vkproof: Succinct verification of indexed verifying keys using modular compilation and polynomial fingerprinting

Antonio Mejías Gil[†,1] and Xueqin Zhao[†,2]

[1]Provable, antonio@provable.com
[2]Chengdu Neusoft University, zhaoxueqin@nsu.edu.cn

Abstract

We introduce vkproof, a preprocessing SNARG which enables verification of the Varuna verifying key (or that of any similar proof system based on Marlin [Chi+20]) for the R1CS compiled from a given higher-level program. It has constant proof size and affords linear verifier costs in the number of instructions of the program rather than the density of the compiled R1CS, which makes it especially appealing in contexts where complex-to-arithmetise functions (such as hashing) appear as program instructions frequently. This verifier succinctness is achieved through *modular compilation* of programs and the use of *fingerprints* to verify polynomial correctness. We augment the algebraic holographic proof (AHP) model of Marlin by allowing oracles to witness polynomials in the instance and queries to linear combinations of indexed polynomials, resulting in a primitive we refer to as *extended algebraic holographic proofs* (eAHP).

Acknowledgements. The authors are indebted to Pratyush Mishra and Victor Sint Nicolaas for many valuable discussions, as well as to the authors of Marlin, which the *extended algebraic holographic proof* primitive is based on.

Contents

1 Introduction **3**

1.1 Our contributions 4

1.2 Related work 7

1.3 Organisation of the text 8

2 Extended algebraic holographic proofs (eAHP) **9**

2.1 Definition of eAHPs 10

[†]The two authors contributed equally to this work.

2.2 Homomorphic, separable and commitment-extractable polynomial commitment schemes 13
2.3 Preprocessing arguments for relations with commitments 18
2.4 Compilation of eAHPs with polynomial commitment schemes 20
2.5 Completeness and soundness of the compiled argument 23

3 Modular compilation of programs **28**
3.1 Vector and matrix fingerprinting 29
3.2 Individual-instruction information 32
3.3 Structural constraints 33
3.4 Hybrid polynomials and full-program relations 35

4 Protocol **39**
4.1 Overview of the eAHP 41
4.2 Indexer 42
4.3 Prover 42
4.4 Verifier 44
4.5 Compilation into a preprocessed SNARG 49

5 Properties **49**
5.1 Soundness of the eAHP 49
5.2 Costs 54

6 Future work **59**

A Power polynomials **63**

B Compilation with hiding polynomial commitment schemes **65**

1 Introduction

Interest in compiling high-level languages into some representation suitable for succinct proving of execution has surged in recent years, with programmable blockchains as one of the main driving forces. Motivating examples for this paradigm include rollups, i. e. L2 blockchains where execution of a number of transactions is proved using a succinct proof system and the resulting proof is sent to an L1 smart contract for verification (for instance Polygon[1] and Scroll[2]); and privacy-focused blockchains where validators verify a proof of correct execution of the private component of a transaction (such as Aztec[3]).

In Aleo[4], for instance, each program written in the custom-domain high-level language *Leo*, or its lower-level counterpart *Aleo instructions*, is compiled into a rank-one constraint system (R1CS) for proving and verification with the Varuna proof system[5], a variation on Marlin [Chi+20]. For reference, Marlin proving keys include three polynomials row, col and val for each of the three R1CS matrices (for a total of nine), whereas Varuna replaces val by the closely related rowcol and rowcolval for efficiency. Although we focus on the latter, the discussion applies to Marlin keys without any substantial changes.

Each program deployed onto the blockchain therefore comes with a fixed associated Varuna key pair: a proving key, used by the party executing the program (or any delegated entity proving that execution on their behalf); and a verifying key, used by any party wishing to verify the proof of execution – most notably, network validators. The question thus arises of *who* should generate those keys. In some conceivable designs, the blockchain validators that receive the verifying key alongside the program must simply *trust* said key. Note that a maliciously generated verifying key would enable critical attacks, as validators would be satisfied with proofs of execution of a functionality different than the program it purportedly corresponds to.

A strawman approach would be to have each validator generate the verifying key by themselves whenever a new program is deployed (we note at this point that Varuna verifying keys are *deterministically* computed from Aleo programs given the code of the latter). There are two fundamental issues with this approach:

- **Large R1CS**: the constraint systems that arise even from moderate programs are very large: both in terms of the number of rows and columns of their matrices and, more importantly for the purposes of Varuna, their densities (that is, the number of the non-zero entries they contain). Even though the R1CS compiled from a typical Aleo program tends to be very sparse (that is, a very large proportion of its entries have the value zero), computing the Varuna verifying key is challenging in practice even for such a constraint system[6]. This is largely due to operations such as hashing (for arithmetic and, especially, bit-based Hash functions) that have complex arithmetisations in terms of finite-field operations.

[1] https://polygon.technology/
[2] https://scroll.io/
[3] https://aztec.network
[4] https://aleo.org/
[5] https://github.com/ProvableHQ/protocol-docs/blob/main/snark/varuna/varuna-spec-prod.pdf
[6] The cost of verifying-key computation grows superlinearly in the density d of those matrices, more concretely as $O(d \cdot \log(d))$.

- **Expensive compilation**: even producing the R1CS (or its sparse description) given the program code can be computationally unfeasible in restricted environments. Compilers for relatively expressive virtual-machine environments such as Aleo's snarkVM[7] into R1CS (or any other typical SNARK-friendly arithmetisation) tend to be quite complex, as we briefly touch upon when discussing monolithic compilation below.

In order to address these challenges, we introduce vkproof, a preprocessing SNARG for Varuna-verifying-key correctness. Any party with access to a program P (for instance, the entity deploying P onto the blockchain) can compute its Varuna verifying key vk together with a *proof* π_{vk} attesting to its correctness. Other parties can then, given P and vk, verify π_{vk} and be satisfied that vk is correct and hence suitable for future verification of Varuna proofs of execution of P.

The main goals when designing vkproof were:

- **Succinctness**: verifier work should essentially be $O(|P|)$, where $|P|$ is the number of instructions in P. In particular, it should have dependence at most logarithmic in the complexity of P, by which we understand the density of the R1CS compiled from it. Note that, as a general guiding principle in SNARK design, $O(|P|)$ is optimal in that the verifier always needs to at least read the full instance – which, in this case, includes the program. This is a qualitative improvement over the $O(d \cdot \log(d))$ field-arithmetic and $O(d)$ elliptic-curve-arithmetic work needed to produce the verifying key, with d denoting the density of the R1CS. The costs of vkproofare dominated by two fixed-base elliptic-curve MSMs of a constant number of points.

  As a second aspect of succinctness, proofs should ideally be short. vkproof achieves constant proof size of $\sim$ 1.7 KB assuming $\sim$ 256-bit base and scalar fields when compiled with the Marlin (algebraic-group-model, non-hiding) PCS.

- **Reduced compilation costs**: the aspect of program compilation performed by the verifier in the proof system (which it necessarily will, since the verifying key encompasses the functionality of P) should be less computationally expensive than the original compilation algorithm, which, as discussed, is prohibitively costly in some relevant contexts.

- **No modifications to the Varuna proof system**: the above improvements should not come at the cost of substantially modifying how the Varuna proof system operates[8]. This would be problematic for existing pipelines using Varuna and also excessively expand the boundaries of the design space.

1.1 Our contributions

The proposed design achieves those aims by means of three main tools:

1) **Modular compilation**: A natural approach to designing an abstract compiler which arithmetises programs over a given language into R1CS triples is the following: for each instruction ν in the language, precompute an R1CS (three matrices (A_ν, B_ν, C_ν)) capturing its functionality. On input a program P, incorporate the R1CS of each instruction in P into a running

[7] https://github.com/ProvableHQ/snarkVM

[8] For instance, at an early stage of the project, expanding Varuna to include a permutation argument to enforce relations between program variables existing as duplicate columns of the R1CS was considered.

overall-program one. If an instruction ν_i is applied to some variables that appeared earlier in the program, some columns of $(A_{\nu_i}, B_{\nu_i}, C_{\nu_i})$ are displaced to certain positions in the running R1CS to force the variables they represent to coincide with their previous appearances in the program. We refer to this paradigm as *monolithic compilation*, because this "column shuffling" makes it difficult to relate individual-instruction R1CS matrices to the overall-program one – and this relation is something a proof system hoping to incur linear verifier costs in the size of P would likely rely on. Modular compilation also imposes a significant computational burden on the compiler itself, which must really *materialise* the overall-program R1CS to capture the shuffling of entries.

Instead, we propose a *modular compiler* where each individual-instruction R1CS really appears as a (contiguous) *submatrix* of the overall-program one. More specifically, the A_{ν_i} are diagonally concatenated to one another, and similarly for B_{ν_i} and C_{ν_i}. This means that a vkproof verifier[9] does not need to materialise the full R1CS: it only needs a *virtual* description of it, which is precisely the program. In order to capture the relations between variables that are lost when diagonally concatenating individual-instruction R1CS, the compiler also adds a number of *structural constraints*. The resulting full-program R1CS does have more non-zero entries than those produced by monolithic compilation, which translates increased computational costs for the Varuna prover. However, structural constraints (the only component introducing new non-zero entries with respect to monolithic compilation) are *extremely* sparse, which makes the additional prover costs negligible if the program has complex instructions.

2) **Preprocessed data**: In order to make vkproof verifier costs only grow as $O(|P|)$ regardless of the complexity (more formally: density) of each of the $|P|$ instructions in the program, each instruction in the language is preprocessed into some data that can be used to process any given program. This preprocessing is carried out by a trusted party known as the *indexer* in the context of algebraic holographic proofs [Chi+20] (AHP)[10] and other proof systems. This indexer only needs to be trusted *once*: after a fixed set of language instructions are preprocessed, verifying-key proofs can be generated for any (admissible) program over that language – in other words, no party needs to be trusted on a per-program basis. In fact the preprocessed data is publicly verifiable without any trust assumptions and can be updated with new instructions after the initial indexing phase.

   Some readers may appreciate the irony of resorting to holography (more informally, preprocessing) in order to verify indexed verifying keys of a proof system which is in itself holographic - indeed, Varuna (and, more generally, Marlin) verifying keys are meant to provide fast verification of R1CS instances by preprocessing the constraint system in question.

3) **Polynomial fingerprinting**: As already mentioned, a Varuna verifying key essentially consists of three polynomials per matrix (for a total of nine; three of which are in fact replaced by six related ones for increased efficiency) which, with some caveats, interpolate the row positions, column positions and values of the matrix non-zero entries. Instead of preprocessing those same polynomials for each individual-instruction R1CS, we turn to fingerprint polynomials. By the *fingerprint* of a polynomial f at a point $\alpha \in \mathbb{F}$ we simply mean the sum of its evaluations (over some specific set) each scaled by a different power of a fixed element

[9] It is possible modular compilation can be extended to also facilitate key *generation*.

[10] This then gives rise to the "preprocessing" in "preprocessing SNARGs" after AHP compilation.

α. The *fingerprint polynomial* of f is the polynomial that maps each α to the fingerprint of f at α. vkproof checks the fingerprints of the polynomials in the Varuna verifying key against their expected values, computed largely from preprocessed data. Switching from preprocessing the Varuna-key polynomials to polynomials which compute their fingerprints has several important benefits:

- It makes the proof size constant as opposed to $O(|P|)$. Without delving into the details, the main reason is that fingerprint polynomials are easier to collate into full-matrix ones than those in the Varuna verifying key. When collating the latter, their variables are shifted in a way which would require the verifier to query them at a distinct point for each polynomial, thus forcing the prover to send an evaluation per Varuna-key polynomial *per instruction*, resulting in linear proof sizes. Furthermore, depending on the PCS chosen for compilation, checking batched opening proofs may still require one elliptic-curve pairing per queried point (this is, for instance, the case in the Marlin PCS [Chi+20]), which would be a prohibitively high cost.
- It significantly reduces the cost associated to structural constraints. The fingerprints (at some $\alpha \in \mathbb{F}$) of the Varuna polynomials of a full-program R1CS easily split into one part corresponding to the submatrix containing the instruction submatrices, and a second one corresponding to structural constraints. Computing the latter has a cost (in terms of field multiplications) which is linear in the number of instructions of the program, and concretely very small. Attempting the same split with the original Varuna polynomials (instead of their fingerprints), would require to evaluate a row, col or val polynomial interpolating the structural-constraint part of the R1CS and *zeros everywhere else in the full Varuna domain*. This means the costs in terms of field arithmetic would still be $O(d \log(d))$ in the density d of the full program R1CS (assuming FFTs were used).

4) **Extended algebraic holographic proofs**: Our information-theoretic protocol is inspired by the algebraic holographic proof (AHP) primitive introduced in Marlin [Chi+20] yet does not quite fit that formalism. One reason is that the relation we eventually want to prove with a preprocessing argument, which captures correctness of Varuna verifying keys, includes polynomial commitments in its instances: indeed, an $\mathcal{R}_{\mathsf{VarunaVK}}$ instance consists of the twelve commitments to the polynomials in the corresponding Varuna proving key, as well as the program P they purportedly capture. Furthermore, our goal of achieving constant-time proofs is only enabled by the fact that the verifier can query a linear combination of polynomials without querying each of them individually (thus avoiding having to receive that many evaluations from the prover).

   In order to address both challenges at the information-theoretic-protocol level, we expand the AHP by allowing the instance to contain oracles to distinguished polynomials, and allowing the verifier to query linear combinations of (certain) polynomial oracles. The result is a primitive we term *extended algebraic holographic proofs* (eAHP). We introduce this primitive, provide a black-box compiler which produces a succinct argument given an eAHP and a polynomial commitment scheme (formally introducing the homomorphic property required of the latter along the way) and prove the completeness and soundness of the compiled protocol.

1.2 Related work

Since our protocol is aimed at shifting verification work on indexed verifying keys from the verifier to the prover and relies, among other techniques, on modular compilation of programs, related work comes from several research areas: proof systems for virtual-machine (VM) environments, distributed or delegated computation of proving and verifying keys, and attempts to verify key correctness through circuit recursion. In this subsection, we emphasize the differences in design goals and threat models between those paradigms and our protocol.

Modular-compilation-based approaches for VM environments A large body of recent work has tackled the problem of proving the execution of complex programs in a virtual machine defined by some instruction-set architecture (ISA), including models enriched with some notion of *public state* to capture the expressivity of programmable blockchains. These approaches compile a program into a sequence of instructions and allow the prover to show the correctness of each instruction's semantics, the program's execution trace, and memory operations. Such techniques are already in use in production systems including SP1's zkVM[11], RISC Zero[12], Polygon's zkEVM[1], and zkSync's EraVM[13]. The cryptography involved typically relies on techniques such as lookups (with Jolt[AST24], which leverages Lasso [STW24] lookups, as a prime example), STARKs and recursion.

The crucial difference between all of the above and vkproof is that our protocol is not aimed at (and indeed does not enable) proving and verification of program execution itself: that is still handled by the Varuna proof system – which does not take particular advantage of modular, ISA-based circuitry. Our only reason for introducing modular compilation is making Varuna proving and verifying keys themselves succinctly provable and verifiable.

Delegated and distributed computation of verification keys A natural way to address the question of verifying-key correctness in contexts where synthesis is computationally intensive is to outsource key generation to a trusted party. However, this entails a non-trivial trust assumption which, in the case of a programmable blockchain with private execution such as Aleo, occurs for each new program function deployed onto the chain. In order to mitigate this, key generation can be performed it in a distributed fashion (for instance, a multi-party computation (MPC) ceremony) with minimal trust assumptions, which only requires an inexpensive verification step afterwards for any interested parties.

Classical and production-grade MPC systems such as Global-Scale Secure MPC [WRK17] and MP-SPDZ [Buc+25] support outsourcing linear algebraic computations while preserving privacy[14]. However, these general-purpose MPC systems typically incur large communication and computation overhead, making them ill-suited for latency-sensitive applications such as blockchains. In the context of zero-knowledge proofs, systems like Eos [Chi+23] and Siniel [Yan+24] have investigated

[11] https://github.com/succinctlabs/sp1
[12] https://risczero.com
[13] https://docs.zksync.io/zksync-protocol/era-vm/vm
[14] In our main setting of interest, privacy is not necessary as verifying (and proving) keys can be synthesised from the publicly available program code.

privacy-preserving verifiable computation delegation, where zkSNARK computation is parallelized across multiple workers.

Our approach differs fundamentally in that it targets Marlin-based zkSNARK systems with circuit-specific setup phases and does not involve delegated or distributed key (or proof) generation. The only trust assumption in vkproof is in the one-off indexing phase (which is publicly verifiable, although computationally expensive): every individual-program key can subsequently be locally verified against malicious key generation. Existing work on distributed or outsourced verification key generation differs from this in terms of the underlying security model, communication cost (which is minimal in vkproof), computational overhead and fault tolerance.

Circuit-based verification of verifying keys A conceptually simple way to ensure correctness of verifying-key synthesis would be to encode the compilation and key-generation process *itself* as a circuit in some standard DSL (e. g. Circom) or lower-level arithmetisation (R1CS, PLONK [GWC19], etc.) and then use an existing compatible proof system (such as Zcash' Halo2[15] for PLONK) to ensure correctness of each program-to-key execution. There is, however, a sizeable practical obstacle to this approach: the arithmetisation of a program such as the snarkVM compiler would be extraordinarily complex. Observe that the resulting arithmetised circuit would capture each possible execution path of the compiler (that is, each possible input program) simultaneously, which in the case of languages as expressive and complex as Aleo instructions (which includes, for instance, input- and output-record encryption and hashing) would be prohibitively expensive to produce proofs for.

The verifier of a proof of execution of such a vast arithmetised compiler would be faced with similar computational challenges, which could be mitigated with the usual recursion techniques. However, proof recursion increases the prover work (which was already too high) and introduces other architectural challenges such as curve chaining, cross-curve compatibility and handling of trusted setups within recursion (such as long structured reference strings). Even when recursion is feasible, the handling of low-level components such as polynomial commitments often incurs a significant overhead, further reducing real-world applicability.

Instead, vkproof offers a lightweight, tailored SNARG which avoids the arithmetisation explosion and the engineering burden of recursive constructions.

1.3 Organisation of the text

Section 2 describes the eAHP primitive, the properties of a PCS that are needed for compilation of an eAHP, and the completeness and soundness of the resulting preprocessing argument. The third section focuses on the specifics of the modular compiler, introducing fingerprint polynomials and describing the relations between individual-instruction data and the Varuna-key polynomials of the compiled R1CS. Section 4 presents the vkproof eAHP itself. Its soundness and costs are the object of section 5. Finally, section 6 outlines some possible directions for future work.

[15] https://zcash.github.io/halo2/index.html

2 Extended algebraic holographic proofs (eAHP)

Our protocol is formulated as a variant of the algebraic holographic proof (AHP) primitive introduced in Marlin [Chi+20]. Recall that an AHP is a special type of interactive proof system where

- a trusted *indexer* precomputes certain polynomials given an index $\mathtt{i}$ and a finite field $\mathbb{F}$.
- the prover receives the same index $\mathtt{i}$ and field $\mathbb{F}$ as the indexer (alongside the instance $\mathtt{x}$ and witness $\mathtt{w}$) and the verifier receives the same field $\mathbb{F}$ as the indexer (alongside $\mathtt{x}$).
- all prover messages to the verifier are polynomials over $\mathbb{F}$ with preestablished degree bounds.
- all verifier messages are sequences of elements in $\mathbb{F}$ of preestablished length.
- assuming the AHP is public coin (that is, verifier messages are uniformly random strings of field elements), the verifier's operation is split into the following phases, in order:
  - an *interactive phase*, where it receives messages from the prover and replies with its own messages.
  - a *query phase*, where it consults the evaluations at any desired points of polynomials precomputed by the indexer or sent by the prover.
  - a *decision phase*, where it performs a number of calculations (possibly involving the evaluations learnt during the query phase) and either accepts or rejects.

More details can be found in section 4 of the cited article. As explained in op. cit., one can compile such an AHP together with an extractable polynomial commitment scheme (PCS) which enforces degree bounds into a *preprocessing argument with universal SRS* by essentially having the indexer and prover *commit* to polynomials instead of directly sending them to the verifier, and subsequently replacing the verifier's query phase by a number of commitment openings and opening checks, which can be batched for efficiency.

If the original AHP was public coin, the preprocessing argument compiled as above can furthermore be made non-interactive using the Fiat-Shamir transform, thus becoming a *preprocessing SNARG with universal SRS*[16] (potentially a SNARK and potentially zero knowledge depending on the properties of the AHP and the PCS). [Chi+20] furthermore restricts its attention to *non-adaptive* AHPs, where the points queried by the verifier during the query phase depend only on its internal randomness and input (and not, for instance, on prover messages or the replies to other queries). This allows the evaluation opening and opening checks to take place in a single execution of the PCS's batched opening and checking methods.

In order to accurately model our protocol and evaluate its security, the AHP primitive must be augmented with two additional features:

i) The instance $\mathtt{x}$ can include oracles to certain polynomials in the witness $\mathtt{w}$.

[16] In the sequel, we may abbreviate this to "preprocessing SNARG" and "preprocessing argument with universal SRS" to "preprocessing argument".

This is dictated by the nature of our problem, where the instance to be verified (which includes a Varuna verifying key) contains commitments to some witness polynomials (namely, to those in the Varuna proving key).

ii) The verifier has the ability to query not only polynomials exposed by the indexer, but also linear combinations of them – that is, expressions of the form $s_0 \cdot f_0 + \ldots + s_n \cdot f_n$, where the f_i are such polynomials and the s_i are elements of $\mathbb{F}$.

This need arises from the vkproof verifier combining preprocessed individual-instruction fingerprint polynomials (or, after compilation with a PCS: their commitments) into their full-program analogues.

Note that, for reasons that will become apparent shortly, the possibility to query linear combinations of polynomials does not extend to polynomial oracles in the instance or to polynomials sent by the prover during the interactive phase.

Some other minor differences are only detailed in the formal definitions below. For instance, degree bounds of round polynomials must be allowed to depend on the (non-oracle part of the) instance $\mathbb{x}$ instead of only the index $\mathbb{i}$, as do the degree bounds of the newly introduced instance polynomials. Motivated by this, indices are required to include an overall degree bound to control the aforementioned, instance-dependent bounds.

We refer to an interactive protocol as above as an *extended* algebraic holographic proof (eAHP). In this section, we formally define eAHPs, describe their compilation to a preprocessing argument with universal SRS and prove the soundness of the latter. Although minor, the introduced modifications give rise to some subtleties (for instance, the need for a formal notion of homomorphic and separable PCS for efficiency or the appearance of commitments in the relevant indexed relation) which are carefully addressed. We do not define or study knowledge soundness or zero knowledge for eAHPs as those are not relevant notions to our protocol of interest: in the Varuna verifying-key relation (detailed at the start of section 4), the witness consists only of the twelve Varuna prover-key polynomials, which can be easily recovered from the program P contained in the instance.

2.1 Definition of eAHPs

The formal definition of extended algebraic holographic proofs largely follows that of standard algebraic holographic proofs in [Chi+20] section 4. For convenience, a full definition is given below and changes with respect to the original are highlighted in blue. Before that, we define a version of indexed relations whose instances may oracles to witness polynomials. This is not an infrequent technique (cf. for instance [Che+23] definition 2.3), but we tailor it to our specific needs.

Definition 1. An **indexed polynomial-oracle relation** $\mathcal{R}$ over a family $\mathcal{F}$ of finite fields is a set of triples $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ where

i) the index $\mathbb{i}$ contains the description of a finite field $\mathbb{F}_{\mathbb{i}} \in \mathcal{F}$ and a tuple of degree bounds $(\mathsf{N}_{\mathbb{i},j})_{j=1}^{d} \in \mathbb{N}^d$. We set $\mathsf{N}_{\mathbb{i}} = \max_{1 \leq j \leq d} \mathsf{N}_{\mathbb{i},j}$ and refer to it as the **index degree bound**.

ii) the witness $\mathbb{w}$ is a pair $\mathbb{w} = (\mathbb{w}_{\mathsf{o}}, \widetilde{\mathbb{w}})$, with $\mathbb{w}_{\mathsf{o}} = (f_i)_{i=1}^{n}$ in turn consisting of a (possibly empty) tuple of univariate polynomials $f_i \in \mathbb{F}_{\mathbb{i}}[x]$.

iii) the instance $\mathbb{x}$ is a pair $\mathbb{x} = (\mathbb{x}_{\mathsf{o}}, \widetilde{\mathbb{x}})$, with $\mathbb{x}_{\mathsf{o}} = ([[f_i]], \mathsf{d}_i)_{i=1}^n$ containing oracles $[[f_i]]$ to all witness polynomials f_i and degree bounds[17] $\mathsf{d}_i \in \mathbb{N}$ such that $\deg(f_i) \leq \mathsf{d}_i$ for all i.

