let $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k)$, $\boldsymbol{y}_i = (y_{ij})$, $\boldsymbol{d} = (\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k)$ 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, $\boldsymbol{y}\cdot\boldsymbol{d} = \sigma$ 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:*

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

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

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

By Theorem 1, $\left(\left(\sum_{i=1}^{k}(\boldsymbol{y}_i\circ\boldsymbol{d}_i)^\top\right)\boldsymbol{\lambda}(X)\right)N_{\mathbb{H},0}(X) - \sigma$ 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 σ. 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 $N_{\mathbb{H},0}(X)$).

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 encodings 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 19 (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:*

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

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 $\boldsymbol{\beta}(X) = (\beta_1(X), \ldots, \beta_m(X))$ 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. In fact, the CSS argument in the amortized setting described in Section 7 is an abstraction of the helped mode of Sonic, that was presented for the encoding with 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 $\mathcal{P}^*_{\mathsf{CSS}}$:

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

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 20. *A CSS argument is ϵ-elusive kernel*[6] *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. An exception is the argument of Section 6.

It is useful in some contexts to generalize the definition of CSS arguments to block matrices, that is, to extend the relation to tuples $\big(\mathbf{M}, \mathsf{cns}, \boldsymbol{D}(X)\big)$, where $\mathbf{M} = (\mathbf{M}_1, \ldots, \mathbf{M}_k)$ and $\boldsymbol{D}(X) = (D_1(X), \ldots, D_k(X))$ and $D_i(X) = \boldsymbol{s}^\top \mathbf{M}_i \boldsymbol{\beta}(X)$, and $\mathbf{M}_i \in \mathbb{F}^{Q \times m}$. This generalization is not necessary if correct sampling is proven for each block individually, but to save on proof size the proofs might be aggregated in some cases. This generalization is useful to formalize this technique.

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 $\boldsymbol{d}^\top = (\boldsymbol{d}_1^\top, \ldots, \boldsymbol{d}_k^\top)$.

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,

[6] The name is inspired by the property of t-elusiveness of [MRV16].

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

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

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:

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

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, $\boldsymbol{Y}^*(X) \cdot \boldsymbol{D}(X) = XR_t(X) + t(X)H_t(X)$ 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. A similar construction for the generalized relation $\mathcal{R}_{\text{W-R1CS}}$ can be found in Appendix B.

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 [CFF+20, CHM+20, GGPR13] 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 [GGPR13], $h(X)$ 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 [SZ20]. 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 $\mathcal{R}'_{\text{R1CS-lite}}$ 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 [CFF+20](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 $\mathbb{F} \setminus \mathbb{H}$. 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. □

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 $\mathbf{M}_i$[7] 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 blocks that admit different efficient CSS 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. Alternatively, the extended Vandermonde approach of Section 6 could also be used for the very dense rows. 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

[7] The naive approach would run both CSS arguments in parallel, but savings are possible by batching the proofs.

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.

We note that this cannot be done generically. The success of this technique 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 Checkable Subspace Sampling Arguments for Sparse Matrices

Given the results of the previous sections, for our R1CS-lite argument it is sufficient to design a CSS scheme 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 sparse 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 schemes 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 [PST13]). 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 a basic CSS scheme, that works only for *simple matrices*, i.e., 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 $\mathbf{M}$ with at most K non-zero elements, where K is the size of a multiplicative subgroup $\mathbb{K} \subset \mathbb{F}$. In Section 5.4, we give an extension of the basic construction to matrices with at most V non-zero elements per column, for some small bound V. This technique is used in Section 5.5 to generalize the second argument to matrices that can be written as a sum of V matrices of sparsity K, resulting on a scheme for matrices with sparsity VK that uses the same multiplicative subgroup and does not increase the communication complexity with respect to the one for matrices with sparsity K. Finally, in Section 5.6 we observe that our results also apply to low tensor rank matrices.

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:

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

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 of Section 3 to prove that the sum of $e_x(X)e_y(X) \mod t(X)$ at $\mathbb{K}$ is σ.

Observe that $e_x(X) = \boldsymbol{\lambda}(x)^\top \mathbf{M}_x \boldsymbol{\mu}(X)$ and $e_y(X) = \boldsymbol{\lambda}(y)^\top \mathbf{M}_y \boldsymbol{\mu}(X)$, 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. 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 ones in Sections 5.4, 5.5 and 5.6. The argument for basic matrices is also the key to our most efficient construction, as discussed in Appendix C.

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, $|\mathbf{M}| \leq K$. 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) $\deg D(X) \leq K-1$, 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 Fig. 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 but 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 Fig. 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 $D(X)$sma 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 $e_x(X)e_y(X) = \sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{c}(\mathsf{k}_\ell)}(y)\lambda_{\mathsf{r}(\mathsf{k}_\ell)}(x)\mu_\ell(X) \mod u(X)$. 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: Sampling: $\mathcal{V}_{\mathsf{CSS}}$ sends $x \leftarrow \mathbb{F}$, and $\mathcal{P}$ outputs $D(X) = P(X, x)$, for $P(X, Y) = \sum_{\ell=1}^{K} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{r}(\mathsf{k}_\ell)}(Y)\lambda_{\mathsf{c}(\mathsf{k}_\ell)}(X)$.
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) $\deg(R_u) \leq K - 2$, (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 $|\mathbf{M}| \leq |\mathbb{K}|$.

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 Section 5.5. Also, in Appendix C we use this argument in the context where $\mathbf{M}$ is a matrix in $\mathbb{F}^{K\times m}$. In that case, the role of the multiplicative subgroups $\mathbb{K},\mathbb{H}$ should be inversed.

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 $1,x,\ldots,x^V$.

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

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

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) $\deg(D)\le K-1$, 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 Fig. 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:

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

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. □

5.5 CSS Argument for Sums of Sparse Matrices

The argument for general sparse matrices of last section can be easily generalized without increasing the communication complexity to any matrix $\tilde{\mathbf{M}}$ such that $\tilde{\mathbf{M}} = \sum_{i=1}^{V} \tilde{\mathbf{M}}_i$, where there exists one function $\mathsf{r} : \mathbb{K} \to [m]$, and, for each i, two functions $\mathsf{c}_i : \mathbb{K} \to [m]$, and $\mathsf{v}_i : \mathbb{K} \to \mathbb{F}$, such that: $\boldsymbol{\lambda}(X)^\top \tilde{\mathbf{M}}_i \boldsymbol{\lambda}(Y) = \tilde{P}_i(X, Y) = \sum_{\ell=1}^{K} \mathsf{v}_i(\mathsf{k}_\ell) \lambda_{\mathsf{r}(\mathsf{k}_\ell)}(Y) \lambda_{\mathsf{c}_i(\mathsf{k}_\ell)}(X)$,

Choosing the row and the column function smartly, this can cover many sparse matrices with KV non-zero entries, considerably increasing the expressiveness of the CSS argument. For this generalization, we observe that if $\tilde{P}(X, Y) = \sum_{i=1}^{V} \tilde{P}_i(X, Y)$, then

$$\tilde{P}(y, x) = \big(\lambda_{\mathsf{r}(\mathsf{k}_1)}(x), \ldots, \lambda_{\mathsf{r}(\mathsf{k}_K)}(x) \big) \cdot \sum_{i=1}^{V} \big(\mathsf{v}_i(\mathsf{k}_1) \lambda_{\mathsf{c}_i(\mathsf{k}_1)}(y), \ldots, \mathsf{v}_i(\mathsf{k}_K) \lambda_{\mathsf{c}_i(\mathsf{k}_K)}(y) \big).$$

We can define $e_x(X)$ as Section 5.3, and $e_y(X) = \sum_{i=1}^{V} \sum_{\ell=1}^{K} \mathsf{v}_i(\mathsf{k}_\ell) \lambda_{\mathsf{c}_i(\mathsf{k}_\ell)}(y) \mu_\ell(X)$. Thus, $e_y(X) = \boldsymbol{\lambda}(Y)^\top \mathbf{M}_y \boldsymbol{\mu}(X)$, where $\mathbf{M}_y$ is a matrix with at most V non-zero entries in each column. The CSS is constructed as in the one for sparse matrices of Section 5.3, except that to prove that $e_y(X)$ is correctly sampled, we use the CSS for sums of basic matrices of Section 5.4. Note that this change does not represent an increase in the communication complexity with respect to Section 5.3, only in the SRS size.

5.6 Extension to Low Tensor Rank Matrices

Similar techniques to the ones in Section 5.1 can be used to construct a CSS scheme for matrices that are not sparse but for which a representation of low tensor rank is known. A matrix $\mathbf{M} \in \mathbb{F}^{m \times m}$ has tensor rank r if there exist vectors $\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i \in \mathbb{F}^m$, $i \in [r]$ such that $\mathbf{M} = \sum_{i=1}^{r} \boldsymbol{\alpha}_i \boldsymbol{\beta}_i^\top$. The main observation is that, in this case, $P(y, x) = \boldsymbol{\lambda}(x)^\top \mathbf{M} \boldsymbol{\lambda}(y) = \sum_i (\boldsymbol{\lambda}(x)^\top \boldsymbol{\alpha}_i) \cdot (\boldsymbol{\beta}_i^\top \boldsymbol{\lambda}(y))$. For each i, we can compute low degree extensions of $(\boldsymbol{\lambda}(x)^\top \boldsymbol{\alpha}_i)$ and $(\boldsymbol{\lambda}(y) \boldsymbol{\beta}_i^\top)$ as before (but taking $\mathbb{K} = \mathbb{H}$), and prove correctness with the basic CSS scheme of Section 5.2. Then, we can use the sumcheck theorem to see that $\sigma_{x,i} = \boldsymbol{\lambda}(x)^\top \boldsymbol{\alpha}_i$, and $\sigma_{y,i} = \boldsymbol{\beta}_i^\top \boldsymbol{\lambda}(y)$, and check $P(y, x) = \sum_{i=1}^{r} \sigma_{x,i} \sigma_{y,i}$. Naturally, the communication complexity depends on the tensor rank.

There is no reason to expect that in practice the tensor rank will be low and, further, in general it is hard to compute. But we think it is of theoretical value to note that sparsity is not always the key for building efficient CSS schemes.

6 A Simple CSS: Extended Vandermonde Sampling

The constructions discussed in the previous section impose (once the finite field is fixed) some conditions of the type of admissible matrices considered by the CSS scheme. For many practical use cases, this does not seem to be a limitation. However, regardless of the types of constraints that appear in applications so far, we think it is interesting to explore ways of constructing CSS arguments for more general matrices both for future applications and for theoretical understanding.

The most trivial CSS scheme for a matrix $\mathbf{M} \in \mathbb{F}^{Q \times m}$ works as follows: indexer sends Q oracle polynomials, one for each row, as $P_i(X) = \sum_{j=1}^{m} m_{ij} \lambda_j(X)$. The verifier samples $x \leftarrow \mathbb{F}$, and both prover and verifier compute the same polynomial $D(X) = \sum_{i=1}^{Q} x^{i-1} P_i(X)$, the verifier only accepts if the prover sends the same $D(X)$ it computed itself. This "Vandermonde Sampling" of polynomials associated with the row

space of $\mathbf{M}$ requires $\mathcal{W}_{\mathsf{CSS}}$ size and prover work to be linear in Q. When using this argument as part of a zkSNARK, the verifier will be linear in the circuit size, which is completely impractical in most scenarios.

Below, we introduce a simple extension of the "Vandermonde sampling" technique, but trading memory for verifier work. This is impractical if $\mathbf{M}$ is the matrix that encodes the circuit's affine constraints, as $Q \approx m$. However, since this CSS scheme works for any arbitrary $\mathbf{M}$, it is interesting to combine it as explained in Section 4.4 with other approaches: for example, this CSS argument can be used to encode a few very dense constraints, and the approach in Section 5.3 can be used for the rest.

The argument depends on two parameters J, ℓ: $J = |\mathcal{J}|$ is the number of exponentiations that the verifier does, and ℓ defines the size of the SRS. As we will prove, the argument is Elusive Kernel with probability $\epsilon = \left(\frac{Q}{Q+\ell}\right)^J$. Fixing the soundness error to some λ, one can derive a trade-off between the size of J, ℓ. Taking ℓ as some constant multiple of Q, for having low verifier work, indexer work would be $O(Qm + Q^2)$ and verifier memory $O(Q)$. Again, this only makes sense when Q represents some small set of constraints.

Offline Phase: $\mathcal{I}_{\mathsf{CSS}}(\mathbb{F}, \mathbf{M}, J, \ell)$: For all $i \in [Q]$, defines the polynomials $P_i(X) = \sum_{j=1}^{m} m_{ij}\lambda_j(X)$. For $i \in [\ell]$, it defines $P_{Q+i}(X) = \sum_{j=1}^{Q} i^{j-1} P_j(X)$. It outputs $\mathcal{W}_{\mathsf{CSS}} = \{P_1(X), \ldots, P_{Q+\ell}(X)\}$.

Online Phase: $\mathcal{V}_{\mathsf{CSS}}$ samples $x \leftarrow \mathbb{F}$ and a set of J indices $\mathcal{J} \subset [Q+\ell]$. $\mathcal{P}_{\mathsf{CSS}}$ computes and outputs $D(X) = \sum_{i_j \in \mathcal{J}} x^{j-1} P_{i_j}(X)$.

Fig. 7. CSS argument with verifier sampling

The prover does not need to send the polynomial $D(X)$ as it is computed by the verifier, and in the decision phase the verifier will always accept, so we omit it.

Theorem 10. *The argument of Fig. 7 is perfectly complete, perfectly sound and ϵ-Elusive Kernel, for $\epsilon = \frac{J}{|\mathbb{F}|} + \left(\frac{Q}{Q+\ell}\right)^J$.*

Proof. The verifier samples $D(X)$ on its own and thus completeness and soundness follow immediately. On the other hand, the probability that $\boldsymbol{y}^*$ is not orthogonal to $\mathbf{M}$ but it is orthogonal to $\sum_{i_j \in \mathcal{J}} x^{j-1} P_{i_j}(X)$ can be upper bounded by standard techniques by $\frac{J}{|\mathbb{F}|} + \left(\frac{Q}{Q+\ell}\right)^J$. Indeed, there are two options, a) either it is orthogonal to all the vectors encoded in $\{P_{i_j}(X)\}_{i_j \in \mathcal{J}}$, or b) it is not. The probability of b) is at most $\frac{J}{|\mathbb{F}|}$ by Schwartz-Zippel. For a), note that if $\boldsymbol{y}^*$ is not orthogonal to $\mathbf{M}$, it can satisfy at most $Q-1$ constraints out of $Q+\ell$. Since the set $\mathcal{J}$ is chosen independently of $\boldsymbol{y}^*$, the probability that the set $\mathcal{J}$ coincides with constraints $\boldsymbol{m}_{i_j}$ such that $\boldsymbol{y} \cdot \boldsymbol{m}_{i_j} = 0$ is at most:

$$\frac{\binom{Q-1}{J}}{\binom{Q+\ell}{J}} \leq \left(\frac{Q}{Q+\ell}\right)^J.$$

□

7 Amortized CSS argument

In this section we present a CSS argument that works only in the *amortized* setting as considered in Sonic [MBKM19]. The construction is basically the protocol in the named work, but for a bivariate polynomial in the Lagrange basis rather than the basis of monomials.

In the amortized setting, the same verifier aims to check the output of different provers $\mathcal{P}_{\mathsf{CSS}}$ in Sampling. The cost of the verification is linear in m and thus the scheme is only recommended when the number of proofs is linear in m as well. The construction is not holographic due to the fact that the verifier needs to read the matrix $\mathbf{M}$ that describes the relation and thus the indexer is trivial.

Still, in the ProveSampling algorithm, the verifier has oracle access to a set $\mathcal{D} = \{D_1(X), \ldots, D_t(X)\}$ of polynomials where each $D_s(X)$ is the output of a different execution of Sampling with verifier's challenge x_s. Following the original definition, the verifier also has oracle access to the polynomials outputted by $\mathcal{P}_{\mathsf{CSS}}$ (instantiated by what in Sonic is called a helper) in ProveSampling.

Online Phase: $\mathcal{V}_{\mathsf{CSS}}$ samples $x_s \leftarrow \mathbb{F}$. $\mathcal{P}_{\mathsf{CSS}}$ defines $P(X,Y) = \sum_{i=1}^{m} \lambda_i(Y) P_i(X)$, for $P_i(X) = \sum_{j=1}^{m} m_{ij} \lambda_j(X)$. It outputs $D_s(X) = P(X, x_s)$.

Online Helped Phase: $\mathcal{V}_{\mathsf{CSS}}$ chooses $u_1 \leftarrow \mathbb{F}$. $\mathcal{P}_{\mathsf{CSS}}$ outputs $\tilde{D}(X) = P(u_1, X)$.

Decision Phase: Chooses $u_2 \leftarrow \mathbb{F}$, and calculate $P(u_1, u_2)$. Accept if and only if $\tilde{D}(u_2) = P(u_1, u_2)$ and, for every $\{x_s\}_{s=1}^{t}$, $\tilde{D}(x_s) = D_s(u_1)$.

Fig. 8. Amortized CSS scheme from [MBKM19].

References

ABLZ17. Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 3–33, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany. 5

AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. 1, 5

BBB+18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press. 1, 5

BBHR18. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. `https://eprint.iacr.org/2018/046`. 1, 5

BBHR19. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 701–732, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. 1

BCC+16. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. 1, 3, 5

BCR+19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. 2, 3, 5, 10, 11

BCS16. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany. 4

BFM88. Manuel Blum, Paul Feldman, and Silvio 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

BFS20. Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. 2

BG12. Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 263–280, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany. 29, 33

BGG19. Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *FC 2018 Workshops*, volume 10958 of *LNCS*, pages 64–77, Nieuwpoort, Curaçao, March 2, 2019. Springer, Heidelberg, Germany. 1

BGM17. Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017. `http://eprint.iacr.org/2017/1050`. 1

CFF+20. Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián 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, 3, 4, 5, 6, 7, 11, 16, 17, 31, 34

CHM+20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. 2, 3, 4, 7, 11, 16

COS20. Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 769–793, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. 1

DRZ20. Vanesa Daza, Carla Ràfols, and Alexandros Zacharakis. Updateable inner product argument with logarithmic verifier and applications. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 527–557, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany. 2

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. 5, 9

Fuc18. Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Heidelberg, Germany. 5

Gab19. Ariel Gabizon. AuroraLight: Improved prover efficiency and SRS size in a sonic-like system. Cryptology ePrint Archive, Report 2019/601, 2019. `https://eprint.iacr.org/2019/601`. 2

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany. 1, 11, 16

GKM+18. Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. 1, 2, 5

GMMM18. Sanjam Garg, Mohammad Mahmoody, Daniel Masny, and Izaak Meckler. On the round complexity of OT extension. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 545–574, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. 7

GMR89. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proofs. In *SIAM Journal on Computing*, pages 186–208, 1989. 1

Gro09. Jens Groth. Linear algebra with sub-linear zero-knowledge arguments. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 192–208, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany. 3

Gro10. Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany. 1

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. 1

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. `https://eprint.iacr.org/2019/953`. 2, 3, 4, 7, 28, 35

Ish20. Yuval Ishai. Zero-knowledge proofs from information theoretic proof systems. In *Zkproofs Blog, https://zkproof.org/2020/08/12/information-theoretic-proof-systems/*, 2020. 2

JR13. Charanjit S. Jutla and Arnab Roy. Shorter quasi-adaptive NIZK proofs for linear subspaces. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 1–20, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany. 3

Kil92. Joe 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

KPV19. Assimakis Kattis, Konstantin Panarin, and Alexander Vlasov. RedShift: Transparent SNARKs from list polynomial commitment IOPs. Cryptology ePrint Archive, Report 2019/1400, 2019. `https://eprint.iacr.org/2019/1400`. 2

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany. 2, 5

MBKM19. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 11–15, 2019. 2, 3, 4, 25

Mic00. Silvio Micali. The knowledge complexity of interactive proofs. In *SIAM Journal on Computing 30 (4)*, pages 1253–1298, 2000. 1

MRV16. Paz Morillo, Carla Ràfols, and Jorge Luis Villar. The kernel matrix Diffie-Hellman assumption. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 729–758, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany. 13

PST13. Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 222–242, Tokyo, Japan, March 3–6, 2013. Springer, Heidelberg, Germany. 3, 18

Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany. 2, 5

SZ20. Alan Szepieniec and Yuncong Zhang. Polynomial iops for linear algebra relations. Cryptology ePrint Archive, Report 2020/1022, 2020. `https://eprint.iacr.org/2020/1022`. 2, 4, 16

WTs+18. Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press. 1, 5

XZZ+19. Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 733–764, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. 1

A A Generalized Constraint System

In this section we present a constraint system that, inspired by Plonk, generalizes R1CS and R1CS-lite as introduced in Section 2.1. We can extend our algebraic framework to this relation to build modularly a (zk)SNARK from an argument for linear relations.