The number n of polynomials, degrees and oracles is the same in the instance and witness but can vary for each triple in $\mathcal{R}$. The **language** defined by $\mathcal{R}$ is the set $\mathcal{L}(\mathcal{R})$ of all pairs $(\mathbb{i}, \mathbb{x})$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ for some $\mathbb{w}$. □

Definition 2. An **extended algebraic holographic proof** (eAHP) for an indexed polynomial-oracle relation $\mathcal{R}$ over a family $\mathcal{F}$ of finite fields is a tuple

$$\mathsf{eAHP} = (\mathsf{k}, \mathsf{s}, \mathsf{d}, \mathbf{I}, \mathbf{P}, \mathbf{V})$$

such that:

- k, s and d are (deterministic) polynomial-time functions $\{0,1\}^* \to \mathbb{N}$ specifying the number $\mathsf{k}(|\mathbb{i}|)$ of rounds of interaction between $\mathbf{I}$ and $\mathbf{P}$; the number $\mathsf{s}(\mathbb{i}, 0)$ of polynomials output by $\mathbf{I}$ during preprocessing and the number $\mathsf{s}(\mathbb{i}, i)$ of them sent by $\mathbf{P}$ in the i-th round of interaction[18]; and the degree bounds of each polynomial, respectively. More precisely,
  - $\mathsf{d}(\mathbb{i}, 0)$ is the degree bound for *all* indexed polynomials given $\mathbb{i}$ (and does not only depend on $|\mathbb{i}|$).
  - $\mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, i, j)$ with $i \in \{1, \ldots, \mathsf{k}(|\mathbb{i}|)\}$ and $j \in \{1, \ldots, \mathsf{s}(\mathbb{i}, i)\}$ specifies the degree bound of the j-th polynomial sent by $\mathbf{P}$ in the i-th round of interaction. Note this can depend on $\mathbb{i}$ (and not only $|\mathbb{i}|$) as well as the non-oracle part $\widetilde{\mathbb{x}}$ of the instance $\mathbb{x}$.

  For any given index $\mathbb{i}$ with degree-bound tuple $(\mathsf{N}_{\mathbb{i},j})_{j=1}^d$ we require that, for all triples $(\mathbb{i}, \mathbb{x} = (([[f_i]], \mathsf{d}_i)_{i=1}^n, \widetilde{\mathbb{x}}), \mathbb{w}) \in \mathcal{R}$, all of the following values appear in the tuple: d_i for all $i \in \{1, \ldots, n\}$, and $\mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, i, j)$ for all $i \in \{1, \ldots, \mathsf{k}(|\mathbb{i}|)\}, j \in \{1, \ldots, \mathsf{s}(\mathbb{i}, i)\}$. Note $\mathsf{d}(\mathbb{i}, 0)$ is not required to belong to the tuple[19]. However, we require $\mathsf{d}(\mathbb{i}, 0) \leq \mathsf{N}_{\mathbb{i}}$.

- $\mathbf{I}$, known as the *indexer*, is a non-interactive deterministic polynomial-time algorithm which receives an index $\mathbb{i}$ and outputs polynomials $f_{0,j}, \ldots, f_{0,\mathsf{s}(\mathbb{i},0)} \in \mathbb{F}_{\mathbb{i}}[x]$ with $\deg(f_{0,j}) \leq \mathsf{d}(\mathbb{i}, 0)$ for all $j \in \{1, \ldots, \mathsf{s}(\mathbb{i}, 0)\}$.
- $\mathbf{P}$ and $\mathbf{V}$ are interactive PPT algorithms that operate as follows for any $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$:
  - $\mathbf{P}$, known as the *prover*, receives as input $(\mathbb{i}, \mathbb{x}, \mathbb{w})$.
  - $\mathbf{V}$, known as the *verifier*, receives oracle access to $\mathbb{i}$ (including the description of $\mathbb{F}_{\mathbb{i}}$) and to the polynomials $f_{0,i}$ output by the indexer on input $\mathbb{i}$; and receives as input the full instance $\mathbb{x}$.

[17] No specific degree bound "$-\infty$" is offered for the zero polynomial. Degree bounds relate to polynomial commitments later in the compilation process, and there is limited interest in committing to a polynomial guaranteed to be zero. This formal case can be handled with the degree bound zero (corresponding to constant polynomials) plus a commitment opening to reveal the constant value.

[18] Unlike in Marlin, we allow these numbers to depend on the index. In the vkproof eAHP, only the number of indexed polynomials does: that of round polynomials is fixed.

[19] The reason is that the information $(\mathsf{N}_{\mathbb{i},j})_{j=1}^d$ will be used by the eAHP indexer $\mathbf{I}$ on input $\mathbb{i}$ in order to trim the public parameters into a PCS committer and verifier key that can be used with all instance and prover-sent commitments for *any* instance $\mathbb{x}$ such that $(\mathbb{i}, \mathbb{x}) \in \mathcal{L}(\mathcal{R})$. However, the degree bound $\mathsf{d}(\mathbb{i}, 0)$, which is specific to $\mathbb{i}$, can be computed by $\mathbf{I}$ on the spot and hence does not need to be provided by $\mathbb{i}$ itself. Indeed, in our eAHP of interest $\mathsf{eAHP}_{\mathsf{vkproof}}$, instance and round polylnomials are limited to a very small set of degree bounds (essentially, a few powers of two) whereas $\mathsf{d}(\mathbb{i}, 0)$ is completely arbitrary depending on $\mathbb{i}$. Forcing $\mathbf{I}$ to trim the public parameters to any possible $\mathsf{d}(\mathbb{i}, 0)$ would result in, for instance, excessively large verifier keys in the case of the Marlin PCS.

- $\mathbf{P}(\mathtt{i}, \mathtt{x}, \mathtt{w})$ and $\mathbf{V}^{\mathtt{i},\mathbf{I}(\mathtt{i})}(\mathtt{x})$ interact for $\mathsf{k}(|\mathtt{i}|)$ rounds.
- In each round $i \in \{1, \ldots, \mathsf{k}(|\mathtt{i}|)\}$, the verifier sends the prover a string $\rho_i \in (\mathbb{F}_\mathtt{i})^*$ of field elements[20] and the prover replies with $\mathsf{s}(\mathtt{i}, i)$ polynomials $f_{i,1}, \ldots, f_{i,\mathsf{s}(\mathtt{i},i)} \in \mathbb{F}_\mathtt{i}[x]$ with each $f_{i,j} \in \mathbb{F}_\mathtt{i}[x]$ of degree $\leq \mathsf{d}(\mathtt{i}, \widetilde{\mathtt{x}}, i, j)$.
- At any point, the verifier may query the evaluation at a point $\alpha \in \mathbb{F}_\mathtt{i}$ of their choice of:
  - a polynomial $f_{i,j}$ sent by the prover (i. e. with $i \geq 1$).
  - a *linear combination* $s_0 \cdot f_{0,j_0} + \ldots + s_m \cdot f_{0,j_m}$, where the $f_{0,j}$ are polynomials output by the indexer and the coefficients $s_j \in \mathbb{F}_\mathtt{i}$ are arbitrarily chosen by $\mathbf{V}$.
  - a polynomial $f_{0,j}$ whose oracle $[[f_{0,j}]] \in \mathtt{x}_\mathsf{o}$ is contained in the instance $\mathtt{x}$.

  For brevity, we unify the notation and terminology of all three cases above and refer to $\mathbf{V}$ querying an *oracle* to $[[f_{i,j}]]$ also in the case of a prover-sent polynomial $f_{i,j}$; and a linear combination $s_1 \cdot [[f_{0,j_1}]] + \ldots + s_n \cdot [[f_{0,j_n}]]$ of *oracles* $[[f_{0,j_t}]]$ in the case of a linear combination of polynomials f_{0,j_j} output by the indexer.

An eAHP as above is said to be

- **public coin** if the messages sent by the verifier during the interactive phase are uniformly sampled from $\mathbb{F}^l$ (for a possibly different l in each round).
- **non-adaptive** if the points, oracles and coefficients s_i of linear combinations of oracles of all verifier queries at the end of the interaction depend only on the non-oracle part $\widetilde{\mathtt{x}}$ of the instance $\mathtt{x}$ and the messages sent by the verifier to the prover. More specifically, there exists a (deterministic) polynomial-time query function $\mathbf{Q}$ such that the set Q of points and oracles (or linear combinations of oracles) queried by $\mathbf{V}$ after the $\mathsf{k}(|\mathtt{i}|)$ rounds of interaction with $\mathbf{P}$ equals[21] $\mathbf{Q}^{\mathtt{i}}(\widetilde{\mathtt{x}}, \rho_1, \ldots, \rho_{\mathsf{k}(|\mathtt{i}|)}, \rho_{\mathsf{k}(|\mathtt{i}|)+1})$.

It is said to

- be **complete** if, for all $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$, the probability that $\mathbf{V}^{\mathtt{i},\mathbf{I}(\mathtt{i})}(\mathtt{x})$ accepts after interacting with $\mathbf{P}(\mathtt{i}, \mathtt{x}, \mathtt{w})$ is 1.
- have **soundness error** $\varepsilon = \varepsilon(\mathbb{F})$ if, for all $(\mathtt{i}, \mathtt{x}) \notin \mathcal{L}(\mathcal{R})$ and all (possibly malicious) admissible provers $\widetilde{\mathbf{P}}$, the probability that $\mathbf{V}^{\mathtt{i},\mathbf{I}(\mathtt{i})}(\mathtt{x})$ accepts after interacting with $\widetilde{\mathbf{P}}$ is at most $\varepsilon(\mathbb{F}_\mathtt{i})$. A prover is said to be admissible if, for all $1 \leq i \leq \mathsf{k}(|\mathtt{i}|)$ and $1 \leq j \leq \mathsf{s}(\mathtt{i}, i)$, the j-polynomial $\widetilde{f}_{i,j}$ it sends in the i-th round has degree at most $\mathsf{d}(\mathtt{i}, \widetilde{\mathtt{x}}, i, j)$ (where $\mathtt{x} = (\mathtt{x}_\mathsf{o}, \widetilde{\mathtt{x}})$), as dictated by eAHP.

Unlike in AHPs, the eAHP primitive requires all indexed polynomials to be endowed with the same degree bound. The motivation is that, in general, only then can commitments produced with those bounds be linearly combined (cf. subsection 2.2 below), which constitutes an important part of the eAHP compilation process. The main purpose of degree bounds in PIOP-like schemes is

[20] We use parentheses to distinguish the family $(\mathbb{F}_\mathtt{i})^*$ of (finite) strings over $\mathbb{F}_\mathtt{i}$ from the group of units $\mathbb{F}_\mathtt{i}^*$ of $\mathbb{F}_\mathtt{i}$.

[21] Although not required by our eAHP of interest, $\mathsf{vkproof}$, this definition allows the coefficients of linear combinations of queried oracles to depend on the query randomness $\rho_{\mathsf{k}(|\mathtt{i}|)+1}$. A more restrictive design which would suffice for our purposes would be to split $\mathbf{Q}$ into two algorithms, one of them receiving only $\rho_1, \ldots, \rho_{\mathsf{k}(|\mathtt{i}|)}$ and outputting the set of queried oracles and linear combinations of oracles (including coefficients); and a second one additionally receiving $\rho_{\mathsf{k}(|\mathtt{i}|)+1}$ and outputting the query points for those oracles.

to bound the soundness error by means of the Schwartz-Zippel lemma, which is not relevant for polynomials known to have been committed honestly – as is the case for indexed polynomials in AHPs, – where all that is needed is an evaluation-binding-inspired guarantee. If necessary, the eAHP could be modified so that the indexer produced two families of polynomials: one which only contained polynomials with the same degree bounds, which would accept queries to linear combinations of the corresponding oracles; and one containing polynomials which separate degree bounds, whose oracles could only be queried individually.

As with AHPs, public-coin eAHPs can be structured into three ordered phases: interaction, querying and decision. Together with public-coinness, non-adaptiveness allows for all queries (including those to linear combinations of indexed polynomials) to be collected into a single query set Q and handled by a single batched opening and opening check after compilation with a PCS. In the sequel, we only concern ourselves with eAHPs satisfying both properties, as well as admissibility.

Compilation of an extended algebraic holographic proof with a polynomial commitment scheme proceeds in a similar manner to that of a standard AHP (cf. [Chi+20] section 8) with two main modifications. Firstly, instance oracles are (similarly to indexed and prover-sent polynomials) replaced by commitments, with queries to them becoming corresponding PCS openings (recall that the witness $\mathbb{w}$ contains the polynomials in full). Secondly, queries to linear combinations of indexed polynomials are processed by having the verifier *compute a commitment to the combined polynomial*, followed by a commitment opening. This necessitates a formal notion of homomorphism for the PCS primitive defined in op. cit. We formalise it and address various subtleties arising from it in the next subsection.

2.2 Homomorphic, separable and commitment-extractable polynomial commitment schemes

Starting at the PCS formalism from Marlin ([Chi+20] section 6), we introduce the three additional properties a given scheme must satisfy in order to be suitable for compilation of an eAHP. For reasons that will become apparent shortly, we restrict our attention to polynomial commitment schemes *without hiding randomness* and stress this by writing $\mathsf{PC}_{\mathsf{NH}} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$. In the two methods involving hiding randomness, Commit and Open, we omit the corresponding argument ω from the notation altogether[22]. Note that it is not sufficient to take a PCS with hiding randomness and simply have the honest prover use empty randomness $\omega = \perp$ in commit and open, since nothing would guarantee the verifier this was indeed the case. For instance, in the specific PCS from [Chi+20] appendices B-D, removing hiding randomness involves discarding the parts in blue, including the generation of the portion $\{\gamma G, \gamma\beta G, \ldots, \gamma\beta^D G\}$ of the SRS: otherwise, the extractor of agiven algebraic adversary is not guaranteed to recover polynomials without hiding randomness. We briefly discuss the possibility of using hiding polynomial commitment schemes for eAHP compilation in appendix B.

All three conditions we introduce below are very mild and, in particular, are satisfied by the PCS in [Chi+20] (even in their hiding variant, which we do not state formally).

[22] This can be enforced still within Marlin's formalism by, for instance, requiring Commit and Open to output the same commitment and proof for any given value of the input ω.

Definition 3. Let $\mathsf{PC}_{\mathsf{NH}} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ be a (non-hiding) polynomial commitment scheme over a family $\mathcal{F}$ of finite fields as defined in [Chi+20] section 6.1. $\mathsf{PC}_{\mathsf{NH}}$ is said to be **homomorphic** if both of the following hold:

i) Commitments belong to vector spaces over the field contained in the parameters. More formally, for each security parameter $\lambda \in \mathbb{N}$, maximum degree bound $D \in \mathbb{N}$ and public parameters[23] $\mathsf{pp} \in \mathrm{supp}(\mathsf{PC}_{\mathsf{NH}}.\mathsf{Setup}(1^\lambda, D))$, there exists a vector space V over the field $\mathbb{F}_{\mathsf{pp}}$ such that, for all

- degree bounds $(d_i)_{i=1}^n \in \mathbb{N}^n$ with $d_i \leq D$ for all i
- polynomials $(f_i)_{i=1}^n \in \mathbb{F}[x]$ with $\deg(f_i) \leq d_i$ for all i

one has

$$\mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}, (f_i)_{i=1}^n, (d_i)_{i=1}^n) = (c_i)_{i=1}^n \in V^n$$

(if $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}$ does not abort), where[24] $(\mathsf{ck}, \mathsf{vk}) = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Trim}^{\mathsf{pp}}(1^\lambda, (d_i)_{i=1}^n)$.

ii) Commitments to polynomials *with the same degree bound* are $\mathbb{F}_{\mathsf{pp}}$-linear. More formally, for all λ, D, pp, $(d)_{i=1}^n$ (note now there is a single degree bound $d = d_1 = \cdots = d_n$) and $(f_i)_{i=1}^n$ as above (in particular, with $\deg(f_i) \leq d$ for all i), one has

$$\sum_{i=1}^n s_i \cdot \mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}, (f_i)_{i=1}^n, (d)_{i=1}^n)_i = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}\left(\mathsf{ck}', \left(\sum_{i=1}^n s_i \cdot f_i\right), (d)\right) \tag{1}$$

(if none of the $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}$ aborts) for all coefficients $s_1, \ldots, s_n \in \mathbb{F}_{\mathsf{pp}}$, where

- $(\mathsf{ck}, \mathsf{vk}) = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Trim}^{\mathsf{pp}}(1^\lambda, (d_i)_{i=1}^n)$ and $(\mathsf{ck}', \mathsf{vk}') = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Trim}^{\mathsf{pp}}(1^\lambda, (d))$ (note that $\sum_{i=1}^n s_i \cdot f_i$ has degree at most d).
- $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}, (f_i)_{i=1}^n, (d_i)_{i=1}^n)_i$ denotes the i-th element c_i of

$$\mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}, (f_i)_{i=1}^n, (d_i)_{i=1}^n) = (c_i)_{i=1}^n.$$

- on the left-hand side, multiplication by s_i and addition occurs in the vector space V from point i), which is the same for all terms of the sum.

iii) There exists an deterministic polynomial-time algorithm to compute a linear combination of commitments as above. □

The second necessary feature is a slight strengthening of the extractability property from [Chi+20] definition 6.2. The latter roughly requires that, for any given adversary which outputs commitments and degree bounds, there exists an extractor which recovers polynomials such that, if the opening check for some claimed evaluations passes on those commitments, then the extracted polynomials satisfy the claimed given bounds and have the purported evaluations. We simply add the very mild condition that the extracted polynomials indeed have the commitments produced by the adversary, which is needed in order to conclude the soundness of the argument compiled from an eAHP and a PCS as, in the relation the argument refers to, the instance contains commitments to witness polynomials. We highlight the only change with respect to Marlin's definition of extractability in blue.

[23] By the support $\mathrm{supp}(A(x))$ of a possibly randomised algorithm A on input x we mean the set of values $A(x)$ outputs with probability greater than zero.

[24] Recall that $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Trim}$ is deterministic by definition.

Definition 4. Let $\mathsf{PC}_{\mathsf{NH}} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ be a (non-hiding) polynomial commitment scheme over a family $\mathcal{F}$ of finite fields as defined in [Chi+20] section 6.1. PC is said to have **commitment-extractability error** $\varepsilon(\lambda)$ if, for every degree round $D \in \mathbb{N}$ and efficient adversary $\mathcal{A}$ there exists an efficient extractor $\mathcal{E}$ such that, for every round bound $\mathsf{r} \in \mathbb{N}$, efficient public-coin challenger $\mathcal{C}$ (i. e. only sending uniformly random strings of a fixed length), efficient query sampler $\mathcal{Q}$ and efficient adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$, one has

$$\Pr\left[\begin{array}{c} \mathsf{Check}(\mathsf{vk}, (c_{i,j})_{i,j}, (d_{i,j})_{i,j}, Q, v, \pi, \xi) = 1 \\ \text{and not} \\ \left(\begin{array}{c} \forall i,j, \deg(f_{i,j}) \le d_{i,j} \le D \text{ and} \\ (f_{i,j})_{i,j}(Q) = v \text{ and} \\ (c_{i,j})_{i,j} = \mathsf{Commit}(\mathsf{ck}, (f_{i,j})_{i,j}, (d_{i,j})_{i,j}) \end{array}\right) \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, D) \\ \hline \text{For } i \in \{1, \dots, \mathsf{r}\}: \\ \rho_i \leftarrow \mathcal{C}(\mathsf{pp}, i) \\ (c_{i,j}, d_{i,j})_{j=1}^{n_i} \leftarrow \mathcal{A}(\mathsf{pp}, (\rho_k)_{k=1}^{i}) \\ (f_{i,i})_{j=1}^{n_i} \leftarrow \mathcal{E}(\mathsf{pp}, (\rho_k)_{k=1}^{i}) \\ \hline Q \leftarrow \mathcal{Q}(\mathsf{pp}, (\rho_k)_{k=1}^{\mathsf{r}}) \\ (v, \mathsf{st}) \leftarrow \mathcal{B}_1(\mathsf{pp}, (\rho_k)_{k=1}^{i}, Q) \\ \xi \xleftarrow{R} \mathbb{F}_{\mathsf{pp}} \\ \pi \leftarrow \mathcal{B}_2(\mathsf{st}, \xi) \\ (\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{Trim}(1^\lambda, (d_{i,j})_{i,j}) \end{array}\right] \le \varepsilon(\lambda).$$

Here $(-)_{i,j}$ denotes $(-)_{1 \le i \le n, 1 \le j \le n_i}$ and similarly for $\forall i, j$; $\mathbb{F}_{\mathsf{pp}} \in \mathcal{F}$ denotes the field described in the public parameters pp; and $(f_{i,j})_{i,j}(Q)$ refers to the tuple of evaluations of the polynomials $f_{i,j}$ at the queried points in Q referring to them.

$\mathsf{PC}_{\mathsf{NH}}$ is said to be **commitment extractable** if $\varepsilon(\lambda)$ is negligible in λ. □

Remark 5. The Marlin PCS (specifically, in its algebraic-group-model variant and without hiding randomness) is commitment extractable in the sense of definition 4. Indeed, in the notation of [Chi+20] lemma B.14, one has that the commitments of the polynomials $\mathbf{p}$ recovered by the extractor $\mathcal{E}_{\mathcal{A}_{\mathsf{alg}}}$ are precisely the $[\langle \mathbf{a}_j, \mathbf{\Sigma}_1 \rangle]_{i=1}^{\mathsf{k}}$ output by $\mathcal{A}_{\mathsf{alg}}$. □

The third and last feature we require is a very natural property that was implicitly used in [Chi+20] but not formally introduced. More specifically, the PCS abstraction therein has a batched Commit method which receives multiple polynomials and degree bounds (and random coins, if hiding) and outputs an equal number of commitments. It is reasonable to expect that the i-th commitment output in a single call to Commit coincides with the result of individually committing to the i-th polynomial with the i-th degree bound. In particular, this (or at least a weaker version of it) is needed to conclude the completeness of the compiled preprocessing argument from that of the polynomial commitment scheme at the end of op. cit. page 42, since the compiler therein performs one call to $\mathsf{PC.Open}$ and $\mathsf{PC.Check}$ with the commitments produced *across all rounds* of interaction (and the single indexing step), and in particular across multiple calls to $\mathsf{PC.Commit}$; yet completeness of the PCS (op. cit. definition 6.1) only refers to the opening of commitments produced in a single call to $\mathsf{PC.Commit}$. Additionally, the compiler makes calls to $\mathsf{PC.Commit}$ with degree bounds $\mathsf{d}^{(0)} = (\mathsf{d}(|\mathbb{i}|, 0, j))_{j=1}^{\mathsf{s}(0)}$ (during indexing) and $\mathsf{d}^{(i)} = (\mathsf{d}(|\mathbb{i}|, i, j))_{j=1}^{\mathsf{s}(i)}$ (during the i-th round of interaction) and a committer key ck produced by $\mathsf{Trim}^{\mathsf{pp}}(1^\lambda, \mathsf{d}^{(0)} \| \cdots \| \mathsf{d}^{(\mathsf{k}(|\mathbb{i}|))})$; yet the aforementioned definition of completeness only refers to commitments produced with the same tuple of degree bounds used to trim the public parameters into a committer key. We address both points with the following property:

Definition 6. Let $\mathsf{PC_{NH}} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ be a (non-hiding) polynomial commitment scheme over a family $\mathcal{F}$ of finite fields as defined in [Chi+20] section 6.1. $\mathsf{PC_{NH}}$ is said to be **separable** if, the following conditions hold:

i) for each

- security parameter $\lambda \in \mathbb{N}$ and maximum degree bound $D \in \mathbb{N}$
- public parameters $\mathsf{pp} \in \mathrm{supp}(\mathsf{PC_{NH}.Setup}(1^\lambda, D))$
- degree-bound tuple $(d_i)_{i=1}^n \in \mathbb{N}^n$ with $\max_{1 \le i \le n'} d_i \le D$
- polynomial tuple $(f_i)_{i=1}^n$ with $\deg(f_i) \le d_i$ for all i

one has

$$\mathsf{PC_{NH}.Commit}(\mathsf{ck}, (f_i)_{i=1}^n, (d_i)_{i=1}^n) = \big(\mathsf{PC_{NH}.Commit}(\mathsf{ck}, (f_i), (d_i))\big)_{i=1}^n \tag{2}$$

where[24] $(\mathsf{ck}, \mathsf{vk}) = \mathsf{PC_{NH}.Trim}^{\mathsf{pp}}(1^\lambda, (d_i)_{i=1}^n)$.

ii) for each λ, D, pp, $(d_i)_{i=1}^n$ and $(f_i)_{i=1}^n$ as above and each

- $(d'_i)_{i=1}^{n'} \in \mathbb{N}^{n'}$ with $\max_{1 \le i \le n'} d'_i \le D$ and $\{d_i\}_{i=1}^n \subseteq \{d'_i\}_{i=1}^{n'}$
- $(d''_i)_{i=1}^{n''} \in \mathbb{N}^{n''}$ with $\max_{1 \le i \le n''} d''_i \le D$ and $\{d_i\}_{i=1}^n \subseteq \{d''_i\}_{i=1}^{n''}$
- commitment tuple $(\tilde{c}_i)_{i=1}^n$, query set Q, purported evaluation tuple v, opening randomness ξ and proof π

one has

$$\mathsf{PC_{NH}.Commit}(\mathsf{ck}', (f_i)_{i=1}^n, (d_i)_{i=1}^n) = \mathsf{PC_{NH}.Commit}(\mathsf{ck}'', (f_i)_{i=1}^n, (d_i)_{i=1}^n), \tag{3}$$

$$\mathsf{PC_{NH}.Open}(\mathsf{ck}', (f_i)_{i=1}^n, (d_i)_{i=1}^n, Q, \xi) = \mathsf{PC_{NH}.Open}(\mathsf{ck}'', (f_i)_{i=1}^n, (d_i)_{i=1}^n, Q, \xi), \tag{4}$$

and

$$\mathsf{PC_{NH}.Check}(\mathsf{vk}', (\tilde{c}_i)_{i=1}^n, (d_i)_{i=1}^n, Q, v, \pi, \xi) = \mathsf{PC_{NH}.Check}(\mathsf{vk}'', (\tilde{c}_i)_{i=1}^n, (d_i)_{i=1}^n, Q, v, \pi, \xi), \tag{5}$$

where $(\mathsf{ck}', \mathsf{vk}') = \mathsf{PC_{NH}.Trim}^{\mathsf{pp}}(1^\lambda, (d'_i)_{i=1}^{n'})$ and $(\mathsf{ck}'', \mathsf{vk}'') = \mathsf{PC_{NH}.Trim}^{\mathsf{pp}}(1^\lambda, (d''_i)_{i=1}^{n''})$. □

Condition i) ensures that the batched commitment method behaves as expected, that is: by outputting the tuple resulting from committing to each of the polynomials separately. The second condition captures the fact that the methods Commit, Open and Check are independent of the specific tuple of degree bounds used to trim pp into ck and vk as long as the latter contained all degree bounds appearing in the call. In particular, repeated application of i) and ii) shows one is free to mix and match commitments produced in different calls to $\mathsf{PC_{NH}.Commit}$.

In the sequel, we only consider separable polynomial commitment schemes. We devote the rest of this subsection to motivating some aspects of definitions 3 and 6 and illustrating some schemes that satisfy them (most notably, the PCS in [Chi+20] appendices B through D).

The definition of a homomorphic PCS is instrumental in making proofs of our (compiled) protocol constant-size. Note that Marlin already uses *some* homomorphic properties of commitments in order to construct batched PC.Open and PC.Check methods. However, the PCS primitive therein does not have a notion of homomorphic commitments and, in particular, the compiled succinct-argument verifier cannot construct linear combinations of commitments (note, however, that this

idea is suggested for the resulting compiled argument in section 9.2 therein). This means in an AHP whose verifier wants to learn the evaluation at a point α of a linear combination of n polynomials, they must query each separate intervening polynomial at α, which results in a communication cost linear in n in the compiled preprocessing argument (as the prover must send all evaluations). By introducing homomorphic commitments at a formal level, the compiled verifier of an eAHP can compute the commitment to the linear combination of the polynomials by themselves, which results in a constant communication cost associated to that query (only one evaluation must be sent). Observe that having the verifier compute such a linear combination likely does not incur extra costs with respect to the multiple-query alternative above, since most reasonable batched $\mathsf{PC.Check}$ methods would also result in the verifier having to compute a linear combination of the commitments (or parts of them). In $\mathsf{vkproof}$, the verifier computes a commitment to a certain fingerprint polynomial by combining commitments associated to each instruction in the program P, which would result in a proof size linear in $|P|$ (the number of instructions in P) without the above trick.

A subtle aspect of definition 3 is the restriction to linear combinations of commitments with the same degree bound. Requiring

$$\mathsf{PC_{NH}.Commit}\left(\mathsf{ck}, \left(\sum_{i=1}^{n} s_i \cdot p_i\right)_{i=1}^{n}, \left(\max_{1\leq i\leq n} d_i\right)_{i=1}^{n}\right) = \sum_{i=1}^{n} s_i \cdot \mathsf{PC_{NH}.Commit}(\mathsf{ck}, (p_i), (d_i))$$

would be too restrictive. For instance, plain KZG does have that property but, crucially, does not enforce degree bounds (other than the maximum bound D). In particular, it does not fit Marlin's PCS primitive, which requires extractability with the claimed degree bounds. It is precisely the introduction of degree-bound-enforcing components that clashes with the above equation. For instance, the PCS in Marlin (cf. appendix C therein), which enforces degree bounds, *is* homomorphic in the sense of definition 3 but does not satisfy the above equation: in the notation of said article,

$$s_1 \cdot \big(p_1(\beta)G, \beta^{D-d_1} \cdot p_1(\beta)G\big) + s_2 \cdot \big(p_2(\beta)G, \beta^{D-d_2} \cdot p_2(\beta)G\big)$$

can in general only be expected to equal

$$\big((s_1 \cdot p_1 + s_2 \cdot p_2)(\beta)G, \beta^{D-\max(d_1,d_2)} \cdot (s_1 \cdot p_1 + s_2 \cdot p_2)(\beta)G\big)$$

if $d_1 = d_2$. A similar situation occurs with the Sonic PCS ([Mal+19] section 6.2), which enforces degree bounds and satisfies (with its slightly different formalism) our notion of homomorphic PCS but not one where degree bounds are allowed to be different for each commitment. Note that, in any case, linearity is only used on trusted commitments (coming from the indexer and referring to public polynomials), where enforcing degree bounds is not essential. One could compile eAHPs with two separate polynomial commitment schemes, one for indexed polynomials and one for instance and prover-sent ones (or a single PCS that allows for "empty" degree bounds). Aside from complicating the formalism, that would prevent commitments to the two types of polynomials from being opened and checked together (unless even more machinery were introduced into the formalism) so we do not pursue that direction further.

Separability is a very mild condition. In particular, it is satisfied by the PCS in Marlin, where prover and verifier keys trimmed to a certain tuple of degree bounds essentially contain those trimmed from the same public parameters with a tuple of degree bounds contained (as sets) in the former. Details can be found in [Chi+20] appendix C.

2.3 Preprocessing arguments for relations with commitments

The existence of commitments to witness polynomials in the instances our relation of interest (capturing the correctness of Varuna verifying keys compiled from programs), which already motivated the introduction of indexed polynomial-oracle relations in subsection 2.1, also requires a minor modification of the "preprocessing argument with universal SRS" primitive from [Chi+20] section 7. We illustrate the issue in the next few lines (which can be safely skipped until definition 7 introducing the updated primitive itself).

Observe first that it is not possible to relegate the commitments to witness polynomials to the *non-oracle part* $\widetilde{\mathbb{x}}$ of the instance $\mathbb{x} = (\mathbb{x}_{\mathsf{o}}, \widetilde{\mathbb{x}})$ of the (uncompiled) indexed polynomial-oracle relation. This would result in the task of PCS openings and checks being deferred to the eAHP itself, yet the eAHP model is too rigid to accommodate the opening and checking procedures of polynomial commitment schemes of interest. Recall that, in that model, all messages from the prover to the verifier are polynomial oracles. Furthermore, placing the aforementioned oracles in $\widetilde{\mathbb{x}}$ would prevent the black-box compilation we propose below to batch the opening of the corresponding commitments with those coming from eAHP-prover messages and indexed polynomials.

A second challenge appears when defining the result of compiling the indexed polynomial-oracle relation with a PCS – which largely means replacing the oracles to witness polynomials by commitments to them. In our PCS formalism, the only relation between polynomials and their commitments is the $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}$ method. Note that in other paradigms, such as LegoSNARK [CFQ19], polynomial commitment schemes come equipped with commitment-verification methods which check whether a given commitment indeed corresponds to a certain polynomial. In our case, when capturing the notion that "the commitments in the instance are indeed commitments to the witness polynomials", one must somehow refer to the committer key ck. Quantifying it with an existential quantifier ("there exists a ck such that $(c_i) = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}, f_i, \ldots)$") has several evident issues. Placing ck in the witness or index of the compiled relation (or even placing its vk counterpart in the instance) gives rise to formal difficulties. For instance, completeness of the preprocessing argument relies on the prover and verifier using a committer key generated by the algorithm $\mathcal{G}$ which could have nothing to do with the one contained in the witness or index (and could even correspond to a different security parameter λ, field, toxic waste, etc.).

The upshot from the above is that the committer key has to be a first-class citizen of the compiled relation separate from the index, witness and verifier; and it has to be related to both the output of $\mathcal{G}$ and the prover's commitment procedure. We capture this in the following definition, where we again highlight changes with [Chi+20] in blue.

Definition 7.

i) Let $\mathsf{PC}_{\mathsf{NH}}$ be a (non-hiding) polynomial commitment scheme. A **indexed relation with commitments over** $\mathsf{PC}_{\mathsf{NH}}$ is a collection $\mathcal{R}$ of quadruples $(\mathbb{i}, \mathsf{ck}, \mathbb{x}, \mathbb{w})$ where each ck has the form of a commitment in $\mathsf{PC}_{\mathsf{NH}}$ and each index $\mathbb{i}$ contains a tuple of degree bounds $(\mathsf{N}_{\mathbb{i},1}, \ldots, \mathsf{N}_{\mathbb{i},d}) \in \mathbb{N}^d$. We denote $\max_{1 \leq j \leq d} \mathsf{N}_{\mathbb{i},j}$ by $\mathsf{N}_{\mathbb{i}}$.

The corresponding language $\mathcal{L}(\mathcal{R})$ is the set of triples $(\mathbb{i}, \mathsf{ck}, \mathbb{x})$ such that $(\mathbb{i}, \mathsf{ck}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ for some $\mathbb{w}$. For $\mathsf{N} \in \mathbb{N}$, we define

$$\mathcal{R}_{\mathsf{N}} = \{(\mathbb{i}, \mathsf{ck}, \mathbb{x}, \mathbb{w}) \in \mathcal{R} : \mathsf{N}_{\mathbb{i}} \leq \mathsf{N}\}$$

with corresponding language $\mathcal{L}(\mathcal{R}_{\mathsf{N}})$.

ii) A preprocessing argument with universal SRS for an indexed relation $\mathcal{R}$ with commitments over $\mathsf{PC}_{\mathsf{NH}}$ is a tuple $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ where

- $\mathcal{G}$, known as the *generator*, is a probabilistic polynomial-time algorithm which, given a security parameter $\lambda \in \mathbb{N}$ and a bound $\mathsf{N} \in \mathbb{N}$, outputs a structured reference string srs that supports indices $\mathbb{i}$ with degree bound $\mathsf{N}_{\mathbb{i}}$ up to N and contains public parameters $\mathsf{pp}_{\mathsf{srs}}$ for $\mathsf{PC}_{\mathsf{NH}}$.
- $\mathcal{I}$, known as the *indexer*, is a deterministic polynomial-time algorithm which, given oracle access to srs and an index $\mathbb{i}$ with degree bound $\mathsf{N}_{\mathbb{i}}$ up to N, outputs an indexed proving key ipk and an indexed verification key ivk.
- $\mathcal{P}$ and $\mathcal{V}$ are probabilistic polynomial-time interactive algorithms known as the *prover* and *verifier*, respectively. The verifier receives oracle access to $\mathbb{i}$ (in addition to ivk).

Such an argument is said to

- be **complete** if, for all security parameters $\lambda \in \mathbb{N}$, bounds $\mathsf{N} \in \mathbb{N}$ and efficient $\mathcal{A}$, one has

$$\Pr\left[\begin{array}{c}(\mathbb{i}, \mathsf{ck}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{N}} \\ \Downarrow \\ \langle \mathcal{P}(\mathsf{ipk}, \mathbb{x}, \mathbb{w}), \mathcal{V}^{\mathbb{i}}(\mathsf{ivk}, \mathbb{x})\rangle = 1\end{array} \middle| \begin{array}{r}\mathsf{srs} \leftarrow \mathcal{G}(1^\lambda, \mathsf{N}) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{srs}) \\ (\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{PC}_{\mathsf{NH}}.\mathsf{Trim}^{\mathsf{pp}_{\mathsf{srs}}}(1^\lambda, (\mathsf{N}_{\mathbb{i},j})_{j=1}^d) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{srs}}(\mathbb{i})\end{array}\right] = 1.$$

- have **soundness error** $\varepsilon = \varepsilon(\lambda)$ if, for all bounds $\mathsf{N} \in \mathbb{N}$ and efficient $(\widetilde{\mathcal{P}}_1, \widetilde{\mathcal{P}}_2)$, one has

$$\Pr\left[\begin{array}{c}(\mathbb{i}, \mathsf{ck}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{N}}) \\ \wedge \\ \langle \widetilde{\mathcal{P}}_2(\mathsf{st}), \mathcal{V}^{\mathbb{i}}(\mathsf{ivk}, \mathbb{x})\rangle = 1\end{array} \middle| \begin{array}{r}\mathsf{srs} \leftarrow \mathcal{G}(1^\lambda, \mathsf{N}) \\ (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \widetilde{\mathcal{P}}_1(\mathsf{srs}) \\ (\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{PC}_{\mathsf{NH}}.\mathsf{Trim}^{\mathsf{pp}_{\mathsf{srs}}}(1^\lambda, (\mathsf{N}_{\mathbb{i},j})_{j=1}^d) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{srs}}(\mathbb{i})\end{array}\right] \leq \varepsilon(\lambda).$$

- **public coin** if every message output by the verifier is a uniformly random string of some prescribed length. □

Making the degree bounds $\mathsf{N}_{\mathbb{i},j}$ (which will ultimately come from their eAHP counterpart) explicit in the definition of indexed relations with commitments allows us to include ck in the relation and ensure it matches the one produced by $\mathcal{I}$ (contained in ipk for our indexers of interest). It also guarantees the SRS produced by $\mathcal{G}(1^\lambda, \mathsf{N})$ can be trimmed to the degree bound $\mathsf{N}_{\mathbb{i}}$ in the index $\mathbb{i}$ of a tuple in $\mathcal{R}_{\mathsf{N}}$. Note that $\mathsf{N}_{\mathbb{i}} \leq \mathsf{N}$ implies $|\mathbb{i}| \leq \mathsf{N}$ (as $\mathbb{i}$ contains a description of $\mathsf{N}_{\mathbb{i}} = \mathsf{N}_{\mathbb{i},j}$ for some j), which was the condition in Marlin[25].

The need to give $\mathcal{V}$ direct oracle access to $\mathbb{i}$ arises from an analogous need in the eAHP verifier (cf. definition 2). In our setting of interest, the index $\mathbb{i}$ contains the various R1CS describing the instructions in the language, which the indexer preprocesses into certain fingerprint polynomials. However, upon reading a program P consisting of a list of instructions, the verifier needs auxiliary

[25] An innocuous condition used in the cited article which was not made explicit is: if $|\mathbb{i}| \leq |\mathbb{i}'|$ then $\max\{\mathsf{d}(|\mathbb{i}|, i, j)\}_{0 \leq i \leq \mathsf{k}(|\mathbb{i}|), 1 \leq j \leq \mathsf{s}(i)} \leq \max\{\mathsf{d}(|\mathbb{i}'|, i, j)\}_{0 \leq i \leq \mathsf{k}(|\mathbb{i}'|), 1 \leq j \leq \mathsf{s}(i)}$. This is implicitly used when claiming completeness due to the trimming performed by the compiled indexer $\mathcal{I}$ in [Chi+20] section 8.1. An analogous condition is not needed in the eAHP setting as our use of the index degree bound $\mathsf{N}_{\mathbb{i}}$ is formally more restrictive.

information such as the dimensions and density of each instruction's R1CS and the location of the instruction's operands within its R1CS. This information is contained in the index $\mathbb{i}$ itself but cannot be read from $\mathbf{I}(\mathbb{i})$, which is only allowed to output polynomials.

2.4 Compilation of eAHPs with polynomial commitment schemes

Intuitively, compiling an eAHP into a succinct argument using a PCS proceeds as in the AHP case with the addition that

i) instance oracles to witness polynomials are replaced with commitments to them[26]. Queries about their evaluations are simply added to the batched opening of the compiled argument system.

ii) whenever the eAHP verifier would query a linear combination $[[f]] = s_1 \cdot [[f_1]] + \ldots + s_n \cdot [[f_n]]$ of oracles to indexed polynomials as above at a point α, the argument verifier instead computes the commitment $c = s_1 \cdot c_1 + \ldots + s_n \cdot c_n$ using the available commitments c_i to the f_i, and requests c to be opened at α (which is batched with the rest of the openings). Note that *this requires all c_i to have been committed to with the same degree bound* by the definition of homomorphic PCS. The prover computes c *as well as* $f = s_1 \cdot f_1 + \ldots + s_n \cdot f_n$ itself in order to be able to open the combined commitment.

In this section, we define the compilation process in detail, starting with the relation that the compiled argument system refers to:

Definition 8. Let $\mathcal{F}$ be a family of finite fields, $\mathcal{R}$ an indexed polynomial-oracle relation over $\mathcal{F}$ and $\mathsf{PC}_{\mathsf{NH}}$ a (non-hiding) polynomial commitment scheme over $\mathcal{F}$. The **commitment to $\mathcal{R}$ under $\mathsf{PC}_{\mathsf{NH}}$** is the indexed relation with commitments

$$\mathcal{R}_{\mathsf{PC}_{\mathsf{NH}}} = \left\{ (\mathbb{i}, \mathsf{ck}, \mathbb{x}, \mathbb{w}) \quad : \begin{array}{l} \mathbb{x} = ((c_i, \mathsf{d}_i)_{i=1}^n, \widetilde{\mathbb{x}}) \\ \mathbb{w} = ((f_i)_{i=1}^n, \widetilde{\mathbb{w}}) \\ (c_i)_{i=1}^n = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}, (f_i)_{i=1}^n, (\mathsf{d}_i)_{i=1}^n) \\ (\mathbb{i}, (([[f_i]], \mathsf{d}_i)_{i=1}^n, \widetilde{\mathbb{x}}), ((f_i)_{i=1}^n, \widetilde{\mathbb{w}})) \in \mathcal{R} \end{array} \right\},$$

where the index degree bound $\mathsf{N}_{\mathbb{i}}$ for $\mathbb{i}$ in $(\mathbb{i}, \mathsf{ck}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{PC}_{\mathsf{NH}}}$ (definition 7) is the same as that for $\mathbb{i}$ in $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ (definition 1). □

An eAHP for $\mathcal{R}$ is compiled into a preprocessing argument for $\mathcal{R}_{\mathsf{PC}_{\mathsf{NH}}}$ as follows (differences with respect to AHP compilation are highlighted in blue):

Definition 9. Consider the following:

- An indexed polynomial-oracle relation $\mathcal{R}$ over a family $\mathcal{F}$ of finite fields (definition 1).
- A (separable, non-hiding) homomorphic polynomial commitment scheme $\mathsf{PC}_{\mathsf{NH}} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ over $\mathcal{F}$ (definitions 3 and 6).

[26] In principle, the commitments in the instance could come from a different PCS than that used for the rest of the compilation, and both could in turn be different from the one used on indexed polynomials as already noted in [Chi+20]. Note that, in that case, the opening and checking procedures of the various commitment schemes could not be batched together. We restrict our attention to the single-PCS scenario for simplicity and efficiency.

- An extended algebraic holographic proof $\mathsf{eAHP} = (\mathsf{k}, \mathsf{s}, \mathsf{d}, \mathbf{I}, \mathbf{P}, \mathbf{V})$ for $\mathcal{R}$ (definition 2) which is public coin and non-adaptive. One can assume without loss of generality that the eAHP is split into the following three phases: interaction, querying and decision.

Then $\mathsf{eAHP}_{\mathsf{PC}_{\mathsf{NH}}} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ is the preprocessing argument with universal SRS for the indexed relation with commitments $\mathcal{R}_{\mathsf{PC}_{\mathsf{NH}}}$ defined as follows:

- $\mathcal{G}$, on input a security parameter $\lambda \in \mathbb{N}$ and bound $\mathsf{N} \in \mathbb{N}$, runs[27] $\mathsf{pp} = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Setup}(1^\lambda, \mathsf{N})$ and outputs $\mathsf{srs} = \mathsf{pp}$. The field $\mathbb{F}_{\mathsf{srs}}$ is simply defined as $\mathbb{F}_{\mathsf{pp}}$, i. e. the field described by the public parameters.

- $\mathcal{I}$, on input an instance[28]

  $\mathbb{i}$ and oracle access to srs,

  - runs $\mathbf{I}(\mathbb{i})$, obtaining polynomials $f_{0,1}, \ldots, f_{0,\mathsf{s}(\mathbb{i},0)} \in \mathbb{F}_{\mathbb{i}}[x]$.
  - trims the SRS to the tuple $(\mathsf{N}_{\mathbb{i},j})_{j=1}^{d} \,\|\, (\mathsf{d}(\mathbb{i},0))$ (that is, the tuple $(\mathsf{N}_{\mathbb{i},j})_{j=1}^{d}$ in the index concatenated with the singleton tuple containing $\mathsf{d}(\mathbb{i},0)$) by computing $(\mathsf{ck}_{\mathbb{i}}, \mathsf{vk}_{\mathbb{i}}) \leftarrow \mathsf{PC}_{\mathsf{NH}}.\mathsf{Trim}^{\mathsf{srs}}(1^\lambda, (\mathsf{N}_{\mathbb{i},j})_{j=1}^{d} \,\|\, (\mathsf{d}(\mathbb{i},0)))$.
  - commits to the indexed polynomials $f_{0,j}$ using the single degree bound $\mathsf{d}(\mathbb{i},0)$ (specified by the eAHP)

    $$(c_{0,j})_{j=1}^{\mathsf{s}(\mathbb{i},0)} = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}_{\mathbb{i}}, (f_{0,j})_{j=1}^{\mathsf{s}(\mathbb{i},0)}, (\mathsf{d}(\mathbb{i},0))_{j=1}^{\mathsf{s}(\mathbb{i},0)}).$$

  - constructs the indexed proving key $\mathsf{ipk} = \big(\mathbb{i}, \mathsf{ck}_{\mathbb{i}}, (f_{0,j})_{j=1}^{\mathsf{s}(\mathbb{i},0)}, (c_{0,j})_{j=1}^{\mathsf{s}(\mathbb{i},0)}\big)$ and the indexed verifying key $\mathsf{ivk} = \big(\mathsf{vk}_{\mathbb{i}}, (c_{0,j})_{j=1}^{\mathsf{s}(\mathbb{i},0)}, \mathsf{msm}\big)$, where msm is some (possibly empty) multi-scalar-multiplication preprocessing information for $(c_{0,j})_{j=1}^{\mathsf{s}(\mathbb{i},0)}$ (with scalars in $\mathbb{F}_{\mathbb{i}}$).
  - outputs $(\mathsf{ipk}, \mathsf{ivk})$.

- $\mathcal{P}(\mathsf{ipk}, \mathbb{x}, \mathbb{w})$ and $\mathcal{V}^{\mathbb{i}}(\mathsf{ivk}, \mathbb{x})$, where

  - ipk and ivk are as output by $\mathcal{I}$ above
  - $\mathbb{w} = ((f_i)_{i=1}^{n}, \widetilde{\mathbb{w}})$
  - $\mathbb{x} = ((c_i, \mathsf{d}_i)_{i=1}^{n}, \widetilde{\mathbb{x}})$ and $\mathbb{x}_{\mathsf{eAHP}}$ is defined as $(([[f_i]], \mathsf{d}_i)_{i=1}^{n}, \widetilde{\mathbb{x}})$

  operate as follows:

  - $\mathcal{P}(\mathsf{ipk}, \mathbb{x}, \mathbb{w})$ runs $\mathbf{P}(\mathbb{i}, \mathbb{x}_{\mathsf{eAHP}}, \mathbb{w})$ and $\mathcal{V}^{\mathbb{i}}(\mathsf{ivk}, \mathbb{x})$ runs $\mathbf{V}^{\mathbb{i}, \mathbf{I}(\mathbb{i})}(\mathbb{x}_{\mathsf{eAHP}})$, interacting throughout $\mathsf{k}(|\mathbb{i}|)$ rounds.

[27] The difference with Marlin being that, now that N refers to an explicit degree bound, $\mathcal{G}$ does not need to deduce any degree bounds from $|\mathbb{i}|$.

[28] In order to conclude completeness, there is an implicit assumption that the field $\mathbb{F}_{\mathsf{pp}}$ defined in the parameters output by $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Setup}$ is the same as the field $\mathbb{F}_{\mathbb{i}}$ of the index $\mathbb{i}$ received by $\mathcal{I}$. This difficulty is already present in Marlin, where indices of the R1CS relation refer to a field (that over which the constraint matrices are defined), and yet the AHP indexer $\mathbf{I}$ receives both such an index and a field $\mathbb{F}$ generated by $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Setup}$, which could be different from the former. There are relatively straightforward ways to circumvent the issue, such as restricting the probabilities in the definition of argument completeness and soundness to the case where $\mathbb{F}_{\mathsf{srs}} = \mathbb{F}_{\mathbb{i}}$. Finer control over the field in the various definitions could also lead to a slightly tighter bound than the one in the soundness proposition 17, where two maxima over the field are taken separately for eAHP and (implicitly) $\mathsf{PC}_{\mathsf{NH}}$ errors and then added when, as the proof shows, the maximum could be taken over the sum of eAHP and PCS errors *over the same field* instead. Since all of those terms should be negligible, this is of limited importance.

In the i-th round of interaction, $\mathbf{V}$ outputs a uniformly random message $\rho_i \in (\mathbb{F}_{\mathbb{i}})^l$ for some l, which $\mathcal{V}$ forwards to $\mathcal{P}$. $\mathcal{P}$ in turn receives it and forwards it to $\mathbf{P}$, which outputs polynomials $f_{i,1}, \ldots, f_{i,\mathsf{s}(\mathbb{i},i)} \in \mathbb{F}_{\mathbb{i}}[x]$ satisfying $\deg(f_{i,j}) \leq \mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, i, j)$.

$\mathcal{P}$ computes $(c_{i,j})_{j=1}^{\mathsf{s}(\mathbb{i},i)} = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}, (f_{i,j})_{j=1}^{\mathsf{s}(\mathbb{i},i)}, (\mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, i, j))_{j=1}^{\mathsf{s}(\mathbb{i},i)})$ and sends the commitments to $\mathcal{V}$ as the reply for the round.