Consider the following universal relation, that depends on a set $\mathcal{M}$ of admissible matrices:

$$\mathcal{R}_{\mathsf{W\text{-}R1CS}} = \left\{ \begin{array}{c} (\mathsf{R},\mathsf{x},\mathsf{w}) := \big((\mathbb{F},m,l,l_b,\mathbf{W},\boldsymbol{q}_M,\boldsymbol{q}_L,\boldsymbol{q}_R,\boldsymbol{q}_C),\boldsymbol{x},(\boldsymbol{a}',\boldsymbol{b}')\big) : \mathbf{W} \in \mathcal{M} \subset \mathbb{F}^{Q\times 3m}, \\ \boldsymbol{q}_\gamma \in \mathbb{F}^m \text{ for } \gamma \in \{L,R,M,C\}, \boldsymbol{x} \in \mathbb{F}^{l-1}, \boldsymbol{a}' \in \mathbb{F}^{m-l}, \boldsymbol{b}' \in \mathbb{F}^{m-l_b} \text{ and for } \boldsymbol{a} := (1,\boldsymbol{x},\boldsymbol{a}'), \boldsymbol{b} = (\mathbf{1}_{l_b},\boldsymbol{b}') \\ \mathbf{W}\begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{b} \\ \boldsymbol{q}_M \circ \boldsymbol{a} \circ \boldsymbol{b} + \boldsymbol{q}_L \circ \boldsymbol{a} + \boldsymbol{q}_R \circ \boldsymbol{b} + \boldsymbol{q}_C \end{pmatrix} = \mathbf{0} \end{array} \right\}.$$

The relation is called $\mathcal{R}_{\mathsf{W\text{-}R1CS}}$ for weighted-R1CS. Indeed, with this arithmetization, the vector $\boldsymbol{c}$ of outputs is implicitly set to be a weighted combination of $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a} \circ \boldsymbol{b}$ and the constant 1.

Below, we show how $\mathcal{R}_{\mathsf{W\text{-}R1CS}}$ captures some existing universal relations, for different types of admissible matrices. In Appendix B, we present a PHP for $\mathcal{R}_{\mathsf{W\text{-}R1CS}}$ that is a straightforward generalization of the one in Section 4.2 and also builds on an argument for linear relations. Such an argument can be built generically from a CSS Argument (as explained in Section 4.1), as long as the argument has compatible admissible matrices. In Appendix C, we give details on different choices of CSS schemes. Finally, in Appendix D, we expand on the compilation step and the efficiency of the resulting possible zkSNARKs. Finally, in Appendix F, we present our most efficient construction, Basilisk, rolled-out.

R1CS-lite. Note that for the case where $l = l_b$, $\boldsymbol{q}_M = \mathbf{1}$, $\boldsymbol{q}_L = \boldsymbol{q}_R = \boldsymbol{q}_C = \mathbf{0}$, and $\mathbf{W} = \begin{pmatrix} \mathbf{I} & \mathbf{0} & -\mathbf{F} \\ \mathbf{0} & \mathbf{I} & -\mathbf{G} \end{pmatrix}$ for matrices $\mathbf{F}, \mathbf{G}$ in $\mathbb{F}^{m\times m}$ containing the coefficients for the linear constraints of the circuit, the relation described above corresponds to R1CS-lite. The set of admissible matrices will depend on the CSS to be used, for instance, it should be the set of matrices $\mathbf{W}$ of the form given above and such that $\mathbf{F}, \mathbf{G}$ have at most K non-zero entries when the scheme of Section 5.3 is applied. We give one instantiation of a CSS scheme for such matrices in Fig. 10.

Plonk's Constraint System. To see that the constraint system presented in Plonk [GWC19] is a particular case of $\mathcal{R}_{\mathsf{W\text{-}R1CS}}$, we first note that the former can be written the following form:

$$\mathcal{R}_{\mathsf{Plonk}} = \left\{ \begin{array}{c} (\mathsf{R},\mathsf{x},\mathsf{w}) := \big((\mathbb{F},m,l,\mathbf{P},\boldsymbol{q}_L,\boldsymbol{q}_R,\boldsymbol{q}_O,\boldsymbol{q}_M,\boldsymbol{q}_C),\boldsymbol{x},(\boldsymbol{a}',\boldsymbol{b},\boldsymbol{c})\big) : \mathbf{P} \in \mathbb{F}^{3m\times 3m} \text{ a permutation matrix}, \\ \boldsymbol{q}_\gamma \in \mathbb{F}^m \text{ for } \gamma \in \{L,R,O,M,C\}, \boldsymbol{x} \in \mathbb{F}^{l-1}, \boldsymbol{a}' \in \mathbb{F}^{m-l}, \boldsymbol{a} = (1,\boldsymbol{x},\boldsymbol{a}'), \\ (1)\ \mathbf{P}\begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{b} \\ \boldsymbol{c} \end{pmatrix} = \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{b} \\ \boldsymbol{c} \end{pmatrix}, \text{ and } (2)\ \boldsymbol{q}_L \circ \boldsymbol{a} + \boldsymbol{q}_R \circ \boldsymbol{b} + \boldsymbol{q}_O \circ \boldsymbol{c} + \boldsymbol{q}_M \circ \boldsymbol{a} \circ \boldsymbol{b} + \boldsymbol{q}_C = \mathbf{0} \end{array} \right\}.$$

The first equation is a "copy constraint" approach that takes care of consistency among wires. The second equation represents the different types of gates. When writing Circuit Satisfiability as satisfiability of this constraint system, m represents the number of additive and multiplicative gates.

This relation is a rewriting of the one considered in Plonk. Indeed, the only difference is that we require $\boldsymbol{a}$ to have the public input in the first positions, instead of forcing this in Eq.(2). Still, this is only a reformulation and does not modify the constraint system itself.

Now, we show that $\mathcal{R}_{\mathsf{W\text{-}R1CS}}$ can encode $\mathcal{R}_{\mathsf{Plonk}}$[8] when the set of admissible matrices includes $\mathbf{P} - \mathbf{I}$, where $\mathbf{P}$ is a permutation and $\mathbf{I}$ the identity matrix, and when we restrict ourselves to relations R such that $\boldsymbol{q}_O = -\mathbf{1}$. Indeed, it suffices to define $\mathbf{W} = \mathbf{P} - \mathbf{I}$, and observe that if Eq.(2) in $\mathcal{R}_{\mathsf{Plonk}}$ is satisfied, this means that

$$\boldsymbol{q}_L \circ \boldsymbol{a} + \boldsymbol{q}_R \circ \boldsymbol{b} + \boldsymbol{q}_M \circ \boldsymbol{a} \circ \boldsymbol{b} + \boldsymbol{q}_C = \boldsymbol{c}.$$

The restriction that $\boldsymbol{q}_O = -\mathbf{1}$ limits the expressiveness of the constraint system, but still captures all the constraint types described in Plonk, as we argue next.

[8] We warn the reader familiar with Plonk that, to be consistent with the notation used in our paper, we have changed the notation of Plonk. We use $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ for the witness, when in Plonk it indicates the wires that each position of the witness should be assigned to.

First, observe that given some encoding of a relation R as in $\mathcal{R}_{\mathsf{Plonk}}$ with vectors $\boldsymbol{q}'_L, \boldsymbol{q}'_R, \boldsymbol{q}'_O, \boldsymbol{q}'_M, \boldsymbol{q}'_C$, such that $(q'_O)_i \neq 0$ for all i, we can rewrite the constraints for some vectors $\boldsymbol{q}_L, \boldsymbol{q}_R, \boldsymbol{q}_O, \boldsymbol{q}_M, \boldsymbol{q}_C$ such that $\boldsymbol{q}_O = -1$ by a normalization process. On the other hand, $(q'_O)_i = 0$ in Plonk only for the case where i corresponds to public inputs or to a boolean constraint. As explained above, we enforce public input constraints separately by including them in $\boldsymbol{a}$; also, boolean constraints can be easily written enforcing $(q_O)_i = -1$: instead of requiring $a_j = b_j$, and $a_j - a_j b_j = 0$ for some b_j such that $\sigma(a_j) = b_j$ as suggested in Plonk, they can be written as $a_j = b_j = c_j$, $a_j b_j - c_j = 0$.

As mentioned before, we present a PHP for $\mathcal{R}_{\mathsf{W\text{-}R1CS}}$ that builds generically on an argument for linear relations. When the latter is an adaptation of the permutation argument of Bayer and Groth [BG12], one essentially recovers Plonk.