- After the $\mathsf{k}(|\mathbb{i}|)$ rounds of interaction, $\mathbf{V}$ sends $\mathcal{P}$ a uniformly random message $\rho_{\mathsf{k}(|\mathbb{i}|)}$ used to determine the query, initiating the $(\mathsf{k}(|\mathbb{i}|)+1)$-th round of interaction. $\mathcal{P}$ computes $Q = \mathbf{Q}^{\mathbb{i}}(\widetilde{\mathbb{x}}, \rho_1, \ldots, \rho_{\mathsf{k}(|\mathbb{i}|)}, \rho_{\mathsf{k}(|\mathbb{i}|)+1})$ as specified by the (non-adaptive, public-coin) eAHP. Some queries in Q refer to linear combinations $s_1 \cdot [[f_{0,j_1}]] + \ldots + s_m \cdot [[f_{0,j_m}]]$ where the f_{0,j_t} are polynomials output by the indexer (which $\mathcal{P}$ has access to via ipk). For each such query, $\mathcal{P}$ computes the corresponding polynomial $f = \sum_{t=1}^{m} s_t \cdot f_{0,j_t}$.

  Let $(g_j)_{j=1}^{q}$ and $(g'_j)_{j=1}^{q'}$ be the individual polynomials and linear combinations queried in Q, respectively. Note that each g_j either belongs to $\mathbb{w}$ (so its oracle belongs to $\mathbb{x}_{\mathsf{eAHP}}$), is an indexed polynomial $f_{0,j'}$ or is a prover-sent round polynomial $f_{i,j'}$ for some $1 \leq i \leq \mathsf{k}(|\mathbb{i}|)$.

  $\mathcal{P}$ sends $\mathcal{V}$ the tuple v of evaluations of the g_j and g'_j at the query points in Q. This concludes the $(\mathsf{k}(|\mathbb{i}|)+1)$-th round of interaction.
- $\mathcal{V}$ samples and sends a uniformly random opening challenge $\xi = \rho_{\mathsf{k}(|\mathbb{i}|)+2} \in \mathbb{F}_{\mathbb{i}}$ to $\mathcal{P}$, initiating the $(\mathsf{k}(|\mathbb{i}|)+2)$-th and final round of interaction.

  For each g_j, let the degree bound d_j be $\mathsf{d}_{j'}$ if g_j is the j'-th witness polynomial $f_{j'}$ (i. e. $([[f_{j'}]], \mathsf{d}_{j'}) \in \mathbb{x}_{\mathsf{o}} \in \mathbb{x}_{\mathsf{eAHP}}$); the bound $\mathsf{d}(\mathbb{i}, 0)$ (specified by eAHP) if g_j is an indexed polynomial; and $\mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, i, j')$ if g_j is the j'-th round polynomial $f_{i,j'}$.

  $\mathcal{P}$ computes the opening proof

$$\pi \leftarrow \mathsf{PC}_{\mathsf{NH}}.\mathsf{Open}(\mathsf{ck}_{\mathbb{i}}, (g_j)_{j=1}^{q} \,\|\, (g'_j)_{j=1}^{q'}, (d_j)_{j=1}^{q} \,\|\, (\mathsf{d}(\mathbb{i}, 0))_{j=1}^{q'}, \widetilde{Q}, \xi), \tag{6}$$

  where $\widetilde{Q}$ is the same query set as Q but whose (labels of) linear combinations of polynomials have been replaced by the labels of the actual corresponding polynomials g'_j (and queries are reordered so that the those to linear combinations appear last).

  $\mathcal{P}$ sends π to $\mathcal{V}$, concluding the $(\mathsf{k}(|\mathbb{i}|)+2)$-th round of interaction, and therefore the interaction altogether.
- $\mathcal{V}$ computes the same query set $Q = \mathbf{Q}^{\mathbb{i}}(\widetilde{\mathbb{x}}, \rho_1, \ldots, \rho_{\mathsf{k}(|\mathbb{i}|)}, \rho_{\mathsf{k}(|\mathbb{i}|)+1})$ as $\mathcal{P}$. For each query to a linear combination $s_0 \cdot f_{0,j_0} + \ldots + s_m \cdot f_{0,j_m}$ therein, $\mathcal{V}$ constructs $\sum_{t=1}^{m} s_t \cdot c_{0,j_t}$ (by virtue of the homomorphic property of the PCS) using the commitment c_{0,j_t} to each f_{0,j_t} available in ipk as well as the msm information in it, if any. Let g_i and g'_i be as above and c_i and c'_i the commitments to g_i and g'_i, respectively, all of which are available to $\mathcal{V}$ at this point.

  $\mathcal{V}$ runs

$$\mathsf{PC}_{\mathsf{NH}}.\mathsf{Check}(\mathsf{vk}_{\mathbb{i}}, (c_i)_{i=1}^{q} \,\|\, (c'_i)_{i=1}^{q'}, (d_i)_{i=1}^{q} \,\|\, (\mathsf{d}(\mathbb{i}, 0))_{i=1}^{q'}, \widetilde{Q}, v, \pi, \xi), \tag{7}$$

  where d_i and $\widetilde{Q}$ are as in the previous point and v is the tuple of evaluations received from the prover. If the check fails, $\mathcal{V}$ rejects.

  Finally, $\mathcal{V}$ accepts if and only if the decision algorithm of $\mathbf{V}$ does given $\widetilde{\mathbb{x}}$, round randomness $\rho_1, \ldots, \rho_{\mathsf{k}(|\mathbb{i}|)+1}$ and query replies v (as well as oracle access to $\mathbb{i}$).

Throughout the entire process, $\mathcal{V}$ provides $\mathbf{V}$ with oracle access to $\mathbb{i}$ via its own oracle access to it. $\square$

2.5 Completeness and soundness of the compiled argument

We now show that the preprocessing argument with universal SRS compiled as above is complete provided the eAHP and PCS are; and that the argument is sound if the eAHP is sound and the PCS is commitment extractable.

Completeness follows from that of the two components in a straightforward manner. We outline the proof in order to illustrate where the differences with respect to algebraic holographic proofs (eAHP admissibility, separable and homomorphic PCS, relations containing committer keys, etc.) come into play.

Proposition 10. *Let $\mathcal{R}$, $\mathsf{PC}_{\mathsf{NH}}$ and eAHP be as in definition 9. If eAHP and $\mathsf{PC}_{\mathsf{NH}}$ are complete (definition 2 and [Chi+20] definition 6.1), then so is $\mathsf{eAHP}_{\mathsf{PC}_{\mathsf{NH}}}$ (definition 7).*

Proof. Let $\lambda \in \mathbb{N}$ be a security parameter, $\mathsf{N} \in \mathbb{N}$ a size bound and $\mathcal{A}$ an efficient algorithm. Suppose $\mathcal{G}(1^\lambda, \mathsf{N})$, i. e. $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Setup}(1^\lambda, \mathsf{N})$ outputs public parameters $\mathsf{srs} = \mathsf{pp}$ and $\mathcal{A}(\mathsf{pp})$ outputs a triple $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ such that $(\mathbb{i}, \mathsf{ck}, \mathbb{x}, \mathbb{w}) \in (\mathcal{R}_{\mathsf{PC}_{\mathsf{NH}}})_{\mathsf{N}}$, where $(\mathsf{ck}, \mathsf{vk}) = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Trim}^{\mathsf{pp}}(1^\lambda, (\mathsf{N}_{\mathbb{i},j})_{j=1}^d)$. In other words,

- the index degree bound $\mathsf{N}_\mathbb{i} = \max_{1 \le j \le d} \mathsf{N}_{\mathbb{i},j}$ in $\mathbb{i}$ and $\mathsf{d}(\mathbb{i}, 0)$ are both at most N, and the field $\mathbb{F}_\mathbb{i}$ belongs to the family $\mathcal{F}$.
- letting $\mathbb{x} = ((c_i, \mathsf{d}_i)_{i=1}^n, \widetilde{\mathbb{x}})$, $\mathbb{w} = ((f_i)_{i=1}^n, \widetilde{\mathbb{w}})$ and $\mathbb{x}_{\mathsf{eAHP}} = (([[f_i]], \mathsf{d}_i)_{i=1}^n, \widetilde{\mathbb{x}})$, one has $(\mathbb{i}, \mathbb{x}_{\mathsf{eAHP}}, \mathbb{w}) \in \mathcal{R}$ and $(c_i) = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}, (f_i), (\mathsf{d}_i))$ for all i.

The argument indexer $\mathcal{I}$ computes

- $f_{0,1}, \ldots, f_{0,\mathsf{s}(\mathbb{i},0)} \leftarrow \mathbf{I}(i)$
- $(\mathsf{ck}_\mathbb{i}, \mathsf{vk}_\mathbb{i}) = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Trim}^{\mathsf{pp}}(1^\lambda, (\mathsf{N}_{\mathbb{i},j})_{j=1}^d \,\|\, (\mathsf{d}(\mathbb{i}, 0)))$, where all of the degree bounds are lower than the maximum N supported by pp.
- $(c_{0,j})_{j=1}^n = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}_\mathbb{i}, (f_{0,j})_{j=1}^n, (\mathsf{d}(\mathbb{i}, 0))_{j=1}^n)$

and produces ipk and ivk (possibly containing MSM information) as described in the definition of the compiler.

Now $\mathcal{P}(\mathsf{ipk}, \mathbb{x}, \mathbb{w})$ and $\mathcal{V}^\mathbb{i}(\mathsf{ivk}, \mathbb{x})$ run $\mathbf{P}(\mathbb{i}, \mathbb{x}_{\mathsf{eAHP}}, \mathbb{w})$ and $\mathbf{V}^{\mathbb{i}, \mathbf{I}(\mathbb{i})}(\mathbb{x}_{\mathsf{eAHP}})$, respectively, interacting throughout $\mathsf{k}(|\mathbb{i}|)$ rounds at first. After that, $\mathbf{V}$ outputs query randomness $\rho_{\mathsf{k}(|\mathbb{i}|)+1}$ and $\mathcal{P}$ obtains the resulting query set $Q = \mathbf{Q}^\mathbb{i}(\widetilde{\mathbb{x}}, \rho_1, \ldots, \rho_{\mathsf{k}(|\mathbb{i}|)}, \rho_{\mathsf{k}(|\mathbb{i}|)+1})$. When $\mathcal{P}$ computes the tuple v of honest evaluations of witness, prover-sent and indexed polynomials (or linear combinations of the latter) at Q and sends it to $\mathcal{V}$, the interaction up until that point is, from the point of view of $\mathbf{V}$, the same as an interaction with $\mathbf{P}(\mathbb{i}, \mathbb{x}_{\mathsf{eAHP}}, \mathbb{w}) \in \mathcal{R}$ where oracle queries are replied to correctly. By the completeness of eAHP, the first check of $\mathcal{V}$, i. e. $\mathbf{V}$'s decision phase, succeeds.

As for the second check of $\mathcal{V}$, note first that, by the separability of $\mathsf{PC}_{\mathsf{NH}}$, one has

$$(c_i) = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}, (f_i), (\mathsf{d}_i)) = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}_\mathbb{i}, (f_i), (\mathsf{d}_i)). \tag{8}$$

for all i. Furthermore, since $\mathcal{I}$ committed to all indexed polynomials $f_{0,j}$ with the same degree bound $\mathsf{d}(\mathbb{i}, 0)$, for any linear combination $f = s_0 \cdot f_{0,j_0} + \ldots + s_t \cdot f_{0,j_t}$ of them, one has

$$\left(\sum_{i=1}^{t} s_i \cdot c_{0,j_i}\right) = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}\left(\mathsf{ck}_{\mathbb{i}}, \left(\sum_{i=1}^{t} s_i \cdot f_{0,j_i}\right), (\mathsf{d}(\mathbb{i}, 0))\right)$$

by the homomorphic property of $\mathsf{PC}_{\mathsf{NH}}$. Here we are implicitly using $\mathsf{d}(\mathbb{i}, 0) \leq \mathsf{N}_{\mathbb{i}}$ once more.

Finally, when $\mathcal{P}$ receives the opening challenge $\rho_{\mathsf{k}(|\mathbb{i}|)+2} = \xi$ and opens all g_i and g'_i in the query set (in the notation of definition 9) with a single call to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Open}$ as in (6) with the expected degree bounds and honest evaluations (and correct randomness), the resulting proof π makes the call $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Check}$ in (7) succeed, since:

- Each degree bound of a commitments (to polynomials) queried by $\mathcal{V}$ belongs to the tuple of indexed bounds $(\mathsf{N}_{\mathbb{i},j})_j$ or is $\mathsf{d}(\mathbb{i}, 0)$.
- The separability of $\mathsf{PC}_{\mathsf{NH}}$ guarantees that the commitments queried by $\mathcal{V}$ (output by $\mathcal{I}$, contained in $\mathbb{x}$, sent by $\mathcal{P}$ or computed by $\mathcal{V}$ itself as linear combinations of indexed ones) could have been produced with a single call to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}$ with committer key $\mathsf{ck}_{\mathbb{i}}$ and the same polynomials and degree bounds.
- Completeness of $\mathsf{PC}_{\mathsf{NH}}$ implies that a (single) call to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}$ and $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Open}$ results in an accepting proof for $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Check}$, provided the correct keys and polynomial information are used.

■

Our attention now turns to soundness:

Proposition 11. *Let* $\mathcal{R}$, $\mathsf{PC}_{\mathsf{NH}} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ *and* $\mathsf{eAHP} = (\mathsf{k}, \mathsf{s}, \mathsf{d}, \mathbf{I}, \mathbf{P}, \mathbf{V})$ *be as in definition 9. If* $\mathsf{PC}_{\mathsf{NH}}$ *has commitment-extractability error* $\varepsilon_{\mathsf{PC}_{\mathsf{NH}}}(\lambda)$ *(definition 4) and* eAHP *has soundness error* $\varepsilon_{\mathsf{eAHP}}(\mathbb{F})$ *as an eAHP for* $\mathcal{R}$ *(definition 2), then* $\mathsf{eAHP}_{\mathsf{PC}_{\mathsf{NH}}}$ *has soundness error* $\varepsilon_{\mathsf{PC}_{\mathsf{NH}}}(\lambda) + \varepsilon_{\mathsf{eAHP},\mathsf{PC}_{\mathsf{NH}}}(\lambda)$ *as a preprocessing argument with universal SRS for* $\mathcal{R}_{\mathsf{PC}_{\mathsf{NH}}}$ *(definition 7), where*

$$\varepsilon_{\mathsf{eAHP},\mathsf{PC}_{\mathsf{NH}}}(\lambda) = \sup\{\varepsilon_{\mathsf{eAHP}}(\mathbb{F}_{\mathsf{pp}}) : \mathsf{N} \in \mathbb{N}, \mathsf{pp} \in \mathrm{supp}(\mathsf{PC}_{\mathsf{NH}}.\mathsf{Setup}(1^{\lambda}, \mathsf{N}))\}.$$

The idea of the proof is very similar to its AHP counterpart (cf. [Chi+20] section 8.2). In the case of eAHPs, the commitment-extractability adversary and extractor have to be modified to account for the witness oracles and for queries to linear combinations of indexed polynomials. Furthermore, attention has to be paid to the fact that eAHP and $\mathsf{eAHP}_{\mathsf{PC}_{\mathsf{NH}}}$ refer to different relations ($\mathcal{R}$ and $\mathcal{R}_{\mathsf{PC}_{\mathsf{NH}}}$, respectively).

Proof. Fix $\mathsf{N} \in \mathbb{N}$ and let $(\widetilde{\mathcal{P}}_1, \widetilde{\mathcal{P}}_2)$ be an efficient $\mathsf{eAHP}_{\mathsf{PC}_{\mathsf{NH}}}$-soundness adversary as in definition 7. We assume without loss of generality that they are both deterministic (otherwise they can be replaced by their optimal-randomness instantiation) and that the hint st for $\widetilde{\mathcal{P}}_2$ output by $\widetilde{\mathcal{P}}_1(\mathsf{srs})$ (alongside $\mathbb{i}$ and $\mathbb{x}$) contains the public parameters $\mathsf{pp}_{\mathsf{srs}}$ (otherwise, it can be augmented without decreasing the adversary's success probability). Note that $\mathsf{pp}_{\mathsf{srs}}$ and srs are in fact the same by construction of $\mathcal{G}$. Let $\mathbb{F} = \mathbb{F}_{\mathsf{pp}}$.

Consider the following commitment-extractability adversary for $\mathsf{PC}_{\mathsf{NH}}$:

$\underline{\mathcal{A}(\mathsf{pp}, \rho_1, \ldots, \rho_i)}$:

1. Compute $(\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \widetilde{\mathcal{P}}_1(\mathsf{srs})$, where $\mathsf{srs} = \mathsf{pp}$, and parse $\mathbb{x}$ as $((c_i, \mathsf{d}_i)_{i=1}^n, \widetilde{\mathbb{x}})$.
2. If $i = 1$, ignore ρ_1 and compute $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{pp}_{\mathsf{srs}}}(\mathbb{i})$. Then output $(c_{0,j}, \mathsf{d}(\mathbb{i}, 0))_{j=1}^{\mathsf{s}(\mathbb{i},0)}$, where $(c_{0,j})_{j=1}^{\mathsf{s}(\mathbb{i},0)}$ are the commitments contained in ivk to the polynomials indexed by $\mathbf{I}$.
3. If $i = 2$, ignore ρ_2 and output $(c_i, \mathsf{d}_i)_{i=1}^n$.
4. If $3 \leq i \leq \mathsf{k}(|\mathbb{i}|) + 2$, set $i' = i - 2 \in \{1, \ldots, \mathsf{k}(|\mathbb{i}|)\}$. Initialise $\widetilde{\mathcal{P}}_2(\mathsf{st})$ and play the verifier for them for i' rounds, using $\rho_3, \ldots, \rho_i$ as the verifier's messages. Receive the i'-th round reply $(c_{i',j})_{j=1}^{\mathsf{s}(\mathbb{i},i')}$ and output $((c_{i',j})_{j=1}^{\mathsf{s}(\mathbb{i},i')}, \mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, i', j))_{j=1}^{\mathsf{s}(\mathbb{i},i')}$.
5. If $i = \mathsf{k}(|\mathbb{i}|) + 3$, obtain $\mathbf{V}$'s query set $Q = \mathbf{Q}^{\mathbb{i}}(\widetilde{\mathbb{x}}, \rho_3, \ldots, \rho_{\mathsf{k}(|\mathbb{i}|)+3})$. Compute $(c_{0,j})_{j=1}^{\mathsf{s}(\mathbb{i},0)}$ as in the case $i = 1$ and, for each linear combination of (indexed) polynomials queried in Q, compute the analogous combination of commitments $c_{0,j}$ with the same coefficients using the homomorphic property of $\mathsf{PC}_{\mathsf{NH}}$.

Since $\widetilde{\mathcal{P}}_1$, $\widetilde{\mathcal{P}}_2$, $\mathcal{I}$, $\mathbf{Q}$ and d are efficient, as is the computation of linear combinations of commitments (recall definition 3 iii)), so is $\mathcal{A}$.

By the definition of commitment extractability, there exists an efficient extractor $\mathcal{E} = \mathcal{E}_{\mathcal{A}}$ which $\mathcal{A}$ beats at the extraction game with probability at most $\varepsilon_{\mathsf{PC}_{\mathsf{NH}}}(\lambda)$. We consider $\mathcal{E}'$ which, on input $(\mathsf{pp}, \rho_1, \ldots, \rho_i)$, behaves as $\mathcal{E}$ except:

- If $i = 1$, in which case it computes $(\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \widetilde{\mathcal{P}}_1(\mathsf{srs})$ (where $\mathsf{srs} = \mathsf{pp}$) and $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{pp}_{\mathsf{srs}}}(\mathbb{i})$, and outputs the indexed polynomials $(f_{0,j})_{j=1}^{\mathsf{s}(\mathbb{i},0)}$ contained in ipk (which are precisely those output by $\mathbf{I}$).
- If $i = \mathsf{k}(|\mathbb{i}|) + 3$, in which case it[29]
  - Computes $(\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \widetilde{\mathcal{P}}_1(\mathsf{srs})$ and parses $\mathbb{x}$ as $((c_i, \mathsf{d}_i)_{i=1}^n, \widetilde{\mathbb{x}})$.
  - Computes $(f_{0,j})_{j=1}^{\mathsf{s}(\mathbb{i},0)}$ as in the case $i = 1$ above.
  - Deduces the query set $Q = \mathbf{Q}^{\mathbb{i}}(\widetilde{\mathbb{x}}, \rho_3, \ldots, \rho_{\mathsf{k}(|\mathbb{i}|)+3})$ and, for each linear combination of (indexed) polynomials queried in Q, computes the analogous combination of polynomials $f_{0,j}$ with the same coefficients.

The efficiency of $\mathcal{E}'$ follows from that of $\mathcal{E}$ and the other algorithms involved. The probability that $\mathcal{A}$ beats $\mathcal{E}'$ at the extraction game is at most the probability that it beats $\mathcal{E}$, since the commitment-correctness condition in the game is automatically satisfied by the polynomials output by $\mathcal{E}'$ in rounds $i = 1$ and $i = \mathsf{k}(|\mathbb{i}|) + 3$ by construction of $\mathcal{A}$ and the homomorphic property of $\mathsf{PC}_{\mathsf{NH}}$. In particular, $\mathcal{A}$ beats $\mathcal{E}'$ with probability at most $\varepsilon_{\mathsf{PC}_{\mathsf{NH}}}(\lambda)$.

We now define the challenger $\mathcal{C}$, evaluation adversary $\mathcal{B}_1$ and proof adversary $\mathcal{B}_2$ from the commitment-extractability game for $\mathcal{A}$ analogously to the AHP case in [Chi+20] p. 45:

$\underline{\mathcal{C}(\mathsf{pp}, i)}$:

[29] Formally, $\mathcal{E}'$ needs to run $\widetilde{\mathcal{P}}_1(\mathsf{srs})$ to obtain $\mathbb{i}$ and deduce whether this is the case. An interactive version of $\mathcal{E}'$ would simply compute $\mathbb{i}$ as above and use it in case $i = 1$ and to detect the case $i = \mathsf{k}(|\mathbb{i}|) + 3$.

1. Compute[30] $(\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \widetilde{\mathcal{P}}_1(\mathsf{srs})$ and $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{pp}_{\mathsf{srs}}}(\mathbb{i})$.
2. If $i = 0$ or $i = 1$, output empty randomness $\rho_0 = \rho_1 = \bot$.
3. If $3 \leq i \leq \mathsf{k}(|\mathbb{i}|) + 3$, sample $\rho_i \xleftarrow{R} \mathbb{F}^l$, where l is the length of the $(i-2)$-th message of $\mathcal{V}^{\mathbb{i}}(\mathsf{ivk}, \mathbb{x})$.

$\underline{\mathcal{B}_1(\mathsf{pp}, (\rho_i)_{i=1}^{\mathsf{k}(|\mathbb{i}|)+2}, Q)}$:

1. Compute $(\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \widetilde{\mathcal{P}}_1(\mathsf{srs})$.
2. Initialise $\widetilde{\mathcal{P}}_2(\mathsf{st})$ and run it for $\mathsf{k}(|\mathbb{i}|)$ rounds (the full eAHP interaction), playing the argument verifier $\mathcal{V}$ and providing it with $\rho_3, \ldots, \rho_{\mathsf{k}(|\mathbb{i}|)+2}$ as the verifier messages. After receiving its last reply, send to $\widetilde{\mathcal{P}}_2$ the query set Q and receive from it a set v of purported evaluations.
3. Set $\mathsf{st}_{\mathsf{PC}} = \left(\mathsf{st}, (\rho_i)_{i=3}^{\mathsf{k}(|\mathbb{i}|)+2}, Q\right)$ and output $(v, \mathsf{st}_{\mathsf{PC}})$.

Note $\mathcal{B}_1$ does not use ρ_1 or ρ_2.

$\underline{\mathcal{B}_2(\mathsf{st}_{\mathsf{PC}_{\mathsf{NH}}}, \xi)}$:

1. Parse $\mathsf{st}_{\mathsf{PC}_{\mathsf{NH}}}$ as $(\mathsf{st}, (\rho_i)_{i=3}^{\mathsf{k}(|\mathbb{i}|)+2}, Q)$.
2. Initialise $\widetilde{\mathcal{P}}_2(\mathsf{st})$ and replay the interaction with $\mathcal{V}$ as in $\mathcal{B}_1$ above. After receiving the set of replies v (which is in fact the same as in $\mathcal{B}_1$ by the determinism of $\widetilde{\mathcal{P}}_2$), supply to it the opening challenge ξ and receive a proof π. Output π.

The above algorithms are enough to run a full extractability game for $\mathcal{A}$. It remains to construct an admissible eAHP prover $\widetilde{\mathbf{P}}$ which deceives the verifier $\mathbf{V}$ whenever $\mathcal{A}$ does not beat the extractability game. We use the extractor $\mathcal{E}'$ to obtain polynomials for $\widetilde{\mathbf{P}}$ to send:

$\underline{\widetilde{\mathbf{P}}(\mathsf{srs})}$:

1. Compute $(\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \widetilde{\mathcal{P}}_1(\mathsf{srs})$ and parse $\mathbb{x}$ as $((c_i, \mathsf{d}_i)_{i=1}^n, \widetilde{\mathbb{x}})$.
2. Upon receiving the i-th round message ρ_i, from the verifier, store it. Using ρ_i and previouly stored messages, compute $(f_{i,j})_{j=1}^{\mathsf{s}(\mathbb{i},i)} \leftarrow \mathcal{E}'(\mathsf{pp}, \bot, \bot, \rho_1, \ldots, \rho_i)$. If any $f_{i,j}$ has degree greater than $\mathsf{d}(\mathbb{i}, \widetilde{x}, i, j)$, abort. Otherwise, reply with $(f_{i,j})_{j=1}^{\mathsf{s}(\mathbb{i},i)}$.

Note that $\widetilde{\mathcal{P}}$ is admissible and it never outputs the first-, second- or last-round outputs of $\mathcal{E}'$ (i. e. the indexed polynomials, instance polynomials and linear combinations of indexed polynomials, respectively).

Consider now the following probability distribution on the objects involved:

- $\mathsf{pp} \leftarrow \mathsf{PC}_{\mathsf{NH}}.\mathsf{Setup}(1^\lambda, \mathsf{N})$.
- $(\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \widetilde{\mathcal{P}}_1(\mathsf{srs})$, where $\mathsf{srs} = \mathsf{pp}$. Parse $\mathbb{x}$ as $((c_i, \mathsf{d}_i)_{i=1}^n, \widetilde{\mathbb{x}})$.
- For $i \in \{1, \ldots, \mathsf{k}(|\mathbb{i}|) + 3\}$:
  - $\rho_i \leftarrow \mathcal{C}(\mathsf{pp}, i)$.
  - $((c_{i,j})_j, (d_{i,j})_j) \leftarrow \mathcal{A}(\mathsf{pp}, \rho_1, \ldots, \rho_i)$. Recall that, when i is 1, 2 or $\mathsf{k}(|\mathbb{i}|) + 3$, the output consists of the commitments to the indexed polynomials, the commitments in the instance $\mathbb{x}$ and the commitments to linear combinations of indexed polynomials, respec-

[30] This is only used to determine the length of the i-th verifier message. If these are considered to be fixed across instances or even index sizes for the entire eAHP, this step can be replaced by the retrieval of those values.

tively, by construction of $\mathcal{A}$.

- $(f_{i,j})_j \leftarrow \mathcal{E}'(\mathsf{pp}, \rho_1, \ldots, \rho_i)$. For $i \in \{1, \mathsf{k}(|\mathbb{i}|)+3\}$, the output consists of polynomials whose commitments were output by $\mathcal{A}$ in the corresponding round.

- $Q \leftarrow \mathbf{Q}(\widetilde{\mathbb{x}}, \rho_3, \ldots, \rho_{\mathsf{k}(|\mathbb{i}|)+3})$.
- $(v, \mathsf{st}_{\mathsf{PC}}) \leftarrow \mathcal{B}(\mathsf{pp}, \rho_1, \ldots, \rho_{\mathsf{k}(|\mathbb{i}|)+2}, Q)$.
- $\rho_{\mathsf{k}(|\mathbb{i}|)+3} = \xi \xleftarrow{R} \mathbb{F}_{\mathbb{i}}$.
- $\pi \leftarrow \mathcal{B}_2(\mathsf{st}_{\mathsf{PC}_{\mathsf{NH}}}, \xi)$.
- Set $\mathbf{f} = (f_{i,j})_{i,j}$, $\mathbf{c} = (c_{i,j})_{i,j}$ and $\mathbf{d} = (d_{i,j})_{i,j}$, where $i \in \{1, \ldots, \mathsf{k}(|\mathbb{i}|)+3\}$ and
  - For $i = 1$, $j \in \{1, \ldots, \mathsf{s}(\mathbb{i}, 0)\}$.
  - For $i = 2$, $j \in \{1, \ldots, n\}$, where n is the number of commitments in $\mathbb{x}$.
  - For $3 \leq i \leq \mathsf{k}(|\mathbb{i}|)+2$, $j \in \{1, \ldots, \mathsf{s}(\mathbb{i}, i)\}$.
  - For $i = \mathsf{k}(|\mathbb{i}|)+3$, $j \in \{1, \ldots, q'\}$, where q' is the number of queries in Q referring to linear combinations of oracles in Q.

Suppose that $(\mathbb{i}, \mathsf{ck}, \mathbb{x}) \notin \mathcal{L}((\mathcal{R}_{\mathsf{PC}_{\mathsf{NH}}})_{\mathsf{N}})$ and $\mathcal{V}$ accepts in the corresponding interaction with $\widetilde{\mathcal{P}}_2$. Define $\mathbb{x}_{\mathsf{eAHP}} = \left(([[f_{2,j}]], d_{2,j})_{j=1}^n, \widetilde{\mathbb{x}}\right)$ (note that $d_{2,j} = \mathsf{d}_j$ by construction of $\mathcal{A}$). Then at least one of the following holds:

i) $(\mathbb{i}, \mathbb{x}_{\mathsf{eAHP}}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{N}})$.

ii) $(c_j)_{j=1}^n \neq \mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}, (f_{2,j})_{j=1}^n, (d_{2,j})_{j=1}^n)$.

Indeed, arguing by contradiction: if $(\mathbb{i}, \mathbb{x}_{\mathsf{eAHP}}, \mathbb{w}) \in \mathcal{R}_{\mathsf{N}}$ for some $\mathbb{w}$ then $\mathbb{w} = ((f_{2,j})_{j=1}^n, \widetilde{\mathbb{w}})$ for some $\widetilde{\mathbb{w}}$ as $\mathcal{R}$ is a polynomial-oracle relation; in which case, if $(c_j)_{j=1}^n = \mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}(\mathsf{ck}, (f_{2,j})_{j=1}^n, (d_{2,j})_{j=1}^n)$, one has $(\mathbb{i}, \mathsf{ck}, \mathbb{x}, \mathbb{w}) \in (\mathcal{R}_{\mathsf{PC}})_{\mathsf{N}}$ by the definition of $\mathcal{R}_{\mathsf{PC}_{\mathsf{NH}}}$.

In case i), suppose first that the evaluations v supplied by $\mathcal{B}$ (and therefore implicitly by $\widetilde{\mathcal{P}}_2$) are not the correct evaluations of the polynomials extracted by $\mathcal{E}'$ at the query set Q (which we identify with its counterpart $\widetilde{Q}$ as in definition 9, referring to linear combinations as explicit polynomials instead of formal tuples of coefficients), in which case $\mathcal{A}$ wins the commitment-extractability game with $\mathcal{E}'$, $\mathcal{C}$, $\mathcal{B}_1$ and $\mathcal{B}_2$ as above (because $\mathcal{V}$'s decision phase involves $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Check}$ on $\mathbf{c}$ and $\mathbf{d}$). This happens with probability at most $\varepsilon_{\mathsf{PC}_{\mathsf{NH}}}(\lambda)$.

Alternatively, suppose that, still in case i), v matches the evaluations of the extracted polynomials at the queried points. That event contains an execution of the interaction between $\widetilde{\mathbf{P}}$ and $\mathbf{V}$ where the latter receives the correct polynomial evaluations of indexed, witness and round polynomials (including linear combinations of indexed ones) in $\mathbf{f}$ – note at this point that the extractor $\mathcal{E}'$ replaced the polynomials extracted by $\mathcal{E}$ corresponding to indexed ones or linear combinations of them by their honest counterparts. Since $(\mathbb{i}, \mathbb{x}_{\mathsf{eAHP}}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{N}})$ and the decision phase of $\mathcal{V}$ includes that of $\mathbf{V}$, this happens with probability at most $\varepsilon_{\mathsf{eAHP},\mathsf{PC}_{\mathsf{NH}}}(\lambda)$.

In case ii) $\mathcal{A}$ also wins the commitment-extractability game, since $\mathcal{V}$'s call to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Check}$ passes while some of the extracted polynomials do not correspond to the adversarial commitments, which happens with probability at most $\varepsilon_{\mathsf{PC}_{\mathsf{NH}}}(\lambda)$. This concludes the proof. ■

3 Modular compilation of programs

This section offers a detailed description of the modular compiler, which takes as input a program P and produces an R1CS which encapsulates its functionality; and also of the relations between various polynomials describing such an R1CS. The first subsection introduces some notation concerning matrices and the Varuna polynomials associated to them, as well as the notion of fingerprints of polynomials (or, equivalently, of matrices and vectors). In subsection 3.2, we delve into the properties of each individual instruction in the language required for the modular compiler to operate (corresponding to point i) above). Subsection 3.3 describes additional constraints added by the modular compiler to, among other aims, correctly relate the inputs and outputs of the various instructions in the program. In the last subsection, we combine the two previous aspects, introduce the crucial notion of hybrid fingerprints and state the relevant relations between individual-instruction polynomials and their full-matrix counterparts.

We understand a program P as a number N of input-output variables and a list of **commands** (ν, ρ), each of which contains:

i) An **instruction** ν (for example: `add`) which specifies the operation to perform. It is important here to distinguish "programmer-facing" instructions from *concrete* instructions, by which we mean the following: an instruction such as `add` behaves very differently if the operands are field elements in $\mathbb{F}$, 8-bit unsigned integers `u8` or, for instance, points of some elliptic curve. Even though the programmer may be oblivious to the low-level differences between those three manifestations of `add`, the resulting R1CS are very different and should be treated separately.

   Note that a concrete instruction is anything that should be reflected in the R1CS in the form of constraints. For instance, the *declaration* of a variable v of type `bool` may result in the incorporation of a constraint of the form $v \cdot (1 - v) = 0$ in the constraint system and should therefore be treated as an instruction for the purposes of the modular compiler.

   Unless otherwise stated, we shall only be concerned with concrete instructions in the sequel, and abstract away the task of identifying which concrete variant of a programmer-facing instruction is being used to a higher-levelcompiler.

ii) **Operand information** ρ which relates the instruction's operands to other variables in the program or constants. It can, for instance, indicate that the first input to the current command is the output of the command three steps above. This and other types of structural relations are specified in detail later in this subsection.

   Recall that the distinction between inputs and outputs of an instruction is purely semantic in the context of R1CS: both are simply entries in the instance part of the assignment vector (or, equivalently, instance columns in the instruction's R1CS). In other words, what a programmer might write as $v_3 = \mathsf{add}(v_1, v_2)$ may be regarded by the compiler as something like $\mathsf{add}(v_1, v_2, v_3)$. For simplicity, we treat the "output" v_3 as one more operand of `add`.

We fix a finite field $\mathbb{F}$ and denote its group of units by $\mathbb{F}^* = \mathbb{F} \setminus \{0\}$. Let $t_{\max} = v_2(|\mathbb{F}| - 1)$ be the exponent of the maximum power of 2 dividing $|\mathbb{F}^*|$. For each $0 \leq t \leq t_{\max}$, K_{2^i} denotes the unique subgroup of $\mathbb{F}^*$ of order 2^t, for which fix a distinguished generator ω_{2^t} in a compatible way: $\omega_{2^t} = (\omega_{2^{t+1}})^2$. Each K_{2^t} is endowed with the order $(1, \omega_{2^t}, \omega_{2^t}^2, \ldots, \omega_{2^t}^{2^i-1})$. In particular "the j-th

element of K_{2^t}" refers to $\omega_{2^t}^j$.

We operate under the convention that the R1CS assignment vector is $z = x \,\|\, w$, where x is the (public) instance – including the constant 1 placed in the first position – and w is the (private) witness. Therefore, in Varuna (and Marlin), the values of x are interpolated in the first entries $(1, \omega, \ldots, \omega^{|x|-1})$ of the ordered subgroup $K = (1, \omega, \ldots, \omega^{|w|-1})$ of $\mathbb{F}^*$ over which the full assignment vector z is interpolated[31].

Given vectors v and w over $\mathbb{F}$, we denote their concatenation by $v \,\|\, w$ and the n-fold concatenation of v with itself by v^n. In particular, we set $(\alpha)^n = (\underbrace{\alpha, \ldots, \alpha}_{n \text{ times}})$ for $\alpha \in \mathbb{F}$ whenever there is no risk of ambiguity with the value α^n.

If $M \in M_{r\times c}(\mathbb{F})$ and $M' \in M_{r'\times c'}(\mathbb{F})$ are matrices over $\mathbb{F}$, we denote their diagonal concatenation by $M \wedge M' \in M_{r+r'\times c+c'}(\mathbb{F})$, that is, the block matrix which contains M as a submatrix in the top-left corner, M' as a submatrix in the bottom-right corner immediately following M, and zeros elsewhere. This is an associative operator, i. e. $(M \wedge M') \wedge M'' = M \wedge (M' \wedge M'')$, and therefore the notation $M_1 \wedge \cdots \wedge M_n$ is unambiguous.

We also use the shorthand notation $(\alpha)^{r\times c} \in M_{r\times c}(\mathbb{F})$ for the r-by-c matrix with all entries equal to $\alpha \in \mathbb{F}$.

$\mathcal{F}_{\mathsf{key}} = \{\mathsf{row}, \mathsf{col}, \mathsf{rowcol}, \mathsf{rowcolval}\}$ denotes the set of *labels* of polynomials in the Varuna proving key, whereas $\mathcal{F}_{\mathsf{mat}} = \{\mathsf{row}, \mathsf{col}, \mathsf{val}\}$ denotes the set of labels of polynomials directly describing a matrix. These two sets will only be used in expressions of the form $\mathsf{f} \in \mathcal{F}_{\mathsf{key}}$ and $\mathsf{f} \in \mathcal{F}_{\mathsf{mat}}$.

Given two subgroups $H \leq G \leq \mathbb{F}^*$, we define the selector polynomial

$$S_{G,H}(x) = \frac{|H|}{|G|} \cdot \frac{x^{|G|} - 1}{x^{|H|} - 1} \in \mathbb{F}^{=|G|-|H|}[x],$$

the unique polynomial of degree less than $|G|$ which maps all elements of H to 1 and all elements of $G \setminus H$ to 0.

Finally, given $n \in \mathbb{N}$, we denote by $\lceil^2 n\rceil$ the smallest power of two greater than or equal to n.

3.1 Vector and matrix fingerprinting

One of the main tools in the protocol are the fingerprints of vectors associated to matrices:

Definition 12.

i) By the **density** d_M of a matrix $M \in M_{r\times c}(\mathbb{F})$ we mean the number of non-zero (or "dense") entries it contains. Given $t \in \mathbb{N}$, we say that M is **t-admissible** if $t \leq t_{\max}$ and d_M, r and c are all $\leq 2^t$. We say M is **admissible** if it is $t_{\max}$-admissible.

[31] This differs from the convention in the public implementation of Varuna, where x is interpolated over a *subgroup* of K and hence its values appear *interleaved* with those of w. The changes needed to adapt the proposed protocol to that convention are straightforward (simply space out the individual-instruction matrices – without splitting them – in order to accommodate input columns at the correct positions) yet notationally cumbersome. Crucially, they come at essentially no additional cost for the vkproof prover and verifier. We proceed under the convention $z = x \,\|\, w$ for clarity of exposition and stress that this is the only aspect where we differ from Varuna as implemented.

ii) Let $M \in M_{r\times c}(\mathbb{F})$ and let $(r_0, \ldots, r_{d_M-1}) \in \mathbb{N}^{d_M}$ be the list of indices of the rows occupied the non-zero entries of M with the following two conventions:

- The entries are sorted in row-major order[32], that is: from the top row to the bottom one, and from left to right within each row.
- The first row is indexed at $0 \in \mathbb{N}$ (and hence appears as $1 \in \mathbb{F}$ in the row vector defined below).

Consider $t \in \mathbb{N}$ with $t \leq t_{\max}$. The **row vector** of depth t of M is defined as

$$\mathsf{vec}_{\mathsf{row},M,2^t} = (\omega_{2^t}^{r_i})_{i=0}^{d_M-1} \in (\mathbb{F}^*)^{d_M},$$

In other words, if the i-th dense entry of M is in row r_i, then the i-th entry of $\mathsf{vec}_{\mathsf{row},M,2^t}$ is the r_i-th element of K_{2^t}.

The **column vector** $\mathsf{vec}_{\mathsf{col},M,2^t}$ of depth t of M is defined analogously using column indices c_i instead of row ones r_i (and the same conventions, including row-major order), whereas the **value vector** of M is simply

$$\mathsf{vec}_{\mathsf{val},M} = (v_i)_{i=0}^{d_M-1} \in (\mathbb{F}^*)^{d_M}$$

with $v_i \in \mathbb{F}^*$ being the value of the i-th dense entry of M.

iii) Let $M \in M_{r\times c}(\mathbb{F})$ and $d' = \lceil^2 d_M\rceil - d_M$. The **padded row, column and value vectors** of M are defined as

$$\overline{\mathsf{vec}}_{\mathsf{row},M} = \mathsf{vec}_{\mathsf{row},M,\lceil^2 r\rceil} \parallel (1)^{d'} \in (\mathbb{F}^*)^{\lceil^2 d_M\rceil}$$
$$\overline{\mathsf{vec}}_{\mathsf{col},M} = \mathsf{vec}_{\mathsf{col},M,\lceil^2 c\rceil} \parallel (1)^{d'} \in (\mathbb{F}^*)^{\lceil^2 d_M\rceil}$$
$$\overline{\mathsf{vec}}_{\mathsf{val},M} = \mathsf{vec}_{\mathsf{val},M} \parallel (0)^{d'} \in \mathbb{F}^{\lceil^2 d_M\rceil}$$

iv) Let $n \in \mathbb{N}$ and consider a vector $v = (v_i)_{i=0}^{n-1} \in \mathbb{F}^n$. The **fingerprint polynomial of** v is defined as

$$\varphi_v(x) = \sum_{i=0}^{n-1} v_i \cdot x^i \in \mathbb{F}^{<n}[x]$$

(with $\varphi_v(x) = 0$ if $n = 0$). By the **fingerprint of** v **at** $\alpha \in \mathbb{F}$ we simply mean the evaluation $\varphi_v(\alpha) \in \mathbb{F}$.

v) Given $M \in M_{r\times c}(\mathbb{F})$ and $t \leq t_{\max}$, we define the **row fingerprint polynomial** of depth t of M as[33]

$$\varphi_{\mathsf{row},M,2^t}(x) = \varphi_{\mathsf{vec}_{\mathsf{row},M,2^t}}(x) \in \mathbb{F}^{=d_M-1}[x],$$

The column $\varphi_{\mathsf{col},M,2^t}(x) \in \mathbb{F}^{=d_M-1}[x]$ fingerprint polynomial of depth t and the value $\varphi_{\mathsf{val},M}(x) \in \mathbb{F}^{=d_M-1}[x]$ fingerprint polynomial are defined analogously.

Observe that, *as long as it is t-admissible*, M is uniquely characterised by its number of rows and columns together with either of the following:

[32] Other orderings are possible, but row major is particularly convenient when concatenating block matrices diagonally, as will be repeatedly done later on.

[33] Expressions such as $\mathbb{F}^{=n}[x]$ with $n < 0$ or $\mathbb{F}^{<n}[x]$ with $n \leq 0$ simply refer to the set $\{0\}$ consisting of the zero polynomial.

- $\mathsf{vec}_{\mathsf{row},M,2^t}$, $\mathsf{vec}_{\mathsf{col},M,2^t}$ and $\mathsf{vec}_{\mathsf{val},M}$.
- $\varphi_{\mathsf{row},M,2^t}(x)$, $\varphi_{\mathsf{col},M,2^t}(x)$ and $\varphi_{\mathsf{val},M}(x)$.

If M is admissible, it is uniquely determined by ts number of rows and columns together with either of the following:

- $\overline{\mathsf{vec}}_{\mathsf{row},M}$, $\overline{\mathsf{vec}}_{\mathsf{col},M}$ and $\overline{\mathsf{vec}}_{\mathsf{val},M}$. Recall that a Varuna proving key consists of, among other polynomials, interpolators of $\overline{\mathsf{vec}}_{\mathsf{row},M}$ and $\overline{\mathsf{vec}}_{\mathsf{col},M}$ over $K_{\lceil 2d_M \rceil}$ for each $M \in \{A, B, C\}$; and the corresponding Varuna proving key contains commitments to those interpolators.
- $\varphi_{\overline{\mathsf{vec}}_{\mathsf{row},M}}(x)$, $\varphi_{\overline{\mathsf{vec}}_{\mathsf{col},M}}(x)$ and $\varphi_{\overline{\mathsf{vec}}_{\mathsf{val},M}}(x)$.

We list some immediate properties of the newly introduced objects:

Lemma 13.

i) For all $n, m, s \in \mathbb{N}$ and $v \in \mathbb{F}^m$, one has

$$\varphi_{(0)^n \,\|\, v \,\|\, (0)^s}(x) = x^n \cdot \varphi_v(x).$$

ii) Linearity: for all $n \in \mathbb{N}$, $v, w \in \mathbb{F}^n$ and $s, t \in \mathbb{F}$, one has

$$\varphi_{s\cdot v + t\cdot w}(x) = s \cdot \varphi_v(x) + t \cdot \varphi_w(x).$$

iii) For all $M \in M_{r\times c}(\mathbb{F})$, $M' \in M_{r'\times c'}(\mathbb{F})$ and $t \in \mathbb{N}$, one has

$$\varphi_{\mathsf{row},M\wedge M',2^t}(x) = \varphi_{\mathsf{row},M,2^t}(x) + \omega_{2^t}^r \cdot x^{d_M} \cdot \varphi_{\mathsf{row},M',2^t}(x),$$

$$\varphi_{\mathsf{col},M\wedge M',2^t}(x) = \varphi_{\mathsf{col},M,2^t}(x) + \omega_{2^t}^c \cdot x^{d_M} \cdot \varphi_{\mathsf{col},M',2^t}(x)$$

and

$$\varphi_{\mathsf{val},M\wedge M'}(x) = \varphi_{\mathsf{val},M}(x) + x^{d_M} \cdot \varphi_{\mathsf{val},M'}(x).$$

Proof. Properties i) and ii) follow immediately from the definitions. For iii), note that

$$\begin{aligned}\mathsf{vec}_{\mathsf{row},M\wedge M',2^t} &= \mathsf{vec}_{\mathsf{row},M,2^t} \,\|\, \omega_{2^t}^r \cdot \mathsf{vec}_{\mathsf{row},M',2^t} \\ &= \left(\mathsf{vec}_{\mathsf{row},M,2^t} \,\|\, (0)^{d_{M'}}\right) + \left((0)^{d_M} \,\|\, \omega_{2^t}^r \cdot \mathsf{vec}_{\mathsf{row},M',2^t}\right)\end{aligned}$$

due to our row-major convention, and hence

$$\begin{aligned}\varphi_{\mathsf{row},M\wedge M',2^t}(x) &= \varphi_{\mathsf{vec}_{\mathsf{row},M,2^t} \,\|\, (0)^{d_{M'}}}(x) + \varphi_{(0)^{d_M} \,\|\, \omega_{2^t}^r \cdot \mathsf{vec}_{\mathsf{row},M',2^t}}(x) \\ &= \varphi_{\mathsf{vec}_{\mathsf{row},M,2^t}}(x) + \omega_{2^t}^r \cdot x^{d_M} \cdot \varphi_{\mathsf{vec}_{\mathsf{row},M',2^t}}(x) \\ &= \varphi_{\mathsf{row},M,2^t}(x) + \omega_{2^t}^r \cdot x^{d_M} \cdot \varphi_{\mathsf{row},M',2^t}(x),\end{aligned}$$

where the first equality uses property ii), the second one relies on i) and ii) and the third one simply follows the definition of the row fingerprint of a matrix. The identities of $\varphi_{\mathsf{col},M\wedge M'}(x)$ and $\varphi_{\mathsf{val},M\wedge M'}(x)$ result from a similar argument. ■

As will be formalised in the soundness analysis (subsection 5.1), the probability that the fingerprints of two different admissible matrices coincide at a random point α is negligible for large enough $|\mathbb{F}|$. The above lemma allows a vkproof verifier to compute a commitment to a modified version of the fingerprint polynomials of a large part of the R1CS matrices of the compiled program given homomorphic commitments to the fingerprint polynomials of the individual-instruction R1CS matrices, which will be crucial to the protocol. We detail this in subsection 3.2.

3.2 Individual-instruction information

For each instruction ν in the language, the modular compiler requires the following data:

i) Three matrices $A_\nu, B_\nu, C_\nu \in M(\mathbb{F})$ encoding ν into an R1CS. The numbers of rows and columns and the density of these matrices do not need to equal a power of two: this does not affect proving or verifying costs of either Varuna or vkproof. The first column of each of those matrices corresponds[34] to the constant 1.

ii) $d_{\nu,A}, d_{\nu,B}, d_{\nu,C} \in \mathbb{N}$: the density of the matrices A_ν, B_ν and C_ν, respectively. These are the same as d_{A_ν}, d_{B_ν} and d_{C_ν} in the notation of definition 12, but we avoid the latter in order to mitigate subscript and superscript clutter.

iii) $r_\nu, c_\nu \in \mathbb{N} \setminus \{0\}$: the number of rows and columns of A_ν, B_ν and C_ν.

iv) Input-output information $\mathrm{IO}_\nu = (\mathrm{IO}_{\nu,1}, \ldots, \mathrm{IO}_{\nu,t})$, where $\mathrm{IO}_{\nu,i} \in \mathbb{N}$ is the index of the column corresponding to the i-th input-output variable of ν in A_ν, B_ν and C_ν. We denote by $|\mathrm{IO}_\nu|$ the number t of such variables.

For instance, if ν has two inputs and one output located in columns 2, 4 and 7 of the R1CS, respectively, one would have $\mathrm{IO}_\nu = (2, 4, 7) \in \mathbb{N}^3$ and $|\mathrm{IO}_\nu| = 3$.