Alternatively, we propose to construct the argument for linear relations from one CSS arguments for matrices of the form $\mathbf{W} = \mathbf{P} - \mathbf{I}$ in Fig. 11. The approach is less efficient in terms of proof size than PLONK, but we think the additional flexibility of the CSS argument is a plus. We argue that combining this approach with the bounded fan-out approach presented next, the SRS size does not need to depend on the total number of gates (additive plus multiplicative), as it will be discussed.

Bounded fan-out. Circuits with fan-out bounded by some constant V can naturally be encoded as an instance of $\mathcal{R}_{\mathsf{W\text{-}R1CS}}$ for the set $\mathcal{M}$ of matrices $\mathbf{W} = \begin{pmatrix} \mathbf{I} & \mathbf{0} & -\mathbf{F} \\ \mathbf{0} & \mathbf{I} & -\mathbf{G} \end{pmatrix}$ with $\mathbf{F}, \mathbf{G} \in \mathbb{F}^{m \times m}$ such that there are at most V non-zero elements per column in each. As we shall see in Fig. 12, there exists a very efficient proof system for this relation, since the structure of the matrices allows to use basic CSS arguments that cannot be used in the general case.

For circuits with bounded fan-out, we can set $l = l_b$, $\boldsymbol{q}_M = \mathbf{1}$, $\boldsymbol{q}_L = \boldsymbol{q}_R = \boldsymbol{q}_C = \mathbf{0}$. This choice also gives very short specific SRS, since these vectors do not need to be computed by the indexer. We present the rolled out zkSNARK for such matrices in Appendix F.

However, we note that a more flexible choice of these values can be helpful to encode general circuits. Indeed, any circuit can be transformed to a circuit with bounded fan-out by artificially augmenting the vector $\boldsymbol{c}$ and adding constraints of the form $c_i = c_j$ that ensure consistency. To express satisfiability of this system as an instance of $\mathcal{R}_{\mathsf{W\text{-}R1CS}}$, the additional constraints $c_i = c_j$ can rewritten as an equation involving left (or right) wires, i.e. $a_i = c_j$, that is encoded in the matrix $\mathbf{W}$, and a gate, i.e. $a_i = c_i$ (setting $(q_M)_i = (q_R)_i = (q_C)_i = 0$, $(q_L)_i = 1$). If the fan-out of a certain gate is κ, this requires extending $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ by approximately $3\kappa/V$ dummy variables, and include $\boldsymbol{q}_M$ and $\boldsymbol{q}_L$ in the SRS (the rest are trivial). The construction of Appendix F can be easily modified for that case and we omit further details.

B A zkSNARK for $\mathcal{R}_{\mathsf{W\text{-}R1CS}}$

Below, we present a PHP for $\mathcal{R}_{\mathsf{W\text{-}R1CS}}$ that uses as building block a linear argument as in Section 4.1. Note that the only difference with the PHP of Fig. 2 is the set W_Y and the encoding $\mathcal{E}_Y$ that is one of the parameters of the linear argument. Given the similarity with Section 2.3, we omit the proof of Theorem 11 and refer the reader to that section.

Offline Phase: $\mathcal{I}_{\text{W-R1CS}}(\mathbb{F}, m, l, l_b, \mathbf{W}, \boldsymbol{q}_M, \boldsymbol{q}_L, \boldsymbol{q}_R, \boldsymbol{q}_C)$ parses $\mathbf{W}$ as $(\mathbf{W}_a, \mathbf{W}_b, \mathbf{W}_c)$ and runs the indexer $\mathcal{I}_{\mathsf{LA}}$ on input $(\mathbb{F}, \mathbf{W}_a, \mathbf{W}_b, \mathbf{W}_c)$ to obtain the set $\mathcal{W}_{\mathsf{LA}} = \mathcal{W}_{\mathsf{LA}a} \cup \mathcal{W}_{\mathsf{LA}b} \cup \mathcal{W}_{\mathsf{LA}c}$.
For $\gamma \in \{L, R, M, C\}$, the indexer computes polynomials $q_\gamma(X) = \boldsymbol{q}_\gamma^\top \boldsymbol{\lambda}(X)$.
Outputs $\mathcal{W}_{\text{W-R1CS}} = \mathcal{W}_{\mathsf{LA}} \cup \{q_L(X), q_R(X), q_M(X), q_C(X)\}$
Online Phase:

- $\mathcal{P}_{\text{W-R1CS}}$ Computes and outputs

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

  where $t_l(X) = \prod_{j=1}^{l}(X - \mathsf{h}_j)$, $t_{l_b}(X) = \prod_{j=1}^{l_b}(X - \mathsf{h}_j)$.
- $\mathcal{P}_{\text{W-R1CS}}$ and $\mathcal{V}_{\text{W-R1CS}}$ instantiate $\mathcal{P}_{\mathsf{LA}}(\mathbb{F}, \mathbf{W}, (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{q}_M \circ \boldsymbol{a} \circ \boldsymbol{b} + \boldsymbol{q}_L \circ \boldsymbol{a} + \boldsymbol{q}_R \circ \boldsymbol{b} + \boldsymbol{q}_C))$ and $\mathcal{V}_{\mathsf{LA}}^{\mathcal{W}_{\mathsf{LA}}}(\mathbb{F})$. Let $\boldsymbol{Y}(X) = (A(X), B(X), q_M(X)A(X)B(X) + q_L(X)A(X) + q_R(X)B(X) + q_C(X))$ be the polynomials $\mathcal{P}_{\mathsf{LA}}$ outputs in the first round.

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

Fig. 9. PHP for the universal relation $\mathcal{R}_{\text{W-R1CS}}$. The set of admissible matrices must coincide with the set of admissible matrices of the linear argument, which in fact depends on the admissible matrices of the CSS argument. The PHP for $\mathcal{R}_{\mathsf{LA}}$ should be instantiated for $W_Y = \{(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{q}_M \circ \boldsymbol{a} \circ \boldsymbol{b} + \boldsymbol{q}_L \circ \boldsymbol{a} + \boldsymbol{q}_R \circ \boldsymbol{b} + \boldsymbol{q}_C) : \boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^m\}$, $\mathcal{E}(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{q}_M \circ \boldsymbol{a} \circ \boldsymbol{b} + \boldsymbol{q}_L \circ \boldsymbol{a} + \boldsymbol{q}_R \circ \boldsymbol{b} + \boldsymbol{q}_C) = (\boldsymbol{a}^\top \boldsymbol{\lambda}(X), \boldsymbol{b}^\top \boldsymbol{\lambda}(X), (\boldsymbol{q}_M^\top \boldsymbol{\lambda}(X))(\boldsymbol{a}^\top \boldsymbol{\lambda}(X))(\boldsymbol{b}^\top \boldsymbol{\lambda}(X)) + (\boldsymbol{q}_L^\top \boldsymbol{\lambda}(X))(\boldsymbol{a}^\top \boldsymbol{\lambda}(X)) + (\boldsymbol{q}_R^\top \boldsymbol{\lambda}(X))(\boldsymbol{b}^\top \boldsymbol{\lambda}(X)) + (\boldsymbol{q}_C^\top \boldsymbol{\lambda}(X))$.

Theorem 11. *When instantiated with a complete, sound and knowledge soundness linear argument, the PHP of Fig. 9 satisfies completeness, knowledge and knowledge-soundness.*

Adding Zero Knowledge To add zero-knowledge to the PHP of Fig. 9 we can proceed as explained in Section 4.3. For $i = m + 1, \ldots, \tilde{m}$, let $(\boldsymbol{q}_M)_i = 1$, $(\boldsymbol{q}_L)_i = (\boldsymbol{q}_R)_i = (\boldsymbol{q}_C)_i = 0$, so $c_i = a_i b_i$ for these indexes, just as before. Choosing $\boldsymbol{r}_a, \boldsymbol{r}_b$ subject to $\sum r_{a,i} + r_{b,i} = 0$ and modifying the matrix of constraints as explained there, we obtain a PHP with zero-knowledge.

C Instantiations of CSS Arguments

In this section we present several instantiations of CSS arguments for different sets of admissible matrices. Using them as a starting point, we can construct linear arguments that can be used as a building block in the PHP of Fig. 9 for different specific families of weighted R1CS relations. The final goal is to study the efficiency trade-offs that result from the different approaches. We start by describing the general approach and some general techniques that allow for better efficiency. We then describe the particular cases separately, presenting a full description of each scheme.

A fundamental observation is that matrix $\mathbf{W}$ can be seen as a matrix with three blocks $(\mathbf{W}_a, \mathbf{W}_b, \mathbf{W}_c)$ and sampling in each of these blocks must be done separately, as the prover needs to receive $(D_a(X), D_b(X), D_c(X))$ to do the inner product with $(A(X), B(X), C(X))$, where $C(X) = q_M(X)A(X)B(X) + q_L(X)A(X) + q_R(X)B(X) + q_C(X)$. Naively, the polynomials $D_a(X), D_b(X)$, and $D_c(X)$ are obtained by running *one* CSS scheme for each matrix $\mathbf{W}_a, \mathbf{W}_b$, and $\mathbf{W}_c$, but more careful approaches can save elements in communication complexity.