This gives rise to the following fingerprint polynomials for each $M_\nu \in \{A_\nu, B_\nu, C_\nu\}$:

- For each power of two $\lceil {}^2 r_\nu \rceil \leq 2^t \leq 2^{t_{\max}}$, the row fingerprint polynomial $\varphi_{\mathsf{row},M_\nu,2^t}(x)$ of depth t.
- For each power of two $\lceil {}^2 c_\nu \rceil \leq 2^t \leq 2^{t_{\max}}$, the column fingerprint polynomial $\varphi_{\mathsf{col},M_\nu,2^t}(x)$ of depth t.
- The (single) value fingerprint polynomial $\varphi_{\mathsf{val},M_\nu}(x)$.

This amounts to at most $3 \cdot (2 \cdot t_{\max} + 1)$ polynomials per instruction.

Given a program P containing a list of commands, the modular compiler accumulates each individual-instruction R1CS into a running overall R1CS diagonally. That is, from a program $P = (N, ((\nu_1, \rho_1), \ldots, (\nu_I, \rho_I)))$, it considers the triple

$$(A_{\nu_1} \wedge \cdots \wedge A_{\nu_I},\ B_{\nu_1} \wedge \cdots \wedge B_{\nu_I},\ C_{\nu_1} \wedge \cdots \wedge C_{\nu_I}),$$

[34] It is possible to conceive instructions which do not require a constant-one column, but we consider it to always be present for notational simplicity. In order to incorporate instructions without it, the compiler, prover and verifier would simply omit the addition of the "local-1-equals-global-1" constraint as described in the next subsection. Note that, for the verifier, such a constraint only adds a small number of field operations (and no elliptic curve operations).

which we denote by $(A_{\mathsf{inst}}, B_{\mathsf{inst}}, C_{\mathsf{inst}})$. Those three matrices are then padded with

- $1 + N$ columns of zeros to the left to accommodate the instance, consisting of the public constant 1 and N input-output variables.
- 3 columns of zeros to the right to accommodate the zero-knowledge variables (see the zero-knowledge structural constraint in the next subsection).

The result of this padding can be written as

$$(0)^{0\times(1+N)} \wedge A_{\mathsf{inst}} \wedge (0)^{0\times 3},$$

which we denote by $\overline{A}_{\mathsf{inst}}$, and analogously for $\overline{B}_{\mathsf{inst}}$ and $\overline{C}_{\mathsf{inst}}$.

Such an R1CS loses crucial program-level information, such as relations between operands across commands. New constraints are needed to enforce them, which we cover next.

3.3 Structural constraints

By **structural constraints** we refer to the triples of rows added under A_{inst}, B_{inst} and C_{inst} to enforce a program-level relation of one of four types detailed below. For a program $P = (N, ((\nu_1, \rho_1), \ldots, (\nu_I, \rho_I)))$, let

$$\tilde{r}_i = \sum_{j=1}^{i-1} r_{\nu_j} \in \mathbb{N} \quad \text{and} \quad \tilde{c}_i = \sum_{j=1}^{i-1} c_{\nu_j} \in \mathbb{N} \tag{9}$$

be the cumulative row and column positions of the individual-instruction matrices (with $\tilde{d}_1 = \tilde{r}_1 = \tilde{c}_1 = 0$). In other words, for $M \in \{A, B, C\}$, the first row of the submatrix M_{ν_i} of M_{inst} lives inside the $\tilde{r}_i$-th row of the latter, and similarly for $\tilde{c}_i$. Note that M_{inst} has $\tilde{r}_{I+1}$ rows and $4 + N + \tilde{c}_{I+1}$ columns ($1 + N$ instance variables, $\tilde{c}_{I+1}$ instruction variables and 3 zero-knowledge variables).

We also introduce the following piece of notation only for this subsection: for $0 \leq c < 4+N+\tilde{c}_{I+1}$, $e(c) \in M_{1\times(4+N+\tilde{c}_I)}(\mathbb{F})$ is defined as the row containing the value $1 \in \mathbb{F}$ at column $c \in \mathbb{N}$ and zeros elsewhere.

For the purposes of exposition, we assume the operand portion ρ_i of a command (ν_i, ρ_i) encodes operand information as a list $\rho_i = (\rho_{i,1}, \ldots, \rho_{i,t})$ of $t = |\mathrm{IO}_{\nu_i}|$ elements, each of one of the following two forms:

- $\rho_{i,j} = \mathsf{const}(k)$ if the j-th operand is the constant $k \in \mathbb{F}$.
- $\rho_{i,j} = \mathsf{var}(v)$ if the j-th operand is the variable v, which we simply regard as an identifier and could potentially be marked as a public program input/output variable.

We stress that these operands refer to R1CS variables, not "code" variables. If, for instance, an input to an instruction is of type "elliptic-curve point in projective form" in the high-level code, it will likely[35] correspond to three R1CS operands, which is the relevant representation for us.

[35] Assuming the curve is defined over $\mathbb{F}$ and no extra field elements are used to encode, for instance, the origin point or twisted-representation information.

The four types of structural constraints produced by the modular compiler are as follows:

- Operand equals preexisting program variable: If $\rho_{i,j}$ is of the form[36] $\mathsf{var}(v)$ and v already appeared as an operand in a previous command or is a public program input/output variable, the compiler adds the structural constraint formed by the triple of rows

$$(e(1 + N + \tilde{c}_i + \mathrm{IO}_{\nu_i,j}), e(0), e(c_v))$$

  (corresponding to A, B and C, respectively), where c_v is the column v was assigned in the overall-program R1CS the first time it was encountered. Recall that $\mathrm{IO}_{\nu_i,j}$ is the relative column position of the j-th operand of $\nu_i$inside the individual-instruction R1CS $(A_{\nu_i}, B_{\nu_i}, C_{\nu_i})$ and hence $1 + N + \tilde{c}_i + \mathrm{IO}_{\nu_i,j}$ computes the column position of the same operand in the full-program R1CS. It follows that the above structural constraint enforces equality between the two R1CS variables corresponding to the program variable v.

  If the compiler encounters an operand $\rho_{i,j} = \mathsf{var}(v)$ where v has not appeared in the program before (and is not a public program input/output), no structural constraint is added: instead, the compiler simply stores the information that v is located at column $c_v = 1+N+\tilde{c}_i+\mathrm{IO}_{\nu_i,j}$ for later.

- Operand equals constant: If $\rho_{i,j}$ is of the form $\mathsf{const}(k)$, the compiler simply adds the structural constraint

$$(e(1 + N + \tilde{c}_i + \mathrm{IO}_{\nu_i,j}), e(0), k \cdot e(0)).$$

  This forces the corresponding R1CS variable to equal k times the global constant 1.

- Local 1 equals global 1: As mentioned in subsection 3.2, the first column of each individual-instruction R1CS $(A_{\nu_i}, B_{\nu_i}, C_{\nu_i})$ is reserved for the constant 1. In order to enforce the honesty of this value within the full-program solution vector, one could introduce a new Varuna instance variable, but we advise against this for various performance and formal reasons. The following structural constraint can be added instead:

$$(e(1 + N + \tilde{c}_i), e(0), e(0)).$$

  This enforces equality between the variable at column $e(1 + N + \tilde{c}_i)$ (the local constant 1 of the individual instruction) and that at column 0 (the global constant 1).

- Zero knowledge: As part of its zero-knowledge design, the current Varuna specification and implementation add three private variables zk_a, zk_b and zk_c at the end of the solution vector and forces them to satisfy $\mathsf{zk}_a \cdot \mathsf{zk}_b = \mathsf{zk}_c$ with the constraint

$$(e(1 + N + \tilde{c}_{I+1}), e(2 + N + \tilde{c}_{I+1}), e(3 + N + \tilde{c}_{I+1}))$$

  appended at the end of the otherwise finalised R1CS matrices. Our compiler incorporates it as a final structural constraint[37] as well.

[36] It is trivial to modify the compiler to handle operands of the form $k \cdot v$ where $k \in \mathbb{F}$ is a constant fixed in the program and v is a preexisting program variable: simply replace the third row $e(c_v)$ in the structural constraint by $k \cdot e(c_v)$.

[37] It would be possible to model this not as a structural constraint but rather as a special final program command $\mathsf{mul}(\mathsf{zk}_a, \mathsf{zk}_b, \mathsf{zk}_c)$ for fresh private variables zk_a, zk_b and zk_c. One reason not to do so is our aim of full fidelity to current Varuna specification, which places the corresponding row at the bottom of the matrix and not "sandwiched"

It should be kept in mind that the "zero knowledge" part of "zero-knowledge structural constraint" refers to *Varuna*'s zero-knowledge – our protocol is not (and, as outlined at the start of section 5, it likely would not make sense for it to be) zero-knowledge.

Remark 14. Observe that all three rows of each structural constraint have a density of exactly one except for those corresponding to operands of the form $\mathsf{const}(0)$, where the C row consists of all zeros.

We denote the three matrices formed by the vertical concatenation of the rows of all structural constraints (in the order produced by the compiler as it reads the program) by A_{stru}, B_{stru} and C_{stru}, respectively.

A_{stru}, B_{stru} and C_{stru} are relatively verbose: they each have a number of rows approximately equal to the number n_{op} of operands of all instructions in the program P. However, the verifier's cost to process them is minimal (a number of field operations linear n_{op}) by virtue of the fingerprinting technique.

3.4 Hybrid polynomials and full-program relations

For each $M \in \{A, B, C\}$, the modular compiler transforms the program $P = (N, ((\nu_1, \rho_1), \ldots, (\nu_I, \rho_I)))$ into the full-program R1CS matrix M_{full} as follows: first, $\overline{M}_{\mathsf{inst}}$ and M_{stru} are computed as explained in the previous two subsections and concatenated vertically (recall that $\overline{M}_{\mathsf{inst}}$ is simply $M_{\mathsf{inst}} = M_{\nu_1} \wedge \ldots \wedge M_{\nu_I}$ padded with $1 + N$ columns of zeros to the left and three to the right). Then the resulting matrix is padded with zeros to the right and bottom so that the final number of rows and columns are both powers of two. For the notationally inclined, this can be expressed as

$$\begin{aligned} M_{\mathsf{full}} &= \Big([(0)^{0\times(1+N)} \wedge M_{\nu_1} \wedge \ldots \wedge M_{\nu_I} \wedge (0)^{0\times 3} \wedge (0)^{r_{\mathsf{stru}}\times 0}] + [(0)^{\tilde{r}_{I+1}\times 0} \wedge M_{\mathsf{stru}}] \Big) \wedge (0)^{(r_{\mathsf{full}} - \tilde{r}_{I+1} - r_{\mathsf{stru}})\times(c_{\mathsf{full}} - c_{\mathsf{stru}})} \\ &= \left([\overline{M}_{\mathsf{inst}} \wedge (0)^{r_{\mathsf{stru}}\times 0}] + [(0)^{\tilde{r}_{I+1}\times 0} \wedge M_{\mathsf{stru}}] \right) \wedge (0)^{(r_{\mathsf{full}} - \tilde{r}_{I+1} - r_{\mathsf{stru}})\times(c_{\mathsf{full}} - c_{\mathsf{stru}})}, \end{aligned}$$

where

- $\tilde{r}_{I+1}$ is as in (9), that is: the number of rows of M_{inst} (the sum of the number of rows of each M_{ν_i}).
- r_{stru} is the number of structural constraints, that is: the number of rows of M_{stru}.
- $r_{\mathsf{full}} = \lceil {}^2\tilde{r}_{I+1} + r_{\mathsf{stru}} \rceil$ is the number of rows of M_{full}.
- $c_{\mathsf{stru}} = 4 + N + \tilde{c}_{I+1}$ (with $\tilde{c}_{I+1}$ as in (9)) is the number of columns of $\overline{M}_{\mathsf{inst}}$ and M_{stru}.
- $c_{\mathsf{full}} = \lceil {}^2 c_{\mathsf{stru}} \rceil$ is the number of columns of M_{full}.

In conjunction with lemma 13, this provides some simple relations between the fingerprints of M_{inst}, M_{stru} and M_{full}. Let us focus on the case of A and row, the discussion for other combinations

between M_{inst} and the structural constraints, as would result from the special-command approach. Similarly, adding it as an instruction would cause the protocol to have to operate on its fingerprint polynomials such as $\varphi_{\mathsf{row},A_{\mathsf{mul}},2^t}(x)$ (or commitments to them, in the case of the verifier), which would be more expensive than the negligible field-operation costs incurred by structural constraints. It would also cause a superfluous "local 1 equals global 1" structural constraint to appear.

being analogous. The verifier will have access to the indexed (homomorphic) commitments to the individual-instruction row fingerprint polynomials $\varphi_{\mathsf{row},A_{\nu_i},r_{\mathsf{full}}}(x)$ and will, as part of the protocol, need to learn the evaluation $\varphi_{\mathsf{row},A_{\mathsf{inst}},r_{\mathsf{full}}}(\alpha)$ at a random point $\alpha \in \mathbb{F}$. Lemma 13 iii) shows how to reconstruct $\varphi_{\mathsf{row},A_{\mathsf{inst}}}(x)$ from the multiple $\varphi_{\mathsf{row},A_{\nu_i},r_{\mathsf{full}}}(x)$, but the identity involves a less-than-palatable multiplication by powers of x which becomes particularly challenging in the context of polynomial commitments[38]. For that reason, we introduce a *hybrid* between $\varphi_{\mathsf{row},A_{\mathsf{inst}},r_{\mathsf{full}}}(x)$ and $\varphi_{\mathsf{row},A_{\mathsf{inst}},r_{\mathsf{full}}}(\alpha)$, which can be regarded as a partially evaluated fingerprint polynomial:

Definition 15. Let $\alpha \in \mathbb{F}$. Consider $n \in \mathbb{N} \setminus \{0\}$, $t \leq t_{\max}$ and a family $\mathcal{M} = (M_i)_{i=1}^n$ with each $M_i \in M_{r_i \times c_i}(\mathbb{F})$ of density d_i. The **hybrid row fingerprint polynomial** of $\mathcal{M}$ of depth t with respect to α is defined as

$$\widetilde{\varphi}_{\mathsf{row},\mathcal{M},2^t,\alpha}(x) = \sum_{i=1}^{n} \omega_{2^t}^{\tilde{r}_i} \cdot \alpha^{\tilde{d}_i} \cdot \varphi_{\mathsf{row},M_i,2^t}(x), \tag{10$_{\mathsf{row}}$}$$

where

$$\tilde{d}_i = \sum_{j=1}^{i-1} d_j \quad \text{and} \quad \tilde{r}_i = \sum_{j=1}^{i-1} r_j$$

are the cumulative densities and row positions of the individual matrices (with $\tilde{d}_1 = \tilde{r}_1 = 0$). We also define column and value hybrids

$$\widetilde{\varphi}_{\mathsf{col},\mathcal{M},2^t,\alpha}(x) = \sum_{i=1}^{n} \omega_{2^t}^{\tilde{c}_i} \cdot \alpha^{\tilde{d}_i} \cdot \varphi_{\mathsf{col},M_i,2^t}(x) \tag{10$_{\mathsf{col}}$}$$

and

$$\widetilde{\varphi}_{\mathsf{val},\mathcal{M},\alpha}(x) = \sum_{i=1}^{n} \alpha^{\tilde{d}_i} \cdot \varphi_{\mathsf{val},M_i}(x), \tag{10$_{\mathsf{val}}$}$$

where $\tilde{c}_i$ is defined analogously to $\tilde{r}_i$ □

The crucial property of these objects is that they satisfy

$$\widetilde{\varphi}_{\mathsf{row},\mathcal{M},2^t,\alpha}(\alpha) = \varphi_{\mathsf{row},M_1 \wedge \ldots \wedge M_n,2^t}(\alpha) \tag{11}$$

and analogously for col and val, as induction on lemma 13 iii) shows.

We note that, while the evaluation $\widetilde{\varphi}_{\mathsf{row},\mathcal{M},\alpha}(\alpha)$ depends only on $M_1 \wedge \ldots \wedge M_n$ (as shown by the above equality), the same is not true of the hybrid polynomial $\widetilde{\varphi}_{\mathsf{row},\mathcal{M},2^t,\alpha}(x)$, which does depend on the family $\mathcal{M}$ itself – this is the reason $\mathcal{M}$ needs to be reflected in the notation. For instance, consider two matrices M_1 and M_2 and the two families $\mathcal{M} = (M_1 \wedge M_2)$ and $\mathcal{M}' = (M_1, M_2)$. Then φ_{row} of the diagonal concatenation is the same in both families, yet $\widetilde{\varphi}_{\mathsf{row},\mathcal{M},2^t,\alpha}(x) \neq \widetilde{\varphi}_{\mathsf{row},\mathcal{M}',2^t,\alpha}(x)$ in general: the latter has a lower degree (this will also result in improved prover costs later on). Broadly speaking, for a family $\mathcal{M} = (M_i)_{i=1}^n$ as above, the fingerprint polynomials $\varphi_{\mathsf{row},M_1 \wedge \cdots \wedge M_n,2^t}(x)$, $\varphi_{\mathsf{col},M_1 \wedge \cdots \wedge M_n,2^t}(x)$ and $\varphi_{\mathsf{val},M_1 \wedge \cdots \wedge M_n}(x)$ have degree approximately equal to $\tilde{d}_{n+1} = \sum_{j=1}^n d_j$, whereas their hybrid counterparts $\widetilde{\varphi}_{\mathsf{row},\mathcal{M},2^t,\alpha}(x)$, $\widetilde{\varphi}_{\mathsf{col},\mathcal{M},2^t,\alpha}(x)$ and $\widetilde{\varphi}_{\mathsf{val},\mathcal{M},\alpha}(x)$ have degree approximately equal to $\max_{i=1}^n d_i$.

[38] The fact that a commitment to $x \cdot f(x)$ should not be feasibly computable from one to $f(x)$ is part of the degree-bound guarantees in schemes such as KZG.

All that remains before describing the protocol operation is to do some bookkeeping in order to account for how the decomposition of the overall-program matrices into instruction-related and structural-constraint-related parts is reflected in the fingerprint polynomials.

Consider a program $P = (N, ((\nu_1, \rho_1), \ldots, (\nu_I, \rho_I)))$ as above and let $d_{\mathsf{inst},M} = \sum_{i=1}^{I} d_{\nu_i,M}$ and $d_{\mathsf{stru},M}$ be the total densities of M_{inst} and M_{stru}, respectively, for each $M \in \{A, B, C\}$. By remark 14, one has $d_{\mathsf{stru},A} = d_{\mathsf{stru},B}$, which we may write as $d_{\mathsf{stru},AB}$. Let also r_{inst} and $r_{\mathsf{stru}} = d_{\mathsf{stru},AB}$ be the number of rows of M_{inst} (or $\overline{M}_{\mathsf{inst}}$) and M_{stru}, respectively, for any M. For $M \in \{A, B, C\}$, we denote by $\mathcal{M} \in \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$ the family of individual-instruction matrices $(M_{\nu_1}, \ldots, M_{\nu_I})$ and by $\overline{\mathcal{M}} = ((0)^{0\times(1+N)}) \,\|\, \mathcal{M} \,\|\, ((0)^{0\times 3}))$ the same family $\mathcal{M}$ with an extra 0-by-$(1+N)$ matrix at the start and a 0-by-3 one at the end. This is simply to account for the columns of zeros that pad M_{inst} to form $\overline{M}_{\mathsf{inst}}$: note that

$$\bigwedge_{M_i \in \mathcal{M}} = M_{\mathsf{inst}} \quad \text{and} \quad \bigwedge_{M_i \in \overline{\mathcal{M}}} = \overline{M}_{\mathsf{inst}}.$$

Before delving into fingerprint relations, we extend the notion of admissibility from matrices to programs the natural way: a program P as above is said to be t-admissible if each matrix in the R1CS $(A_{\mathsf{full}}, B_{\mathsf{full}}, C_{\mathsf{full}})$ produced by the modular compiler on input P is 2^t-admissible (in the sense of definition 12) – that is, if r_{full}, c_{full}, $d_{A_{\mathsf{full}}}$, $d_{B_{\mathsf{full}}}$ and $d_{C_{\mathsf{full}}}$ are all $\leq 2^t$. Recall that only then are the vectors $\overline{\mathsf{vec}}_{\mathsf{row},M_{\mathsf{full}}}$ and $\overline{\mathsf{vec}}_{\mathsf{col},M_{\mathsf{full}}}$ interpolated by the Varuna polynomials well defined. We say P is admissible if it is $t_{\max}$-admissible.

Assume for the rest of this subsection that P is indeed admissible. We discuss row, column and value objects separately:

- **Row objects**: One has

$$\mathsf{vec}_{\mathsf{row},M_{\mathsf{full}},r_{\mathsf{full}}} = \mathsf{vec}_{\mathsf{row},\overline{M}_{\mathsf{inst}},r_{\mathsf{full}}} \,\|\, \omega_{r_{\mathsf{full}}}^{r_{\mathsf{inst}}} \cdot \mathsf{vec}_{\mathsf{row},M_{\mathsf{stru}},r_{\mathsf{full}}}$$

  in the notation of definition 12, and hence

$$\varphi_{\mathsf{row},M_{\mathsf{full}},r_{\mathsf{full}}}(x) = \varphi_{\mathsf{row},M_{\mathsf{inst}},r_{\mathsf{full}}}(x) + \omega_{r_{\mathsf{full}}}^{r_{\mathsf{inst}}} \cdot x^{d_{\mathsf{inst},M}} \cdot \varphi_{\mathsf{row},M_{\mathsf{stru}},r_{\mathsf{full}}}(x)$$

  by lemma 13 (note that $\varphi_{\mathsf{row},\overline{M}_{\mathsf{inst}},r_{\mathsf{full}}}(x) = \varphi_{\mathsf{row},M_{\mathsf{inst}},r_{\mathsf{full}}}(x)$). In particular, for any $\alpha \in \mathbb{F}$,

$$\begin{aligned}\varphi_{\mathsf{row},M_{\mathsf{full}},r_{\mathsf{full}}}(\alpha) &= \varphi_{\mathsf{row},M_{\mathsf{inst}},r_{\mathsf{full}}}(\alpha) + \omega_{r_{\mathsf{full}}}^{r_{\mathsf{inst}}} \cdot \alpha^{d_{\mathsf{inst},M}} \cdot \varphi_{\mathsf{row},M_{\mathsf{stru}},r_{\mathsf{full}}}(\alpha) \\ &= \widetilde{\varphi}_{\mathsf{row},\mathcal{M},r_{\mathsf{full}},\alpha}(\alpha) + \omega_{r_{\mathsf{full}}}^{r_{\mathsf{inst}}} \cdot \alpha^{d_{\mathsf{inst},M}} \cdot \varphi_{\mathsf{row},M_{\mathsf{stru}},r_{\mathsf{full}}}(\alpha)\end{aligned}$$

  holds by (11).

  Recall that Varuna verification keys contain interpolators not of $\mathsf{vec}_{\mathsf{row},M_{\mathsf{full}},r_{\mathsf{full}}}$ for $M \in \{A, B, C\}$, but rather of the vector $\overline{\mathsf{vec}}_{\mathsf{row},M_{\mathsf{full}},r_{\mathsf{full}}}$ which is padded with ones to length $\lceil {}^2 d_{M_{\mathsf{full}}} \rceil$. By the previous displayed equation, one has

$$\varphi_{\overline{\mathsf{vec}}_{\mathsf{row},M_{\mathsf{full}}}}(\alpha) = \widetilde{\varphi}_{\mathsf{row},\mathcal{M},r_{\mathsf{full}},\alpha}(\alpha) + \omega_{r_{\mathsf{full}}}^{r_{\mathsf{inst}}} \cdot \alpha^{d_{\mathsf{inst},M}} \cdot \varphi_{\mathsf{row},M_{\mathsf{stru}},r_{\mathsf{full}}}(\alpha) + \sum_{i=d_{M_{\mathsf{full}}}}^{\lceil {}^2 d_{M_{\mathsf{full}}} \rceil - 1} \alpha^i \qquad (12_{\mathsf{row}})$$

  where the last term can be computed as $(\alpha^{\lceil {}^2 d_{M_{\mathsf{full}}} \rceil} - \alpha^{d_{M_{\mathsf{full}}}})/(\alpha - 1)$ for $\alpha \neq 1$.