Before we flesh out the different options of CSS arguments for $\mathbf{W}$ we note some general principles to improve efficiency and possible trade-offs:

a) Each of the column blocks $\mathbf{W}_a, \mathbf{W}_b, \mathbf{W}_c$ is a matrix of Q rows, where $Q = 2m$ or $Q = 3m$. One possibility is to use one of the CSS arguments described in Sections 5.2,5.3,5.4,5.5 directly for each one of these matrices of Q rows (assuming they belong to the set of admisible matrices). Another possibility is to cut each $\mathbf{W}_a, \mathbf{W}_b, \mathbf{W}_c$ into blocks of m rows. For instance, this is the approach explained before for R1CS-lite, where $\mathbf{W}_c = \begin{pmatrix} \mathbf{F} \\ \mathbf{G} \end{pmatrix}$ and, assuming $|\mathbf{F}| + |\mathbf{G}| \leq K$, we implicitly use the CSS argument of Section 5.3 for the matrix $\mathbf{F} + z\mathbf{G} \in \mathbb{F}^{m \times m}$, where z is an element chosen by the verifier. Technically, this is in fact a CSS argument for the matrix $\mathbf{W}_c$ where the sampling coefficients depends also on z.

b) When a block of size $m \times m$ is trivial, that is, either $\mathbf{0}$ or $\mathbf{I}$, the corresponding $D(X)$ is either zero or can be opened by the verifier. Indeed, when the block is $\mathbf{0}$ so is the resulting polynomial, and when it is $\mathbf{I} \in \mathbb{F}^{m \times m}$, we define $P(X, Y) = \boldsymbol{\lambda}(Y)^\top \mathbf{I} \lambda(X)^\top = \lambda(Y)^\top \lambda(X)$, and the value $P(y, x) = \big(t(x)y - xt(y)\big)/(x - y)$ can be calculated by the verifier with $O(\log m)$ field operations (a proof of this can be found, for example, in Lemma 3 of Lunar [CFF+20]). Thus, when $\mathbf{W}_a, \mathbf{W}_b$ consist of trivial blocks of size $m \times m$, as in R1CS-lite, it makes sense to use the approach described in a) and cut these matrices in blocks of m rows. The verifier can then open $D_a(X), D_b(X)$ itself (as they are linear combination of the polynomials corresponding to trivial blocks), so there is no need to use a CSS argument to prove correct sampling.

c) The proofs that $D_a(X), D_b(X), D_c(X)$ are correctly sampled (in case neither of the matrices has a simple form and none of these polynomials can be sampled by the verifier) can be aggregated.

Sparse Matrix Approach: This is the approach followed by Marlin and Lunar, and also explored in Section 5. The main idea in this line of work is to prove correct sampling in the row space of some matrix $\mathbf{M}$ with no particular structure except for being sparse. The number of non-zero entries is at most K, which is the size of some multiplicative group $\mathbb{K}$ of $\mathbb{F}$. This approach was introduced in Marlin, and is pursued in Lunar and also in Section 5.3 of this work. It can be generalized to sums of sparse matrices without increasing the communication complexity, as explained in Section 5.5.

Below, we give the full details of the CSS argument for R1CS-lite tweaking the techniques introduced in Section 5.3, for the case where a matrix in $\mathbb{F}^{Q \times m}$ is split into blocks of m rows. Given that the encoding of $\mathbf{W}_a$ and $\mathbf{W}_b$ can be opened and checked by the verifier, we only need to run a CSS argument for $\mathbf{W}_c$. We assume two multiplicative subgroups, $\mathbb{H}$ with size at least m and $\mathbb{K}$ with size $K \geq |\mathbf{F}| + |\mathbf{G}|$, and define K_1, K_2 such that $|\mathbf{F}| \leq K_1, |\mathbf{G}| \leq K_2$ and $K = K_1 + K_2$. Technically, we construct a CSS argument for the matrix $\mathbf{W}_c$ where the coefficients depend on x, z. As mentioned before, this corresponds to implicitly applying the results in Section 5.3 to a matrix $\hat{\mathbf{W}}_{\mathbf{c}} = \mathbf{W}_c^1 + z\mathbf{W}_c^2$ that depends on the verifier's challenge z.

Let $\mathsf{v} : \mathbb{K} \to \mathbb{F}$ be the function that maps an element $\mathsf{k}_\ell \in \mathbb{K}$ to the value of the ℓth non-zero element of matrix $\mathbf{F}$, if $\ell \leq K_1$, and to the value of the $(\ell - K_1)$th element of $\mathbf{G}$ if $\ell > K_1$. Define also $\mathsf{c}, \mathsf{r} : \mathbb{K} \to [m]$ as the functions that output its row and column position in the corresponding matrix, we define $P_1(X, Y) = \sum_{\ell=1}^{K_1} \mathsf{v}(\mathsf{k}_\ell) \lambda_{\mathsf{r}(\mathsf{k}_\ell)}(Y) \lambda_{\mathsf{c}(\mathsf{k}_\ell)}(X)$ the sparse encoding of $\mathbf{F}$ and $P_2(X, Y) = \sum_{\ell=K_1+1}^{K_2} \mathsf{v}(\mathsf{k}_\ell) \lambda_{\mathsf{r}(\mathsf{k}_\ell)}(Y) \lambda_{\mathsf{c}(\mathsf{k}_\ell)}(X)$ the sparse encoding of $\mathbf{G}$. Our argument implicitly constructs the sparse encoding of $\mathbf{F} + z\mathbf{G}$ as $P(X, Y) = P_1(X, Y) + zP_2(X, Y)$. As explained in Section 5.1, $P(y, x)$ 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 argument of Section 5.2 can be used to prove correctness. We present this scheme in Fig. 10.

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

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

$$v^1_{2,\mathsf{c}}(X) = m^{-1} \sum_{\ell=1}^{K_1} \mathsf{v}(\mathsf{k}_\ell)\mathsf{h}_{\mathsf{c}(\mathsf{k}_\ell)} \mu_\ell(X), \qquad v^2_{2,\mathsf{c}}(X) = m^{-1} \sum_{\ell=K_1+1}^{K} \mathsf{v}(\mathsf{k}_\ell)\mathsf{h}_{\mathsf{c}(\mathsf{k}_\ell)} \mu_\ell(X).$$

Online Phase: $\mathsf{Sampling}$: $\mathcal{V}_{\mathsf{CSS}}$ sends $x, z_1 \leftarrow \mathbb{F}$, and $\mathcal{P}$ outputs $D(X) = P(X, x)$, for $P(X, Y) = \sum_{\ell=1}^{K_1} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{r}(\mathsf{k}_\ell)}(Y)\lambda_{\mathsf{c}(\mathsf{k}_\ell)}(X) + z_1 \sum_{\ell=K_1+1}^{K} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{r}(\mathsf{k}_\ell)}(Y)\lambda_{\mathsf{c}(\mathsf{k}_\ell)}(X)$.
$\mathsf{ProveSampling}$: $\mathcal{V}_{\mathsf{CSS}}$ sends $y \leftarrow \mathbb{F}$ and $\mathcal{P}_{\mathsf{CSS}}$ outputs $\sigma = D(y)$ and $e_x(X), e_y(X)$, where $e_x(X) = \sum_{\ell=1}^{K} \lambda_{\mathsf{r}(\mathsf{k}_\ell)}(x)\mu_\ell(X)$, $e_y(X) = \sum_{\ell=1}^{K_1} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{c}(\mathsf{k}_\ell)}(y)\mu_\ell(X) + z_1 \sum_{\ell=K_1+1}^{K} \mathsf{v}(\mathsf{k}_\ell)\lambda_{\mathsf{c}(\mathsf{k}_\ell)}(y)\mu_\ell(X)$,
$\mathcal{V}_{\mathsf{CSS}}$ sends $z_2 \leftarrow \mathbb{F}$ and $\mathcal{P}_{\mathsf{CSS}}$ computes $H_{u,x}(X)$, $H_{u,y}(X)$, $R_u(X)$, $H_{u,x,y}(X)$ such that:

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

$$e_y(X)(y - v_{1,\mathsf{c}}(X)) = t(y)(v^1_{2,\mathsf{c}}(X) + z_1 v^1_{2,\mathsf{c}}(X)) + H_{u,y}(X)u(X)$$

$$Ke_x(X)e_y(X) - \sigma = XR_u(X) + u(X)H_{u,x,y}(X),$$

It also defines $H_u(X) = H_{u,x,y}(X) + z_2 H_{u,x}(X) + z_2^2 H_{u,y}(X)$, and outputs $\big(R_u(X), H_u(X)\big)$.
Decision Phase: Accept if and only if (1) $\deg(R_u) \leq K - 2$, (2) $D(y) = \sigma$, and (3) for $i_x(X) = (x - v_{\mathsf{r}}(X))$, $i_y(X) = (y - v_{1,\mathsf{c}}(X))$

$$\begin{aligned}(e_x(X) + z_2^2 i_y(X))(e_y(X) + z_2 i_x(X)) - z_2^3 i_x(X) i_y(X) \\ - z_2^2 t(y)(v^1_{2,\mathsf{c}}(X) + z_1 v^2_{2,\mathsf{c}}(X)) - \sigma/K - z_2 t(x) m^{-1} v_{\mathsf{r}}(X) = XR_u(X) + H_u(X)u(X).\end{aligned}$$

Fig. 10. CSS Argument for $\mathbf{W}_c = \begin{pmatrix}\mathbf{F}\\ \mathbf{G}\end{pmatrix}$, a matrix with at most K non-zero entries.

Permutation Matrix Approach: As explained before, in order to instantiate $\mathcal{R}_{\mathsf{W\text{-}R1CS}}$ following Plonk, we consider matrices of the form $\mathbf{W} = \mathbf{P} - \mathbf{I}$, where $\mathbf{P}$ is a matrix of permutations in $\mathbb{F}^{3n \times 3n}$. For simplifying notation, we define the mapping $\iota : \{1, 2, 3\} \to \{a, b, c\}$ as $\iota(1) = a$, $\iota(2) = b$ and $\iota(3) = c$.

There are several possible ways of proving correct sampling in the rows of $\mathbf{P} - \mathbf{I}$. For instance, we could consider $\mathbf{P}$ as a matrix of three column blocks $\mathbf{P}_a, \mathbf{P}_b$ and $\mathbf{P}_c$, and define the polynomial encoding of each block as $\boldsymbol{\mu}(Y)^\top \mathbf{P}_\gamma \boldsymbol{\lambda}(X)$, where $\boldsymbol{\mu}(X)^\top = (\mu_1(X), \ldots, \mu_{3m}(X))$ are Lagrange interpolation polynomials associated a multiplicative subgroup of size at least $3m$. However, the simplest and most efficient one, splits this matrix into 9 blocks $m \times m$. Since all the blocks of m rows of $\mathbf{I} \in \mathbb{F}^{3m \times 3m}$ are either $\mathbf{0}$ or $\mathbf{I}$, the verifier can open the polynomial associated to $\mathbf{I} \in \mathbb{F}^{3m \times 3m}$ on its own, and a CSS argument is necessary only to sample in the rows of $\mathbf{P}$.