- **Column objects**: One has

$$\mathsf{vec}_{\mathsf{col},M_{\mathsf{full}},c_{\mathsf{full}}} = \mathsf{vec}_{\mathsf{col},\overline{M}_{\mathsf{inst}},c_{\mathsf{full}}} \,\|\, \mathsf{vec}_{\mathsf{col},M_{\mathsf{stru}},c_{\mathsf{full}}} = (\omega_{c_{\mathsf{full}}}^{1+N} \cdot \mathsf{vec}_{\mathsf{col},M_{\mathsf{inst}},c_{\mathsf{full}}}) \,\|\, \mathsf{vec}_{\mathsf{col},M_{\mathsf{stru}},c_{\mathsf{full}}}$$

and therefore

$$\varphi_{\mathsf{col},M_{\mathsf{full}},c_{\mathsf{full}}}(x) = \omega_{c_{\mathsf{full}}}^{1+N} \cdot \varphi_{\mathsf{col},M_{\mathsf{inst}},c_{\mathsf{full}}}(x) + x^{d_{\mathsf{inst},M}} \cdot \varphi_{\mathsf{col},M_{\mathsf{stru}},c_{\mathsf{full}}}(x).$$

In particular, for any $\alpha \in \mathbb{F}$,

$$\begin{aligned}\varphi_{\mathsf{col},M_{\mathsf{full}},c_{\mathsf{full}}}(\alpha) &= \omega_{c_{\mathsf{full}}}^{1+N} \cdot \varphi_{\mathsf{col},M_{\mathsf{inst}},c_{\mathsf{full}}}(\alpha) + \alpha^{d_{\mathsf{inst},M}} \cdot \varphi_{\mathsf{col},M_{\mathsf{stru}},c_{\mathsf{full}}}(\alpha) \\ &= \omega_{c_{\mathsf{full}}}^{1+N} \cdot \widetilde{\varphi}_{\mathsf{col},\mathcal{M},c_{\mathsf{full}},\alpha}(\alpha) + \alpha^{d_{\mathsf{inst},M}} \cdot \varphi_{\mathsf{col},M_{\mathsf{stru}},c_{\mathsf{full}}}(\alpha)\end{aligned}$$

holds. Just like in the case of row objects, we incorporate the density padding present in Varuna-key polynomials:

$$\varphi_{\overline{\mathsf{vec}}_{\mathsf{col},M_{\mathsf{full}}}}(\alpha) = \omega_{c_{\mathsf{full}}}^{1+N} \cdot \widetilde{\varphi}_{\mathsf{col},\mathcal{M},c_{\mathsf{full}},\alpha}(\alpha) + \alpha^{d_{\mathsf{inst},M}} \cdot \varphi_{\mathsf{col},M_{\mathsf{stru}},c_{\mathsf{full}}}(\alpha) + \sum_{i=d_{M_{\mathsf{full}}}}^{\lceil 2 d_{M_{\mathsf{full}}} \rceil - 1} \alpha^i. \qquad (12_{\mathsf{col}})$$

- **Value objects** are comparatively simple due to their invariance under padding: one has

$$\mathsf{vec}_{\mathsf{val},M_{\mathsf{full}}} = \mathsf{vec}_{\mathsf{val},\overline{M}_{\mathsf{inst}}} \,\|\, \mathsf{vec}_{\mathsf{val},M_{\mathsf{stru}}}$$

and

$$\varphi_{\mathsf{val},M_{\mathsf{full}}}(x) = \varphi_{\mathsf{val},\overline{M}_{\mathsf{inst}}}(x) + x^{d_{\mathsf{inst},M}} \cdot \varphi_{\mathsf{val},M_{\mathsf{stru}}}(x),$$

where again $\varphi_{\mathsf{val},\overline{M}_{\mathsf{inst}}}(x) = \varphi_{\mathsf{val},M_{\mathsf{inst}}}(x)$. In particular, for any $\alpha \in \mathbb{F}$,

$$\begin{aligned}\varphi_{\mathsf{val},M_{\mathsf{full}}}(\alpha) &= \varphi_{\mathsf{val},M_{\mathsf{inst}}}(\alpha) + \alpha^{d_{\mathsf{inst},M}} \cdot \varphi_{\mathsf{val},M_{\mathsf{stru}}}(\alpha) \\ &= \widetilde{\varphi}_{\mathsf{val},\mathcal{M},\alpha}(\alpha) + \alpha^{d_{\mathsf{inst},M}} \cdot \varphi_{\mathsf{val},M_{\mathsf{stru}}}(\alpha)\end{aligned}$$

holds. Since $\overline{\mathsf{vec}}_{\mathsf{val},M_{\mathsf{full}}}$ is padded with zeros, its fingerprint coincides with that of $\mathsf{vec}_{\mathsf{val},M_{\mathsf{full}}}$ and hence

$$\varphi_{\overline{\mathsf{vec}}_{\mathsf{val},M_{\mathsf{full}}}}(\alpha) = \widetilde{\varphi}_{\mathsf{val},\mathcal{M},\alpha}(\alpha) + \alpha^{d_{\mathsf{inst},M}} \cdot \varphi_{\mathsf{val},M_{\mathsf{stru}}}(\alpha) \qquad (12_{\mathsf{val}})$$

Remark 16. It may seem as though, in order to verify (12_{row}) - (12_{val}), one would need to compute $\varphi_{\mathsf{row},M_{\mathsf{stru}},r_{\mathsf{full}}}(\alpha), \varphi_{\mathsf{col},M_{\mathsf{stru}},c_{\mathsf{full}}}(\alpha)$ and $\varphi_{\mathsf{val},M_{\mathsf{stru}}}(\alpha)$ for each $M \in \{A, B, C\}$. However, these objects satisfy some trivial relations which reduce the amount of computation required:

i) The vectors $\mathsf{vec}_{\mathsf{row},A_{\mathsf{stru}},r_{\mathsf{full}}}$ and $\mathsf{vec}_{\mathsf{row},B_{\mathsf{stru}},r_{\mathsf{full}}}$ coincide: they are simply $(1, \omega_{r_{\mathsf{full}}}, \ldots, \omega_{r_{\mathsf{full}}}^{d_{\mathsf{stru},AB}-1})$. One therefore has

$$\varphi_{\mathsf{row},A_{\mathsf{stru}},r_{\mathsf{full}}}(\alpha) = \varphi_{\mathsf{row},B_{\mathsf{stru}},r_{\mathsf{full}}}(\alpha) = \sum_{i=0}^{d_{\mathsf{stru},AB}-1} (\omega_{r_{\mathsf{full}}} \cdot \alpha)^i.$$

An analogous equality to the above does not hold for C due to program operands of the form $\mathsf{const}(0)$ introducing all-zero rows into C_{stru}.

ii) The vectors $\mathsf{vec}_{\mathsf{val},A_{\mathsf{stru}}}$ and $\mathsf{vec}_{\mathsf{val},B_{\mathsf{stru}}}$ also coincide: they are the constant vector $(1)^{d_{\mathsf{stru},AB}}$. In particular, one has

$$\varphi_{\mathsf{val},A_{\mathsf{stru}}}(\alpha) = \varphi_{\mathsf{val},B_{\mathsf{stru}}}(\alpha) = \sum_{i=0}^{d_{\mathsf{stru},AB}-1} \alpha^i,$$

which equals $= (\alpha^{d_{\mathsf{stru},AB}} - 1)/(\alpha - 1)$ for $\alpha \neq 1$. This has a negligible computational cost as $\alpha^{d_{\mathsf{stru},AB}}$ and the inverse $(\alpha - 1)^{-1}$ (when $\alpha \neq 1$) need to be computed in other parts of the protocol regardless. Operands of the form $\mathsf{const}(k)$ prevent the same relation from holding for C.

iii) The vector $\mathsf{vec}_{\mathsf{col},B_{\mathsf{stru}},c_{\mathsf{full}}}$ equals $(1)^{d_{\mathsf{stru},AB}-1} \,\|\, (\omega_{c_{\mathsf{full}}}^{2+N+\tilde{c}_{I+1}})$, since all structural constraints place their single non-zero B entry at column 0 except for the zero-knowledge constraint (cf. the previous subsection). Hence

$$\varphi_{\mathsf{col},B_{\mathsf{stru}}}(\alpha) = \left(\sum_{i=0}^{d_{\mathsf{stru},AB}-2} \alpha^i \right) + \omega_{c_{\mathsf{full}}}^{2+N+\tilde{c}_{I+1}} \cdot \alpha^{d_{\mathsf{stru},AB}-1}.$$

All in all, only four (meaningful) computations are needed to cover all seemingly nine structural parts of the $\overline{\mathsf{vec}}$ fingerprints in equations (12_{row}) - (12_{val}) for $M \in \{A, B, C\}$: $\varphi_{\mathsf{row},C_{\mathsf{stru}}}$, $\varphi_{\mathsf{col},A_{\mathsf{stru}}}$, $\varphi_{\mathsf{col},C_{\mathsf{stru}}}$ and $\varphi_{\mathsf{val},C_{\mathsf{stru}}}$.

4 Protocol

We first present the protocol $\mathsf{eAHP}_{\mathsf{vkproof}}$ as an eAHP for the following indexed polynomial-oracle relation $\mathcal{R}_{\mathsf{VarunaVK}}$:

- An index $\mathbb{i}$ contains the description of a finite field $\mathbb{F}_{\mathbb{i}}$, a maximum depth $t_{\mathbb{i}}$ and a list of instruction information $\mathbb{i}_{\nu_1}, \ldots, \mathbb{i}_{\nu_n}$, where $\nu_1, \ldots, \nu_n$ are distinct instructions over $\mathbb{F}_{\mathbb{i}}$ and each $\mathbb{i}_{\nu_i}$ contains ν_i's individual-instruction parameters described in points i) - iv) from the beginning of subsection 3.2. For each ν_i, the number r_{ν_i} and c_{ν_i} of rows and columns, as well as the densities $d_{A_{\nu_i}}$, $d_{B_{\nu_i}}$ and $d_{C_{\nu_i}}$ must be at most $2^{t_{\mathbb{i}}}$.

  The index degree bounds $\mathsf{N}_{\mathbb{i},j}$ are the elements of the set

  $$\{2^i - 1, 2^i - 2 : 1 \leq i \leq t_{\mathbb{i}}\},$$

  and therefore $\mathsf{N}_{\mathbb{i}} = 2^{t_{\mathbb{i}}} - 1$.

- An instance $\mathbb{x} = (([[\mathsf{f}_M]], \mathsf{d}_{\mathsf{f},M})_{\mathsf{f} \in \mathcal{F}_{\mathsf{key}}, M \in \{A,B,C\}}, \widetilde{\mathbb{x}} = P)$ contains oracle access to the twelve witness polynomials below (which, after compilation with a PCS, result in a Varuna verifying key). Its non-oracle part $\widetilde{\mathbb{x}}$ is a program $P = (N, ((\nu_1, \rho_1), \ldots, (\nu_I, \rho_I)))$ as in subsection 3.4.

- A witness $\mathbb{w} = ((\mathsf{f}_M)_{\mathsf{f} \in \mathcal{F}_{\mathsf{key}}, M \in \{A,B,C\}}, \widetilde{\mathbb{w}} = \bot)$ is a Varuna proving key – that is, the twelve polynomials

$$\mathsf{row}_A(x), \mathsf{col}_A(x), \mathsf{rowcol}_A(x), \mathsf{rowcolval}_A(x),$$
$$\mathsf{row}_B(x), \mathsf{col}_B(x), \mathsf{rowcol}_B(x), \mathsf{rowcolval}_B(x),$$
$$\mathsf{row}_C(x), \mathsf{col}_C(x), \mathsf{rowcol}_C(x) \text{ and } \mathsf{rowcolval}_C(x).$$

The non-oracle part $\widetilde{\mathbb{w}}$ of the witness is empty. Recall that a Marlin proving key is almost the same, but contains val_M instead of rowcol_M and $\mathsf{rowcolval}_M$ for $M \in \{A, B, C\}$.

- A triple $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ belongs to $\mathcal{R}_{\mathsf{VarunaVK}}$ if and only if
  - $\mathbb{F}_{\mathbb{i}}$ is large enough to accommodate depth $t_{\mathbb{i}}$, that is: $2^{t_{\max}} \mid |\mathbb{F}_{\mathbb{i}}| - 1$ (equivalently, $t_{\mathbb{i}} \leq t_{\max}$ where $t_{\max}$ is the maximum depth of $\mathbb{F}_{\mathbb{i}}$).
  - All instructions in P appear in $\mathbb{i}$.
  - P is $t_{\mathbb{i}}$-admissible as defined in subsection 3.4 when compiled with the modular compiler from section 3 (in other words, the densities and matrix sizes of the compiled R1CS are at most $2^{t_{\mathbb{i}}}$).
  - P is well formed[39] as described in section 3 – for instance, each operand information ρ_i for an instruction ν_i has $|\mathrm{IO}|_{\nu_i}$ elements.
  - The twelve polynomials in the witness correspond to the R1CS $(A_{\mathsf{full}}, B_{\mathsf{full}}, C_{\mathsf{full}})$ compiled from P using the modular compiler. Recall that, for $\mathsf{f} \in \{\mathsf{row}, \mathsf{col}, \mathsf{val}\}$, $\mathsf{f}_M(x)$ is the interpolator of the *padded* vector $\overline{\mathsf{vec}}_{\mathsf{f},M_{\mathsf{full}}}$ over the distinguished domain of $\mathbb{F}_{\mathbb{i}}$ of size $\lceil {}^2 d_{M_{\mathsf{full}}} \rceil$, where row and col are padded with ones and val with zeros (cf. definition 12); and rowcol_M and $\mathsf{rowcolval}$ are the interpolators of $\mathsf{row}_M(x) \cdot \mathsf{col}_M(x)$ and $\mathsf{row}_M(x) \cdot \mathsf{col}_M(x) \cdot \mathsf{val}_M(x)$ over the same domain, respectively.
  - For each $\mathsf{f} \in \mathcal{F}_{\mathsf{key}}$ and $M \in \{A, B, C\}$, the degree bound $\mathsf{d}_{\mathsf{f},M}$ for f_M contained in the instance is $\lceil {}^2 d_{M_{\mathsf{full}}} \rceil - 1$, where $d_{M_{\mathsf{full}}}$ is the density of matrix M in the above R1CS.

The extended algebraic proof $\mathsf{eAHP}_{\mathsf{vkproof}} = (\mathsf{k}, \mathsf{s}, \mathsf{d}, \mathbf{I}, \mathbf{P}, \mathbf{V})$ we describe below has the following parameters:

- $\mathsf{k}(s) = 2$ for all index sizes $s = |\mathbb{i}|$.
- $\mathsf{s}(\mathbb{i}, 0)$ is the number of fingerprint polynomials corresponding to the instructions $\nu_1, \ldots, \nu_n$ in $\mathbb{i}$ up to depth $t_{\mathbb{i}}$:

$$\mathsf{s}(\mathbb{i}, 0) = 3 \cdot \sum_{i=1}^{n} (t_{\mathbb{i}} - \lceil \log_2(r_{\nu_i}) \rceil + 1) + (t_{\mathbb{i}} - \lceil \log_2(c_{\nu_i}) \rceil + 1) + 1$$

  with r_{ν_i} and c_{ν_i} as above.
- $\mathsf{s}(\mathbb{i}, 1) = 3$ (corresponding to $\mathsf{val}_A(x)$, $\mathsf{val}_B(x)$ and $\mathsf{val}_C(x)$) and $\mathsf{s}(\mathbb{i}, 2) = 6$ (corresponding to $\mathsf{pow}_{A,\alpha}(x)$, $\mathsf{pow}_{B,\alpha}(x)$, $\mathsf{pow}_{C,\alpha}(x)$, $h(x)$, $g(x)$ and $q(x)$).
- $\mathsf{d}(\mathbb{i}, 0)$, the common degree bound for all indexed polynomials, is the maximum degree among all fingerprint polynomials of the language described by $\mathbb{i}$, that is:

$$\mathsf{d}(\mathbb{i}, 0) = \max\{d_{M_{\nu_i}} - 1 : M \in \{A, B, C\}, 1 \leq i \leq n\}$$

  with $\nu_1, \ldots \nu_n$ as above.
- $\mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, 1, 1) = \lceil {}^2 d_{A_{\mathsf{full}}} \rceil - 1$, $\mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, 1, 2) = \lceil {}^2 d_{B_{\mathsf{full}}} \rceil - 1$ and $\mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, 1, 3) = \lceil {}^2 d_{C_{\mathsf{full}}} \rceil - 2$, where $(A_{\mathsf{full}}, B_{\mathsf{full}}, C_{\mathsf{full}})$ is the R1CS compiled from $\widetilde{\mathbb{x}} = P$ using the modular compiler. The bounds correspond to the $\mathsf{s}(\mathbb{i}, 1) = 3$ polynomials $\mathsf{val}_M(x)$ in the same order introduced above.

[39] We implicitly assume throughout that the verifier aborts if it encounters any formal defects in the program while processing its commands.

- $\mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, 2, 1) = \lceil {}^2 d_{A_{\mathsf{full}}} \rceil - 1$, $\mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, 2, 2) = \lceil {}^2 d_{B_{\mathsf{full}}} \rceil - 1$ and $\mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, 2, 3) = \lceil {}^2 d_{C_{\mathsf{full}}} \rceil - 1$ (bounds for the three pow polynomials).
- $\mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, 2, 4) = \mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, 2, 5) = \mathsf{d}(\mathbb{i}, \widetilde{\mathbb{x}}, 2, 6) = \max\{\lceil {}^2 d_{A_{\mathsf{full}}} \rceil - 2, \lceil {}^2 d_{B_{\mathsf{full}}} \rceil - 2, \lceil {}^2 d_{C_{\mathsf{full}}} \rceil - 2, 0\}$ (bounds for $h(x)$, $g(x)$ and $q(x)$).

Note that, as required by the definition of eAHPs, each of the above degree bounds $\mathsf{d}(-)$ except for $\mathsf{d}(|\mathbb{i}|, 0)$ is one of the $\mathsf{N}_{\mathbb{i},j}$.

4.1 Overview of the eAHP

Before delving into the indexer, prover and verifier algorithms in detail, we give an overview of their operation. Using vague terminology to simultaneously account for $\mathcal{P}$ and $\mathcal{V}$, the idea for each $M \in \{A, B, C\}$ is as follows:

- Send ($\mathcal{V}$: receive) a purported value polynomial val_M analogous to row_M and col_M.
- Reduce the correctness of rowcol_M and $\mathsf{rowcolval}_M$ to that of row_M, col_M and val_M using quotients $q_{\mathsf{rowcol},M}$ and $q_{\mathsf{rowcolval}_M}$.
- Verify the two quotient relations from the previous point at a random point.

  From now on, the correctness of row_M, col_M *and* val_M *is established by showing that their* actual *fingerprint at a random point* α *matches the* honest *fingerprint:*
- Receive ($\mathcal{V}$: sample and send) $\alpha \xleftarrow{R} \mathbb{F}$.
- Send ($\mathcal{V}$: receive) a polynomial $\mathsf{pow}_{M,\alpha}$ interpolating $(1, \alpha, \ldots, \alpha^{d_M - 1})$ over the domain K_M, where $d_M = \lceil {}^2 d_{M_{\mathsf{full}}} \rceil = \lceil {}^2 d_{\mathsf{inst},M} + d_{\mathsf{stru},M} \rceil$ and $K_M = K_{d_M}$.

  Honest fingerprint:
- Using (10_{row}) - (10_{val}), compute the *honest* hybrid fingerprint polynomials $\widetilde{\varphi}_{\mathsf{row},\mathcal{M},r_{\mathsf{full}},\alpha}(x)$, $\widetilde{\varphi}_{\mathsf{col},\mathcal{M},c_{\mathsf{full}},\alpha}(x)$ and $\widetilde{\varphi}_{\mathsf{val},\mathcal{M},\alpha}(x)$, where $\mathcal{M}$ is as in subsection 3.4 (after definition 15), then reveal ($\mathcal{V}$: query) their evaluation at α.
- Manually compute the *honest* fingerprints $\varphi_{\mathsf{row},M_{\mathsf{stru}},r_{\mathsf{full}}}(\alpha)$, $\varphi_{\mathsf{row},M_{\mathsf{stru}},c_{\mathsf{full}}}(\alpha)$, $\varphi_{\mathsf{val},M_{\mathsf{stru}}}(\alpha)$ at α.
- Using the previous values, compute $\varphi_{\overline{\mathsf{vec}}_{\mathsf{row},M_{\mathsf{full}},r_{\mathsf{full}}}}(\alpha)$, $\varphi_{\overline{\mathsf{vec}}_{\mathsf{col},M_{\mathsf{full}},c_{\mathsf{full}}}}(\alpha)$ and $\varphi_{\overline{\mathsf{vec}}_{\mathsf{val},M_{\mathsf{full}}}}(\alpha)$ by means of (12_{row}) - (12_{val}).

  Check against the actual fingerprint:
- Perform univariate sumchecks on the polynomials $\mathsf{row}_M \cdot \mathsf{pow}_{M,\alpha}$, $\mathsf{col}_M \cdot \mathsf{pow}_{M,\alpha}$ and $\mathsf{val}_M \cdot \mathsf{pow}_{M,\alpha}$ over the domain K_M using as the expected sum of evaluations the fingerprint at α from the previous point. Accept if and only if sumcheck verification passes.

The final interactive protocol is the result of transforming the above outline in several ways:

- In line with the eAHP formalism, the verifier's operation is split into an interactive phase, where all it does is send uniformly random field elements; a query phase, where the evaluations

of various polynomials at certain points are learnt; and a decision phase, where all expected relations are checked using the learnt values.

- Several steps are batched together across the three matrices (and the different functions f where possible), most notably the check on q_{rowcol} and $q_{\mathsf{rowcolval}}$ as well as the final univariate sumchecks. The powers of a random element β are used as batching randomness.
- Several steps which require a pass over all instructions (most notably, the computation of the various hybrid fingerprint polynomials and of the fingerprint of M_{stru} at α) are bundled together into a single pass.
- The relations between structural-constraint fingerprints from remark 16 are incorporated.

What follows are the full indexer, prover and verifier algorithms of the eAHP. As explained in section 2, most polynomials in the eAHP (more concretely: those output by the indexer, sent by the prover or appearing in the witness) will appear to the verifier as commitments once the protocol is compiled into a preprocessing argument. In the remainder of this section, we denote the index field $\mathbb{F}_{\mathbb{i}}$ simply by $\mathbb{F}$.

4.2 Indexer

$\mathbf{I}(\mathbb{i}) \to \{\{\varphi_{\mathsf{row},M_\nu,2^t}(x)\}_t, \{\varphi_{\mathsf{col},M_\nu,2^t}(x)\}_t, \varphi_{\mathsf{val},M_\nu}(x)\}_{M\in\{A,B,C\},\nu}$:

For each $\mathbb{i}_{\nu_i} = ((A_{\nu_i}, B_{\nu_i}, C_{\nu_i}), (d_{\nu_i,A}, d_{\nu_i,B}, d_{\nu_i,C}), (r_{\nu_i}, c_{\nu_i}), \mathrm{IO}_{\nu_i})$ in $\mathbb{i}$, $\mathbf{I}$ computes and outputs the following polynomials described in subsection 3.2:

- For each power of two $\lceil {}^2 r_{\nu_i} \rceil \leq 2^t \leq 2^{t_{\mathbb{i}}}$, the row fingerprint polynomials $\varphi_{\mathsf{row},A_{\nu_i},2^t}(x)$, $\varphi_{\mathsf{row},B_{\nu_i},2^t}(x)$ and $\varphi_{\mathsf{row},C_{\nu_i},2^t}(x)$ of depth t.
- For each power of two $\lceil {}^2 c_{\nu_i} \rceil \leq 2^t \leq 2^{t_{\mathbb{i}}}$, the column fingerprint polynomials $\varphi_{\mathsf{col},A_{\nu_i},2^t}(x)$, $\varphi_{\mathsf{col},B_{\nu_i},2^t}(x)$ and $\varphi_{\mathsf{col},C_{\nu_i},2^t}(x)$ of depth t.
- The value fingerprint polynomials $\varphi_{\mathsf{val},A_{\nu_i}}(x)$, $\varphi_{\mathsf{val},B_{\nu_i}}(x)$ and $\varphi_{\mathsf{val},C_{\nu_i}}(x)$.

4.3 Prover

$\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$:

The prover receives an index $\mathbb{i}$, an instance $\mathbb{x} = (([[\mathsf{f}_M]], \mathsf{d}_{\mathsf{f},M})_{\mathsf{f}\in\mathcal{F}_{\mathsf{key}},M\in\{A,B,C\}}, P)$ containing a program $P = (N, ((\nu_1,\rho_1),\ldots,(\nu_I,\rho_I)))$ and a witness $\mathbb{w} = ((\mathsf{f}_M)_{\mathsf{f}\in\mathcal{F}_{\mathsf{key}},M\in\{A,B,C\}}, \bot)$ as described at the beginning of this section.

1) **First round**:

   $\leftarrow$ Receive from $\mathcal{V}$ an empty message[40] $\bot \in \mathbb{F}^0$.

[40] This is caused by the eAHP formalism, which requires the verifier to make the first move (mirroring the AHP model). Our eAHP is in essence a sigma-protocol.

- For $M \in \{A, B, C\}$, compute the value vector $\mathsf{vec}_{\mathsf{val},M}$ corresponding to the full matrix M_{full} compiled from the program P, whose length is by definition the density $d_{\mathsf{inst},M} + d_{\mathsf{stru},M}$. Set $d_M = \lceil^2 d_{M_{\mathsf{full}}}\rceil = \lceil^2 d_{\mathsf{inst},M} + d_{\mathsf{stru},M}\rceil$ and $K_M = K_{d_M}$, then compute the polynomial $\mathsf{val}_M(x)$ which interpolates $\overline{\mathsf{vec}}_{\mathsf{val},M}$ over K_M – that is, it maps the i-th element of K_M to the value of the i-th non-zero entry of M if $i < d_{\mathsf{inst},M} + d_{\mathsf{stru},M}$ and to 0 otherwise. Note val_M can also be obtained directly by interpolating the values $(\mathsf{rowcolval}_M(k)/\mathsf{rowcol}_M(k))_{k\in K_M}$ over K_M.

→ Send to $\mathcal{V}$ the polynomials $\mathsf{val}_A(x)$, $\mathsf{val}_B(x)$ and $\mathsf{val}_C(x)$.

2) **Second round**:

← Receive from $\mathcal{V}$ $(\alpha, \beta) \in \mathbb{F}^2$.

- For[41] $M \in \{A, B, C\}$, compute the power polynomial[42] $\mathsf{pow}_{M,\alpha}(x) \in \mathbb{F}^{<d_M}[x]$ which maps the i-th element of K_M to[43] α^i.

  Instead of using an FFT, this can be done in linear time using lemma 18 ii) and iii). Note furthermore that all the inverses inside the sum in (25) featured in the lemma can be computed, using Montgomery's trick, with a single batch inversion in which the overall product $\prod_{i=0}^{d_M-1}(\alpha \cdot \omega^{-i} - 1)$ to be inverted is simply $1 - \alpha^{d_M}$.