For this approach, parse $\mathbf{P}$ as $(\mathbf{P}_a, \mathbf{P}_b, \mathbf{P}_c)$ and, for $i = 1, 2, 3$, each $\mathbf{P}_{\iota(i)}$ as three matrices $\mathbb{F}^{m \times m}$ corresponding to blocks of m rows and denoted as $\mathbf{P}^1_{\iota(i)}, \mathbf{P}^2_{\iota(i)}$, and $\mathbf{P}^3_{\iota(i)}$. For $i = 1, 2, 3$, define the function $\mathsf{r}_i : \mathbb{H} \to [3m]$ that, given an element $\mathsf{h}_\ell \in \mathbb{H}$ outputs the row corresponding to the only non-zero element in column ℓ of matrix $\mathbf{P}_{\iota(i)}$. For $k = 1, 2, 3$, the polynomial encoding of $\mathbf{P}^k_{\iota(i)}$ is $P^k_{\iota(i)}(X, Y) = \boldsymbol{\lambda}(Y)^\top \mathbf{P}^k_{\iota(i)} \boldsymbol{\lambda}(X) = \sum_{\ell:(k-1)m+1 \leq \mathsf{r}_i(\mathsf{h}_\ell) \leq km} \lambda_{\mathsf{r}_i(\mathsf{h}_\ell)-(k-1)m}(Y)\lambda_\ell(X)$. The polynomial $D_{\iota(i)}(X)$ is $P_{\iota(i)}(X, x) = P^1_{\iota(i)}(X, x) + zP^2_{\iota(i)}(X, x) + z^2 P^3_{\iota(i)}(X, x)$. The two key elements for efficiency are: 1) the observation that each column block $\mathbf{P}_{\iota(i)}$ is a simple matrix, since it has at most one non-zero element per columnn, and 2) the fact that the proofs for each of these blocks can be batched together.

Offline Phase: $\mathcal{I}_{\mathsf{CSS}}(\mathbb{F}, \mathbf{M})$: For $i = 1, 2, 3$, $k = 1, 2, 3$ define $\mathcal{V}^{i,k} = \{\ell \in [m] : (k-1)m + 1 \leq \mathsf{r}_i(\mathsf{h}_\ell) \leq km\}$, and

$$v^{i,k}(X) = \sum_{\ell \in \mathcal{V}^{i,k}} \mathsf{h}_{\mathsf{r}_i(\mathsf{h}_\ell)-(i-1)m} \lambda_\ell(X).$$

Output $\mathcal{W}_{\mathsf{CSS}} = \left\{\{v^{i,k}(X)\}_{i,k=1}^{3}\right\}$.

Online Phase: Sampling: $\mathcal{V}_{\mathsf{CSS}}$ outputs $x, z \leftarrow \mathbb{F}$ and $\mathcal{P}_{\mathsf{CSS}}$ computes and outputs $D_{\iota(i)}(X) = P_{\iota(i)}(X, x)$ for $i = 1, 2, 3$.

ProveSampling: $\mathcal{V}_{\mathsf{CSS}}$ outputs δ. For $i = 1, 2, 3$, $v^i(X) = \sum_{k=1}^{3} v^{i,k}(X)$ and $v_z^i(X) = \sum_{k=1}^{3} z^{k-1} v^{i,k}(X)$, the prover $\mathcal{P}_{\mathsf{CSS}}$ finds and outputs $H_t(X)$ such that

$$D_a(X)(x - v^1(X)) + \delta D_b(X)(x - v^2(X)) + \delta^2 D_c(X)(x - v^3(X)) = t(x)(v_z^1(X) + \delta v_z^2(X) + \delta^2 v_z^3(X)) + H_t(X)t(X).$$

Decision Phase: Accept if and only if (1) $\deg(D_{\iota(i)}) \leq m - 1$, for $i = 1, 2, 3$ and (2)

$$D_a(X)(x - v^1(X)) + \delta D_b(X)(x - v^2(X)) + \delta^2 D_c(X)(x - v^3(X)) = t(x)(v_z^1(X) + \delta v_z^2(X) + \delta^2 v_z^3(X)) + H_t(X)t(X).$$

Fig. 11. CSS Argument for $\mathbf{P} \in \mathbb{F}^{3m \times 3m}$.

Bounded Fan-out: Finally, we consider the case of circuits with bounded fan-out, that is, the case where the circuit can be represented with a matrix $\mathbf{W} = \begin{pmatrix} \mathbf{I} & \mathbf{0} & -\mathbf{F} \\ \mathbf{0} & \mathbf{I} & -\mathbf{G} \end{pmatrix}$ that is a sum of at most V simple matrices, i.e. it has at most V non-zero elements per column.

As before, and since the other blocks of m rows are the identity matrix or the zero matrix, it suffices to use a CSS argument to sample in the image of $\mathbf{W}_c = -\begin{pmatrix} \mathbf{F} \\ \mathbf{G} \end{pmatrix}$. For that, we first write the matrix $\mathbf{W}_c = \sum_{i=1}^{V} \begin{pmatrix} \mathbf{F}_i \\ \mathbf{G}_i \end{pmatrix}$, where each $\begin{pmatrix} \mathbf{F}_i \\ \mathbf{G}_i \end{pmatrix}$ is a simple matrix. Once more, we will implicitly construct the scheme for $\hat{\mathbf{W}} = \mathbf{F} + z\mathbf{G}$, that can be written as $\sum_{i=1}^{V} \mathbf{F}_i + z\mathbf{G}_i$, with each $\mathbf{F}_i + z\mathbf{G}_i$ having at most one non-zero element in each column. We define two functions associated to each $\mathbf{W}_{c,i} = \begin{pmatrix} \mathbf{F}_i \\ \mathbf{G}_i \end{pmatrix}$. The function $\mathsf{r}_i : \mathbb{H} \to [2m]$ that, given an element $\mathsf{h}_\ell \in \mathbb{H}$ outputs the row corresponding to the only non-zero element in column ℓ of matrix $\mathbf{W}_{c,i}$ and the function $\mathsf{v}_i : \mathbb{H} \to \mathbb{F}$ that outputs the value of this non-zero entry. The details of the scheme are given in Fig. 12 and for simplicity in the notation, we define the sets $\mathcal{V}_\ell^1 = \{i \in [V] : 1 \leq \mathsf{r}_i(\mathsf{h}_\ell) \leq m\}$, $\mathcal{V}_\ell^2 = \{i \in [V] : m + 1 \leq \mathsf{r}_i(\mathsf{h}_\ell) \leq 2m\}$, $S_\ell = \left\{\{\mathsf{r}_i(\mathsf{h}_\ell) : i \in \mathcal{V}_\ell^1\} \cup \{\mathsf{r}_i(\mathsf{h}_\ell) - m : i \in \mathcal{V}_\ell^2\}\right\}$ and $\hat{\mathcal{V}}_i^1 = \{\ell \in [m] : 1 \leq \mathsf{r}_i(\mathsf{h}_\ell) \leq m\}$, $\hat{\mathcal{V}}_i^2 = \{\ell \in [m] : m + 1 \leq \mathsf{r}_i(\mathsf{h}_\ell) \leq 2m\}$.

Mixing the Bounded Fan-out and the Permutation Approach. Bayer and Groth [BG12] introduce techniques to prove that a vector is a permutation of another one. This approach is useful for many applications, but for the ones discussed in this section it has the drawback that it is not easy to extend it to sums of permutations without increasing the communication complexity. This is exactly the issue in the fully succinct mode of Sonic, where complexity grows with the number of permutation matrices into which the constraint matrix can be decomposed.

To counter this issue, Plonk proposes to define the permutation $\mathbf{P} \in \mathbb{F}^{3m \times 3m}$. As mentioned before, the idea is to create a vector of copy constraints. With this approach, the fan-out can be unlimited. Values that are repeated are encoded as a cycle of the permutation and the price to pay is that additive gates are no longer for free.

It is worth investigating if these ideas can be mixed. Namely, since in our case we can increase the fan-out to V without paying in terms of proof size, one could follow the copy constraint approach only for the wires

exceeding the fan-out bound. The result would be that additive gates involving only output wires that are input of less than V multiplication gates would be for free.

Offline Phase: $\mathcal{I}_{\mathsf{CSS}}(\mathbb{F}, \mathbf{M})$: Define the polynomials $\hat{R}^1_\ell(Y), \hat{R}^2_\ell(Y), \hat{I}_\ell(Y)$, and its coefficients $\hat{R}^1_{\ell j}, \hat{R}^2_{\ell j}, \hat{I}_{\ell j}$:

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

$$\hat{R}^2_\ell(Y) = \frac{1}{m} \sum_{i \in \mathcal{V}^2_\ell} \mathsf{v}_i(\mathsf{h}_\ell) \mathsf{h}_{\mathsf{r}_i(\mathsf{h}_\ell)-m} \prod_{s \in S_\ell - \{\mathsf{r}_i(\mathsf{h}_\ell)-m\}} (Y - \mathsf{h}_s) = \sum_{j=0}^{V-1} \hat{R}^2_{\ell j} Y^j,$$

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

Define

$$v^{\hat{R},1}_j(X) = \sum_{\ell=1}^{m} \hat{R}^1_{\ell j} \lambda_\ell(X), \qquad v^{\hat{R},2}_j(X) = \sum_{\ell=1}^{m} \hat{R}^2_{\ell j} \lambda_\ell(X).$$

$$v^{\hat{I}}_j(X) = \sum_{\ell=1}^{m} \hat{I}_{\ell j} \lambda_\ell(X).$$

Output $\mathcal{W}_{\mathsf{CSS}} = \left\{ \{v^{\hat{I}}_j(X)\}_{j=0}^{V}, \{v^{\hat{R},1}_j(X), v^{\hat{R},2}_j(X)\}_{j=0}^{V-1} \right\}$.

Online Phase: Sampling: $\mathcal{V}_{\mathsf{CSS}}$ outputs $x, z \leftarrow \mathbb{F}$ and $\mathcal{P}_{\mathsf{CSS}}$ computes $D(X) = P(X, x)$, for $P(X, Y) = \sum_{i=1}^{V} \left(\sum_{\ell \in \hat{\mathcal{V}}^1_i} \mathsf{v}_i(\mathsf{h}_\ell) \lambda_{\mathsf{r}_i(\mathsf{h}_\ell)}(Y) \lambda_\ell(X) \right) + z \left(\sum_{\ell \in \hat{\mathcal{V}}^2_i} \mathsf{v}_i(\mathsf{h}_\ell) \lambda_{\mathsf{r}_i(\mathsf{h}_\ell)-m}(Y) \lambda_\ell(X) \right)$.

ProveSampling: $\mathcal{P}_{\mathsf{CSS}}$ finds and outputs $H_t(X)$ such that, if $\hat{R}_x(X) = \sum_{j=0}^{V-1} x^j \left(v^{\hat{R},1}_j(X) + z v^{\hat{R},2}_j(X) \right)$, and $\hat{I}_x(X) = \sum_{j=0}^{V} x^j v^{\hat{I}}_j(X)$,

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

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

Fig. 12. CSS Argument for $\mathbf{W}_c = \begin{pmatrix} -\mathbf{F} \\ -\mathbf{G} \end{pmatrix}$, where $\mathbf{F}, \mathbf{G}$ have at most V non-zero elements per column. Bounded fan-out circuits can be naturally encoded in this way.

D zkSNARKs from CSS arguments

Even though there are several differences in the CSS schemes presented along this work, all our constructions are compiled in a similar manner, following the compiler in [CFF+20].

The universal SRS of the zkSNARK will be $\mathsf{srs}_\mathsf{u} = \left(\{[\tau^i]_1\}_{i=1}^{\rho}, [\tau]_2 \right)$, where ρ is the maximum degree among all polynomials in $\mathcal{W}_{\mathsf{CSS}}$ or sent by the prover. $\mathsf{srs}_\mathbf{W}$ consists of the evaluation in τ of the polynomials that $\mathcal{I}_{\mathsf{LA}}$ outputs, thus, $|\mathsf{srs}_\mathbf{W}| = |\mathcal{W}_{\mathsf{CSS}}| + 4$, due to polynomials $q_L(X), q_r(X), q_M(X)$ and $q_c(X)$. Still, in all schemes but the one of Fig. 11 these polynomials are zero and then the size of $\mathsf{srs}_\mathbf{W}$ is the size of $\mathcal{W}_{\mathsf{CSS}}$.

Prover and Verifier instantiate $\mathcal{P}_{\mathsf{W\text{-}R1CS}}$ and $\mathcal{V}_{\mathsf{W\text{-}R1CS}}$ for the PHP of Fig. 9 that achieves zero-knowledge through the changes presented in Fig. 3, as presented below.

All oracle polynomials sent by $\mathcal{P}_{\mathsf{W\text{-}R1CS}}$ are translated into polynomials evaluated (in the source group) at τ. For degree checks with $\deg(p) < \mathsf{dg}$, $\mathsf{dg} < \rho$, the prover sends a single extra polynomial and field element (see Section 2.4), while checks for $\mathsf{dg} = \rho$ 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 [GWC19] 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 two pairings, which sets proof size (in terms of group elements) as the amount of oracles sent by $\mathcal{P}_{\text{W-R1CS}}$ plus one element for degree check of $R_t(X)$ in the linear argument and one for each polynomial equation. The number of field elements sent by the prover changes depending on the amount of terms included in the final polynomial equation as explained above, but always include one element for each polynomial commitment opening.

Prover's work includes running $\mathcal{P}_{\text{W-R1CS}}$ as well as the computation of the polynomial commitment opening procedures. Verifier work is also $\mathcal{V}_{\text{W-R1CS}}$ 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 argument, but concrete efficiency depends on the specifics of it and also how the blocks of rows of $\mathbf{W}$ are combined. 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 (see Section 4.4).

For the sparse matrix 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 z, as shown in Fig.10. This set the size of the universal srs to $K - 1$ and the size of $\mathsf{srs}_{\mathbf{W}}$ to 4. Prover sends 11 polynomials, 8 of degree up to $m - 1$ and the rest $(e_x(X), e_y(X), R_u(X), H_u(X))$ of degree $K - 1$. Verifier performs two pairings and the field operations to compute $D_a(y), D_b(y), t_l(y), u(y)$, and $t(y)$.

The zkSNARK that uses the CSS argument of Fig. 11 has a universal SRS of size $n + 5$, for n the *total* number of gates of the circuit, i.e., multiplicative and additive gates as well, while the relation dependent has 11 group elements. The proof has 9 group elements (this time the prover has to send $D_a(X)$ and $D_b(X)$) and 5 field elements (as it also sends its evaluations on challenge y). Prover work depends only on these polynomials and consists of $9m$ group operations. Similar to other constructions, verifier work includes 2 pairings and $O(l + \log m)$ field operations.

Finally, the zkSNARK that builds on the CSS scheme of Fig. 12 (*Basilisk*) is given in Fig. 13: the universal SRS has size $m + 6$, for m the number of multiplicative gates of the circuit, and the one describing the relation includes $3V + 1$ group elements. The proof includes 7 group and 3 field elements. Prover work is dominated by the generation of these polynomials, and consists of $7m$ group operations. Verifier performs $O(l + \log m)$ operations to compute $\sum_{i=1}^{l} x_i \lambda_i(y)$ from the public input $\boldsymbol{x}$ and challenge y and to evaluate $t(x), t(y), t_l(y), D_a, D_b$ from its challenges x and y. It also does two pairings to check the final equation. As explained in Appendix B, to generalize the construction to arbitrary circuits we can add dummy variables (the exact number depends on the number of gates that exceed the fan-out bound). The SRS will grow accordingly, and the derived SRS needs to include two additional polynomials.

If the extended Vandermonde technique of (Fig. 7) is used for some set of Q rows, we set $\rho = 2m - 1$. $\mathsf{srs}_{\mathbf{W}}$ outputs $Q + \ell$ vectors of three polynomials. Prover only sends commitments to $A'(X)$ and $B'(X)$ and the polynomials $R_t(X), H_t(X)$ of the linear argument (Fig.1). Verifier checks degree of $R_t(X)$ and one polynomial equation of three terms, two of which include polynomials it can evaluate itself (X and $t(X)$).

E Eliminating Non-Trivial Degree Checks

As explained in Apendix D, checking that a polynomial is of degree at most $m-1$ is for free, as the srs includes only powers of τ up to this bound. On the other hand, checks for smaller degrees require the prover to send two extra elements, one in the field and one in the group. In all our constructions such degree check is required for the linear argument, since Theorem 2 states that the degree of polynomial $R(X)$ has to be at most $m-2$. Also, in the CSS of Fig. 5, it must be the case that $\deg(R_u) \leq m-2$.

Below, we present a simple corollary of Theorem 2 to augment the degree of $R(X)$ by 1. This trick allows to save the aforementioned elements in the proof.

Corollary 1. *Let $k, m, \boldsymbol{y}, \boldsymbol{d}, \mathbb{F}, \mathbb{H}$ be as in Theorem 2 and let $u \in \mathbb{F}^*$, $u \notin \mathbb{H}$. Then, $\boldsymbol{y} \cdot \boldsymbol{d} = \sigma$ if and only if there exist $H(X), R(X) \in \mathbb{F}[X]$, $R(X)$ of degree at most $m-1$, such that the following relation holds:*

$$\boldsymbol{Y}(X) \cdot \boldsymbol{D}(X)(X-u) - \frac{\sigma}{m}(X-u) = XR(X) + t(X)H(X)(X-u), \tag{5}$$

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

Proof. By Theorem 2, $\boldsymbol{y} \cdot \boldsymbol{d} = \sigma$ if and only if there exists some $R'(X)$ of degree at most $m-2$ such that $\boldsymbol{Y}(X) \cdot \boldsymbol{D}(X) - \frac{\sigma}{m} = XR'(X) + t(X)H(X)$.

If $\boldsymbol{y} \cdot \boldsymbol{d} = \sigma$ then such $R'(X)$ exists and the polynomial $R(X) = (X-u)R'(X)$ is of degree at most $m-1$ and satisfies Eq. (5).

We now prove the reciprocal. Suppose some $R(X)$ of degree at most $m-1$ exists such that Eq. (5) holds. Since all the sum terms in the equation, except for $XR(X)$ are divisible by $X-u$, and $u \neq 0$, then $(X-u)$ divides $R(X)$. Define $R'(X) = R(X)/(X-u)$. Dividing Eq. (5) by $X-u$, it follows that $R'(X)$ satisfies Eq. (2) and is of degree at most $m-2$, so by Theorem 2 it follows that $\boldsymbol{y} \cdot \boldsymbol{d} = \sigma$. □

F Rolled-out zkSNARK for Circuits with Bounded Fan-Out

Below we present the final zkSNARK protocol for proving some relation $\mathsf{R} \in \mathcal{R}_{\text{W-R1CS}}$ that represents a circuit with bounded fan-out is satisfied, i.e., sets $\mathbf{W} = \begin{pmatrix} \mathbf{I} & \mathbf{0} & -\mathbf{F} \\ \mathbf{0} & \mathbf{I} & -\mathbf{G} \end{pmatrix}$, where $\mathbf{W}_c = \begin{pmatrix} -\mathbf{F} \\ -\mathbf{G} \end{pmatrix}$ has at most V non-zero elements per column. Our choice is due to the fact that this CSS scheme is the most efficient among all the presented ones. Also, circuits can be transformed into this form by adding additional dummy constraints. We present the scheme for the case where $l = l_b$, $\boldsymbol{q}_M = \mathbf{1}$, $\boldsymbol{q}_L = \boldsymbol{q}_R = \boldsymbol{q}_C = \mathbf{0}$ but it is straightforward to modify the argument for other values. In blue we highlight the modifications to the PHP of Fig. 9 in order to make it zero knowledge. The functions $\{\mathsf{v}_i, \mathsf{r}_i\}_{i=1}^V$, $P(X,Y)$, and the sets $\{S_\ell, \mathcal{V}_\ell^1, \mathcal{V}_\ell^2\}_{\ell=1}^m$ are defined as above. If $\iota(1) = a, \iota(2) = b, \iota(3) = c$, and for $i = 1, 2$, $k = 1, 2$, let $(P')_{\iota(i)}^k(X,Y) = \boldsymbol{\lambda}(Y)^\top(\mathbf{W}_{\iota(i)}^1 + z_1 \mathbf{W}_{\iota(i)}^2)\boldsymbol{\lambda}(X)$, and $(D')_{\iota(i)}^k(X) = (P')_{\iota(i)}^k(X,x)$. Note that we eliminate the non-trivial degree check as in App. E.

$\mathsf{KeyGen}(\mathcal{R})$: Sample $\tau \leftarrow \mathbb{F}$ and output $\tau, \mathsf{srs}_{\mathsf{u}} = \big(\{[\tau^i]_1\}_{i=0}^{m-1}, \{[\tau^i]_1\}_{i=m}^{m+5}, [\tau]_2\big)$. Choose an arbitrary $u \in \mathbb{F}^*$, $u \notin \mathbb{H}$.

$\mathsf{KeyGenD}(\mathsf{srs}_{\mathsf{u}}, \mathbf{W}', \boldsymbol{w}')$: Parse $\mathbf{W}' = \big(\mathbf{W}'_a, \mathbf{W}'_b, \mathbf{W}'_c\big)$ and $\mathbf{W}'_c$ as $\mathbf{W}'_c = \begin{pmatrix} \mathbf{F} & \mathbf{0}_{m\times 6} \\ \mathbf{G} & \mathbf{0}_{m\times 6} \end{pmatrix}$, $\mathbf{F}, \mathbf{G} \in \mathbb{F}^{m\times m}$. For $i \in [V], k = 1, 2$ define $\hat{R}^k_\ell(Y)$, and its coefficients $\hat{R}^k_{\ell j}$ as:

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

Also, let $\hat{I}_\ell(Y)$ and $\hat{I}_{\ell j}$ be such that $\hat{I}_\ell(Y) = \prod_{s\in S_\ell}(Y - \mathsf{h}_s) = \sum_{j=0}^{V} \hat{I}_{\ell j} Y^j$.

Finally, for $j = 0, \ldots, V-1$ define $v_j^{\hat{R},1}(X) = \sum_{\ell=1}^m \hat{R}^1_{\ell j}\lambda_\ell(X)$, $v_j^{\hat{R},2}(X) = \sum_{\ell=1}^m \hat{R}^2_{\ell j}\lambda_\ell(X)$, and, for $j = 0, \ldots, V$ $v_j^{\hat{I}}(X) = \sum_{\ell=1}^m \hat{I}_{\ell j}\lambda_\ell(X)$. Compute $[v_j^{\hat{I}}]_1 = [v_j^{\hat{I}}(\tau)]_1, [v_j^{\hat{R},1}]_1 = [v_j^{\hat{R},1}(\tau)]_1, [v_j^{\hat{R},2}]_1 = [v_j^{\hat{R},2}(\tau)]_1$.

Output $\mathsf{srs}_{\mathbf{W}} = \Big(\mathsf{srs}_{\mathsf{u}}, \big\{[v_j^{\hat{I}}]_1\big\}_{j=0}^{V}, \big\{[v_j^{\hat{R},1}]_1, [v_j^{\hat{R},2}]_1\big\}_{j=0}^{V-1}\Big)$.

$\mathsf{Prove}(\mathbf{W}, \mathsf{srs}_{\mathbf{W}}, (\boldsymbol{x}, (\boldsymbol{a}', \boldsymbol{b}'))):$ Sample $\boldsymbol{r}_a \leftarrow \mathbb{F}^4, \boldsymbol{r}_b \leftarrow \mathbb{F}^2$ and define $\boldsymbol{a} = (\boldsymbol{x}, \boldsymbol{a}', \boldsymbol{r}_a, \mathbf{1})$, $\boldsymbol{b} = (\mathbf{1}, \boldsymbol{b}', \mathbf{1}, \boldsymbol{r}_b)$. Then compute $A(X) = \sum_{j=1}^{m+6} a_j\lambda_j(X)$, $B(X) = \sum_{j=1}^{m+6} b_j\lambda_j(X)$, $B'(X) = \big(B(X)-1\big)/\big(t_l(X)\prod_{i=1}^{4}(X - \mathsf{h}_{m+i})\big)$, and

$$A'(X) := \left(\left(\sum_{j=l+1}^{m+6} a_j\lambda_j(X)\right) - t_l(X)\right) / \big(t_l(X)(X - \mathsf{h}_{m+5})(X - \mathsf{h}_{m+6})\big).$$

Output $\pi_1 = \big([A']_1 = [A'(\tau)]_1, [B']_1 = [B'(\tau)]_1\big)$.

$\mathsf{Verify}(\mathsf{srs}_{\mathbf{W}}, \boldsymbol{x}, \pi_1)$: Send $x, z_1, z_2 \leftarrow \mathbb{F}$.

$\mathsf{Prove}(\mathbf{W}, \mathsf{srs}_{\mathbf{W}}, (\boldsymbol{x}, (\boldsymbol{a}', \boldsymbol{b}')), x, z_1, z_2)$: For $i = 1, 2, 3$, let $D'_{\iota(i)}(X) = (D')^1_{\iota(i)}(X) + z_1(D')^2_{\iota(i)}(X)$. Let $D_a(X) = D'_a(X) + z_1^2\sum_{j=m+1}^{m+6}\lambda_j(X)$, $D_b(X) = D'_b(X) + z_1^2\sum_{j=m+1}^{m+6}\lambda_j(X)$ and $D_c(X) = D'_c(X)$.

Find $R(X), H_1(X), H_2(X)$ such that:

$$A(X)D_a(X)(X-u) + B(X)D_b(X)(X-u) - D_c(X)A(X)B(X)(X-u) = XR(X) + t(X)H_1(X)(X-u)$$

and, if $\hat{R}_x(X) = \sum_{j=0}^{V-1} x^j\big(v_j^{\hat{R},1}(X) + z_1 v_j^{\hat{R},2}(X)\big)$ and $\hat{I}_x(X) = \sum_{j=0}^{V} x^j v_j^{\hat{I}}(X)$,

$$D_c(X)\hat{I}_x(X) = t(x)\hat{R}_x(X) + H_2(X)t(X).$$

Output $\pi_2 = \big([D_c]_1 = [D_c(\tau)]_1,\ [H]_1 = [H_1(\tau)]_1 + z_2[H_2(\tau)]_1, [R]_1 = [R(\tau)]_1$.

$\mathsf{Verify}(\mathsf{srs}_{\mathbf{W}}, \boldsymbol{x}, \pi_1, \pi_2)$: Send $y, \gamma \leftarrow \mathbb{F}$.

$\mathsf{Prove}(\mathbf{W}, \mathsf{srs}_{\mathbf{W}}, (\boldsymbol{x}, (\boldsymbol{a}', \boldsymbol{b}')), x, z_1, z_2, y, \gamma)$: Define $\sigma = D_c(y)$ and, for

$$E(X) = A(y)D_a(y)(y-u) + B(X)D_b(y)(y-u) + \sigma\big(-A(y)B(X)(y-u) + z_2\hat{I}_x(X)\big) - yR(X) - z_2t(x)\hat{R}_x(X) - t(y)H(X)(y-u),$$

$\boldsymbol{p}(X) = (A(X), D_c(X), E(X))$, and $\boldsymbol{d} = (m-1, m-1, m-1)$, calculate $([w]_1, (a, \sigma, 0)) \leftarrow \mathsf{PC.Open}\,(\mathsf{srs}_{\mathsf{u}}, \boldsymbol{p}(X), \boldsymbol{d}, y, \gamma)$ and output $\pi_3 = ([w]_1, (a, \sigma))$.

$\mathsf{Verify}(\mathsf{srs}_{\mathbf{W}}, \boldsymbol{x}, \pi_1, \pi_2, \pi_3)$: Define $s = a + \gamma\sigma$ and $D_a = D_b = \big(t(x)y - xt(y)\big)/(x-y) - \sum_{j=m+1}^{m+6}\lambda_j(x)\lambda_j(y)$. Compute $[A]_1 = \big([A']_1(y - \mathsf{h}_{m+5})(y - \mathsf{h}_{m+6}) + 1\big)t_l(y) + \sum_{i=1}^{l} x_i[\lambda_i(\tau)]_1$, $[B]_1 = \big([B']_1 t_l(y)\prod_{i=1}^{4}(y - \mathsf{h}_{m+i}) + 1\big)$, $[\hat{R}_x]_1 = \sum_{j=0}^{V-1} x^j\big([v_j^{\hat{R},1}]_1 + z_1[v_j^{\hat{R},2}]_1\big)$, $[\hat{I}_x]_1 = \sum_{j=0}^{V} x^j[v_j^{\hat{I}}]_1$, and

$$[p]_1 = [A]_1 + \gamma[D_c]_1 + \gamma^2\big(aD_a + z_1D_b[B]_1 + \sigma\big(-a[B]_1(y-u) + z_2[\hat{I}_x]_1\big) - y[R]_1 - z_2t(x)[\hat{R}_x]_1 - t(y)[H]_1(y-u)\big)$$

Output 1 if and only if

$$e\big([p]_1 - [s]_1, [1]_2\big) = e\big([w]_1, [\tau - y]_2\big).$$

Fig. 13. zkSNARK for circuits with bounded fan-out. Elements in blue are added to achieve zero-knowledge.