- Set $d_{\max} = \max\{d_A, d_B, d_C\}$ and $K = K_{d_{\max}}$. Compute the univariate-sumcheck polynomials h and g uniquely defined by

$$\sum_{M\in\{A,B,C\}} \beta^M \cdot S_{K,K_M}(x) \cdot \mathsf{pow}_{M,\alpha}(x) \cdot \sum_{\mathsf{f}\in\mathcal{F}_{\mathsf{mat}}} \beta^{\mathsf{f}} \cdot \mathsf{f}_M(x) = h(x) \cdot v_K(x) + x \cdot g(x) + \Phi \quad (13)$$

  with $\deg(g) < d_{\max} - 1$ and $\Phi \in \mathbb{F}$, where

$$\beta^A = \beta^0, \quad \beta^B = \beta^1, \quad \beta^C = \beta^2, \quad \beta^{\mathsf{row}} = \beta^0, \quad \beta^{\mathsf{col}} = \beta^3, \quad \beta^{\mathsf{val}} = \beta^6$$

  and $v_K(x) = x^{d_{\max}} - 1$ is the vanishing polynomial over K. If $\mathcal{P}$'s inputs are honest, one has

$$\Phi = \frac{\sum_{M\in\{A,B,C\}} \sum_{\mathsf{f}\in\mathcal{F}_{\mathsf{mat}}} \beta^M \cdot \beta^{\mathsf{f}} \cdot \varphi_{\overline{\mathsf{vec}}_{\mathsf{f},M_{\mathsf{full}}}}(\alpha)}{d_{\max}}. \quad (14)$$

- Compute the batched rowcol and $\mathsf{rowcolval}$ quotient polynomials

$$q_{\mathsf{rowcol}}(x) = \sum_{M\in\{A,B,C\}} \beta^M \cdot \frac{S_{K,K_M}(x) \cdot (\mathsf{row}_M(x) \cdot \mathsf{col}_M(x) - \mathsf{rowcol}_M(x))}{v_K(x)} \quad (15)$$

  and

$$q_{\mathsf{rowcolval}}(x) = \sum_{M\in\{A,B,C\}} \beta^M \cdot \frac{S_{K,K_M}(x) \cdot (\mathsf{rowcol}_M(x) \cdot \mathsf{val}_M(x) - \mathsf{rowcolval}_M(x))}{v_K(x)}, \quad (16)$$

  then batch them into

$$q(x) = q_{\mathsf{rowcol}}(x) + \beta^3 \cdot q_{\mathsf{rowcolval}}(x). \quad (17)$$

→ Send to $\mathcal{V}$ the polynomials $\mathsf{pow}_{A,\alpha}(x)$, $\mathsf{pow}_{B,\alpha}(x)$, $\mathsf{pow}_{C,\alpha}(x)$, $h(x)$, $g(x)$ and $q(x)$.

[41] Since K_M has size d_M, which is a power of two which pads $d_{\mathsf{inst},M} + d_{\mathsf{stru},M}$, it is possible two or more among K_A, K_B and K_C coincide with relative frequency. In that case, the corresponding power polynomials $\mathsf{pow}_{M,\alpha}$ also coincide and do not need to be sent (in the compiled SNARG: committed) separately or have their correctness tested separately in the verifier's decision phase.

[42] The use of these polynomials, which always take the value 1 at the point 1 by definition, admits a small optimisation where the prover and verifier operate on $(f(x) - 1)/(x - 1)$ for each of them instead. This reduces the number of queries by 3 and the number of points queried by 1.

[43] For $\alpha = 0$, we set $\mathsf{pow}_\alpha(1) = 1$ for formal reasons.

4.4 Verifier

$\underline{\mathbf{V}^{\mathbb{i},\mathbf{I}(\mathbb{i})}(\mathbb{x})}$:

The verifier receives an instance $\mathbb{x} = (([[\mathsf{f}_M]], \mathsf{d}_{\mathsf{f},M})_{\mathsf{f}\in\mathcal{F}_{\mathsf{key}},M\in\{A,B,C\}}, P)$ with oracle access to the witness polynomials and containing program $P = (N, ((\nu_1,\rho_1),\ldots,(\nu_I,\rho_I)))$. It then operates as follows:

1) **First round**:

   $\leftarrow$ Send to $\mathcal{P}$ an empty message[40] $\perp \in \mathbb{F}^0$.

   $\rightarrow$ Receive from $\mathcal{P}$ the polynomials $\mathsf{val}_A(x)$, $\mathsf{val}_B(x)$ and $\mathsf{val}_C(x)$.

2) **Second round**:

   $\leftarrow$ Sample $(\alpha,\beta) \xleftarrow{R} \mathbb{F}^2$ and send it to $\mathcal{P}$.

   $\rightarrow$ Receive from $\mathcal{P}$ the polynomials $\mathsf{pow}_{A,\alpha}(x)$, $\mathsf{pow}_{B,\alpha}(x)$, $\mathsf{pow}_{C,\alpha}(x)$, $h(x)$, $g(x)$ and $q(x)$.

3) **Query phase**:

   - Sample $\gamma \xleftarrow{R} \mathbb{F}$.

   Before querying any polynomials, $\mathbf{V}$ must determine the (single) linear combination $\overline{\varphi}_{\mathsf{batch},\mathsf{inst}}(x)$ of indexed polynomials which will be queried. **Imporant note**: only in this query phase, we abuse notation by writing expressions such as

$$\overline{\varphi}_{\mathsf{row},\mathsf{inst},M}(x) = \overline{\varphi}_{\mathsf{row},\mathsf{inst},M}(x) + \overline{\omega}_{\mathsf{inst},\mathsf{row}} \cdot \overline{\alpha}_{\mathsf{inst},M} \cdot \varphi_{\mathsf{row},M_{\nu_i},r_{\mathsf{full}}}(x)$$

   Such a formula simply means that the eAHP verifier is updating the *coefficient* for the *label* of $\varphi_{\mathsf{row},M_{\nu_i},r_{\mathsf{full}}}(x)$ in the running linear combination $\overline{\varphi}_{\mathsf{row},\mathsf{inst},M}(x)$ by adding[44] $\overline{\omega}_{\mathsf{inst},\mathsf{row}} \cdot \overline{\alpha}_{\mathsf{inst},M} \in \mathbb{F}$.

   The computation of $\overline{\varphi}_{\mathsf{row},\mathsf{inst},M}(x)$ proceeds as follows:

   - For each $M \in \{A,B,C\}$, compute the density[45] $d_{\mathsf{stru},M}$ of M_{stru}. Recall this is the same for A and B, namely the number of structural constraints in the program; and that $d_{C,\mathsf{stru}}$ is the same magnitude minus the number of operands of the form $\mathsf{const}(0)$.
   - Compute $d_{\mathsf{inst},M} = \sum_{i=1}^{I} d_{M_{\nu_i}}$ and the padded full-program density $d_M = \lceil {}^2 d_{\mathsf{inst},M} + d_{\mathsf{stru},M} \rceil$, then set $K_M = K_{d_M}$ with distinguished generator $\omega_M = \omega_{d_M}$.
   - For $M \in \{A,B,C\}$, initialise $\overline{\alpha}_{\mathsf{inst},M} = \overline{\omega}_{\mathsf{inst},\mathsf{row}} = 1 \in \mathbb{F}$ and $\overline{\omega}_{\mathsf{inst},\mathsf{col}} = \omega_{c_{\mathsf{full}}}^{1+N} \in \mathbb{F}$.
   - For $M \in \{A,B,C\}$ and $\mathsf{f} \in \mathcal{F}_{\mathsf{mat}} = \{\mathsf{row},\mathsf{col},\mathsf{val}\}$, initialise $\overline{\varphi}_{\mathsf{f},\mathsf{inst},M}(x)$ as the zero linear combination of indexed polynomials.

[44] After compilation with a homomorphic PCS, this would translate to a corresponding operation on commitments. Note, however, that the verifier does not update the commitment to the linear combination instruction by instruction, but instead computes the final coefficients and then performs a single fixed-base MSM. Aside from the gains obtained from efficient MSM algorithms, this prevents the argument verifier from scaling the same commitment several times if the corresponding instruction appears more than once in the program.

[45] This step already necessitates the construction of the $\mathsf{locations}$ dictionary introduced a few lines below. The reason is that whether an operand ρ_i introduces a new structural constraint or not depends on whether the variable it refers to (if any) has already appeared in the program or not. We only defer the exposition of the construction of said dictionary to step iii) below for clarity.

- For each command (ν_i, ρ_i) in the program:
  - For $M \in \{A, B, C\}$, update

$$\overline{\varphi}_{\mathsf{row},\mathsf{inst},M}(x) = \overline{\varphi}_{\mathsf{row},\mathsf{inst},M}(x) + \overline{\omega}_{\mathsf{inst},\mathsf{row}} \cdot \overline{\alpha}_{\mathsf{inst},M} \cdot \varphi_{\mathsf{row},M_{\nu_i},r_{\mathsf{full}}}(x),$$

$$\overline{\varphi}_{\mathsf{col},\mathsf{inst},M}(x) = \overline{\varphi}_{\mathsf{col},\mathsf{inst},M}(x) + \overline{\omega}_{\mathsf{inst},\mathsf{col}} \cdot \overline{\alpha}_{\mathsf{inst},M} \cdot \varphi_{\mathsf{col},M_{\nu_i},c_{\mathsf{full}}}(x),$$

    and

$$\overline{\varphi}_{\mathsf{val},\mathsf{inst},M}(x) = \overline{\varphi}_{\mathsf{val},\mathsf{inst},M}(x) + \overline{\alpha}_{\mathsf{inst},M} \cdot \varphi_{\mathsf{val},M_{\nu_i}}(x).$$

  - Update the counters: set $\overline{\alpha}_{\mathsf{inst},M} = \overline{\alpha}_{\mathsf{inst},M} \cdot \alpha^{d_{\nu_i,M}}$ for $M \in \{A, B, C\}$, $\overline{\omega}_{\mathsf{inst},\mathsf{row}} = \overline{\omega}_{\mathsf{inst},\mathsf{row}} \cdot \omega_{r_{\mathsf{full}}}^{r_{\nu_i}}$ and $\overline{\omega}_{\mathsf{inst},\mathsf{col}} = \overline{\omega}_{\mathsf{inst},\mathsf{col}} \cdot \omega_{c_{\mathsf{full}}}^{c_{\nu_i}}$.
- Finally, batch the nine polynomial accumulators into a single polynomial

$$\overline{\varphi}_{\mathsf{batch},\mathsf{inst}}(x) = \sum_{M \in \{A,B,C\}} \sum_{\mathsf{f} \in \mathcal{F}_{\mathsf{mat}}} \beta^M \cdot \beta^{\mathsf{f}} \cdot \overline{\varphi}_{\mathsf{f},M}(x) \tag{18}$$

  with

$$\beta^A = \beta^0, \quad \beta^B = \beta^1, \quad \beta^C = \beta^2, \quad \beta^{\mathsf{row}} = \beta^0, \quad \beta^{\mathsf{col}} = \beta^3 \text{ and } \beta^{\mathsf{val}} = \beta^6.$$

At the end of the computation, the accumulators $\overline{\varphi}_{\mathsf{row},M}$, $\overline{\varphi}_{\mathsf{col},M}$ and $\overline{\varphi}_{\mathsf{val},M}$ are precisely the hybrid fingerprint polynomials

$$\widetilde{\varphi}_{\mathsf{row},\overline{\mathcal{M}},r_{\mathsf{full}},\alpha}(x) = \widetilde{\varphi}_{\mathsf{row},\mathcal{M},r_{\mathsf{full}},\alpha}(x),$$
$$\widetilde{\varphi}_{\mathsf{col},\overline{\mathcal{M}},c_{\mathsf{full}},\alpha}(x) = \omega_{c_{\mathsf{full}}}^{1+N} \cdot \widetilde{\varphi}_{\mathsf{col},\mathcal{M},r_{\mathsf{full}},\alpha}(x) \text{ and}$$
$$\widetilde{\varphi}_{\mathsf{val},\overline{\mathcal{M}},\alpha}(x) = \widetilde{\varphi}_{\mathsf{val},\mathcal{M},\alpha}(x)$$

respectively, with $\mathcal{M} = (M_{\nu_i})_{i=1}^I \in \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$ and $\overline{\mathcal{M}} \in \{\overline{\mathcal{A}}, \overline{\mathcal{B}}, \overline{\mathcal{C}}\}$ as in section 3.4.

The polynomial querying now proceeds as follows:

- Query all polynomials in the instance $\mathbb{x}$ at the point γ:

$$\mathsf{row}_A(x), \mathsf{col}_A(x), \mathsf{rowcol}_A(x), \mathsf{rowcolval}_A(x),$$
$$\mathsf{row}_B(x), \mathsf{col}_B(x), \mathsf{rowcol}_B(x), \mathsf{rowcolval}_B(x),$$
$$\mathsf{row}_C(x), \mathsf{col}_C(x), \mathsf{rowcol}_C(x) \text{ and } \mathsf{rowcolval}_C(x).$$

- Query all polynomials sent by $\mathcal{P}$ in the first round at the point γ:

$$\mathsf{val}_A(x), \mathsf{val}_B(x) \text{ and } \mathsf{val}_C(x).$$

- Query all polynomials sent by $\mathcal{P}$ in the second round at the point γ:

$$\mathsf{pow}_{A,\alpha}(x), \mathsf{pow}_{B,\alpha}(x), \mathsf{pow}_{C,\alpha}(x), h(x), g(x) \text{ and } q(x).$$

- For each $M \in \{A, B, C\}$, query the power polynomial $\mathsf{pow}_{M,\alpha}$ sent by $\mathcal{P}$ in the second round at the points 1 and $\omega_M^{-1} \cdot \gamma$.
- Query the batched hybrid fingerprint polynomial $\overline{\varphi}_{\mathsf{batch},\mathsf{inst}}(x)$, which is a linear combination of indexed polynomials, at the point α.

This amounts to

- 18 polynomials queried at the same point γ.
- 3 polynomials ($\mathsf{pow}_{A,\alpha}(x)$, $\mathsf{pow}_{B,\alpha}(x)$ and $\mathsf{pow}_{C,\alpha}(x)$) queried at three points each, the first one being the same γ as above, the second one being 1 and the third one being (potentially) different for each polynomial.
- 1 polynomial queried at α.

4) **Decision phase**:

In each of the checks described below, the verifier rejects if the tested equality (denoted $\stackrel{?}{=}$) does not hold.

Set $d_{\max} = \max\{d_A, d_B, d_C\}$ and $K = K_{d_{\max}}$.

i) **Verification of the rowcol and $\mathsf{rowcolval}$ quotients**: Verify that

$$q(\gamma) \cdot v_K(\gamma) \stackrel{?}{=} \sum_{M\in\{A,B,C\}} \beta^M \cdot S_{K,K_M}(\gamma) \cdot (\mathsf{row}_M(\gamma) \cdot \mathsf{col}_M(\gamma) - \mathsf{rowcol}_M(\gamma)) \\ + \beta^3 \cdot \sum_{M\in\{A,B,C\}} \beta^M \cdot S_{K,K_M}(\gamma) \cdot (\mathsf{rowcol}_M(\gamma) \cdot \mathsf{val}_M(\gamma) - \mathsf{rowcolval}_M(\gamma)) \tag{19}$$

with β^M as in the query phase. All evaluations come from the query phase except for $v_K(\gamma)$ and $S_{K,K_M}(\gamma)$, which can be computed with $O(\log(d_{\max}))$ squarings in $\mathbb{F}$ and will be reused in several subsequent steps.

ii) **Verification of the power-polynomial quotients**: For $M \in \{A, B, C\}$, verify that

$$\alpha \cdot \mathsf{pow}_{M,\alpha}(\omega^{-1} \cdot \gamma) - \mathsf{pow}_{M,\alpha}(\gamma) \stackrel{?}{=} \frac{\alpha^{d_M} - 1}{d_M} \cdot v_{K_M\setminus\{1\}}(\gamma), \tag{20}$$

(cf. lemma 18) and

$$\mathsf{pow}_{M,\alpha}(1) \stackrel{?}{=} 1.$$

All evaluations involved come from the query phase except for $v_{K_M\setminus\{1\}}(\gamma)$, which can be computed as $(\gamma^{d_M} - 1)/(\gamma - 1)$ if $\gamma \neq 1$ and simply equals d_M when $\gamma = 1$. Note that some of these values have likely already been computed when evaluating $S_{K,K_M}(\gamma)$ in step i).

iii) **Computation of structural fingerprints**: Perform a single pass[46] over the program to compute the fingerprints $\varphi_{\mathsf{row},M_{\mathsf{stru}},r_{\mathsf{full}}}(\alpha)$, $\varphi_{\mathsf{col},M_{\mathsf{stru}},c_{\mathsf{full}}}(\alpha)$ and $\varphi_{\mathsf{val},M_{\mathsf{stru}}}(\alpha)$ for $M \in \{A, B, C\}$. By remark 16, the main computation is restricted to four values out of those nine. In preparation for this step, do the following:

- Initialise an empty dictionary $\mathsf{locations}\colon \mathsf{VarIdentifiers} \to \mathbb{N}$. Let $v_1, \ldots, v_N$ be the identifiers of the variables in the Varuna instance – that is, the public input-output variables of the program P. For $1 \leq i \leq n$, set $\mathsf{locations}[v_i] = i$ (location 0 is reserved for the global constant 1).
- Initialise $\overline{\omega}_{\mathsf{stru},\mathsf{row}} = 1 \in \mathbb{F}$ and reset $\overline{\omega}_{\mathsf{inst},\mathsf{col}}$ to $\omega_{c_{\mathsf{full}}}^{1+N} \in \mathbb{F}$.

[46] In the compiled SNARG, this could be integrated in the same pass as the computation of the (commitments to) the hybrid fingerprint polynomials from the query phase. If done this way, the same counters $\overline{\omega}_{\mathsf{inst},\mathsf{row}}$ and $\overline{\omega}_{\mathsf{inst},\mathsf{col}}$ can be used for both purposes instead of resetting them as indicated in the second point below.

- Initialise $\overline{\alpha}_{\mathsf{stru},C} = 1 \in \mathbb{F}$.
- Initialise the fingerprint accumulators

$$\overline{\varphi}_{\mathsf{row},\mathsf{stru},C} = \overline{\varphi}_{\mathsf{col},\mathsf{stru},A} = \overline{\varphi}_{\mathsf{col},\mathsf{stru},C} = \overline{\varphi}_{\mathsf{val},\mathsf{stru},C} = 0 \in \mathbb{F}.$$

Then, for each command (ν_i, ρ_i) in the program, do the following:

- Accumulate the "local constant 1 equals global constant 1" constraint: set

$$\begin{aligned}
\overline{\varphi}_{\mathsf{row},\mathsf{stru},C} &= \overline{\varphi}_{\mathsf{row},\mathsf{stru},C} + \overline{\omega}_{\mathsf{stru},\mathsf{row}} \cdot \overline{\alpha}_{\mathsf{stru},C} \\
\overline{\varphi}_{\mathsf{col},\mathsf{stru},A} &= \overline{\varphi}_{\mathsf{col},\mathsf{stru},A} + \overline{\omega}_{\mathsf{inst},\mathsf{col}} \cdot \overline{\alpha}_{\mathsf{stru},AB} \\
\overline{\varphi}_{\mathsf{col},\mathsf{stru},C} &= \overline{\varphi}_{\mathsf{col},\mathsf{stru},C} + \overline{\alpha}_{\mathsf{stru},C} \\
\overline{\varphi}_{\mathsf{val},\mathsf{stru},C} &= \overline{\varphi}_{\mathsf{val},\mathsf{stru},C} + \overline{\alpha}_{\mathsf{stru},C}
\end{aligned}$$

- Update the structure-related counters: set $\overline{\omega}_{\mathsf{stru},\mathsf{row}} = \overline{\omega}_{\mathsf{stru},\mathsf{row}} \cdot \omega_{r_{\mathsf{full}}}$ and $\overline{\alpha}_{\mathsf{stru},C} = \overline{\alpha}_{\mathsf{stru},C} \cdot \alpha$.
- For each operand information $\rho_{i,j} \in \rho_i = (\rho_{i,1}, \ldots, \rho_{i,|\mathrm{IO}_{\nu_i}|})$:
  - **If $\rho_{i,j}$ is of the form** $\mathsf{var}(v)$ **and v is not a key in** $\mathsf{locations}$:
    Set $\mathsf{locations}[v] = \overline{\omega}_{\mathsf{inst},\mathsf{col}} \cdot \omega_{c_{\mathsf{full}}}^{\mathrm{IO}_{\nu_i,j}}$, where $\mathrm{IO}_{\nu_i,j}$ is the j-th element of IO_{ν_i} as defined in subsection 3.2 (that is, the relative column position of the j-th operand in ν_i's R1CS). Then move onto the next operand, skipping the remaining steps in the inner loop.
  - **Else if $\rho_{i,j}$ is of the form** $\mathsf{const}(k)$:
    Accumulate the "operand equals constant (with value k)" constraint: set

$$\overline{\varphi}_{\mathsf{col},\mathsf{stru},A} = \overline{\varphi}_{\mathsf{col},\mathsf{stru},A} + (\overline{\omega}_{\mathsf{inst},\mathsf{col}} \cdot \omega_{c_{\mathsf{full}}}^{\mathrm{IO}_{\nu_i,j}}) \cdot \overline{\alpha}_{\mathsf{stru},AB}$$

    Furthermore, **if** $k \neq 0$, also set

$$\begin{aligned}
\overline{\varphi}_{\mathsf{row},\mathsf{stru},C} &= \overline{\varphi}_{\mathsf{row},\mathsf{stru},C} + \overline{\omega}_{\mathsf{stru},\mathsf{row}} \cdot \overline{\alpha}_{\mathsf{stru},C} \\
\overline{\varphi}_{\mathsf{col},\mathsf{stru},C} &= \overline{\varphi}_{\mathsf{col},\mathsf{stru},C} + \overline{\alpha}_{\mathsf{stru},C} \\
\overline{\varphi}_{\mathsf{val},\mathsf{stru},C} &= \overline{\varphi}_{\mathsf{val},\mathsf{stru},C} + k \cdot \overline{\alpha}_{\mathsf{stru},C}
\end{aligned}$$

    and update $\overline{\alpha}_{\mathsf{stru},C} = \overline{\alpha}_{\mathsf{stru},C} \cdot \alpha$.
  - **Else (necessarily: $\rho_{i,j}$ is of the form** $\mathsf{var}(v)$ **and v is a key in** $\mathsf{locations}$**)**:
    Accumulate the "operand equals preexisting program variable (at column $\mathsf{locations}[v]$)" constraint:

$$\begin{aligned}
\overline{\varphi}_{\mathsf{row},\mathsf{stru},C} &= \overline{\varphi}_{\mathsf{row},\mathsf{stru},C} + \overline{\omega}_{\mathsf{stru},\mathsf{row}} \cdot \overline{\alpha}_{\mathsf{stru},C} \\
\overline{\varphi}_{\mathsf{col},\mathsf{stru},A} &= \overline{\varphi}_{\mathsf{col},\mathsf{stru},A} + (\overline{\omega}_{\mathsf{inst},\mathsf{col}} \cdot \omega_{c_{\mathsf{full}}}^{\mathrm{IO}_{\nu_i,j}}) \cdot \overline{\alpha}_{\mathsf{stru},AB}. \\
\overline{\varphi}_{\mathsf{col},\mathsf{stru},C} &= \overline{\varphi}_{\mathsf{col},\mathsf{stru},C} + \mathsf{locations}[v] \cdot \overline{\alpha}_{\mathsf{stru},C} \\
\overline{\varphi}_{\mathsf{val},\mathsf{stru},C} &= \overline{\varphi}_{\mathsf{val},\mathsf{stru},C} + \overline{\alpha}_{\mathsf{stru},C}.
\end{aligned}$$

    Also set $\overline{\alpha}_{\mathsf{stru},C} = \overline{\alpha}_{\mathsf{stru},C} \cdot \alpha$.
  - Update the remaining structure-related counter: set $\overline{\omega}_{\mathsf{stru},\mathsf{row}} = \overline{\omega}_{\mathsf{stru},\mathsf{row}} \cdot \omega_{r_{\mathsf{full}}}$.
- Update the instruction-related column counter: set $\overline{\omega}_{\mathsf{inst},\mathsf{col}} = \overline{\omega}_{\mathsf{inst},\mathsf{col}} \cdot \omega_{c_{\mathsf{full}}}^{c_{\nu_i}}$.

After going through all program commands, computation of the structural fingerprints concludes as follows:

- Expand the four fingeprints into the expected nine by setting

$$\overline{\varphi}_{\mathsf{row},\mathsf{stru},A} = \overline{\varphi}_{\mathsf{row},\mathsf{stru},B} = \sum_{i=0}^{d_{\mathsf{stru},A}-1} (\alpha \cdot \omega_{r_{\mathsf{full}}})^i.$$
$$\overline{\varphi}_{\mathsf{col},\mathsf{stru},B} = \sum_{i=0}^{d_{\mathsf{stru},B}-2} \alpha^i$$
$$\overline{\varphi}_{\mathsf{val},\mathsf{stru},A} = \overline{\varphi}_{\mathsf{val},\mathsf{stru},B} = \sum_{i=0}^{d_{\mathsf{stru},A}-1} \alpha^i,$$

  where the sums can be computed with the usual geometric-sum formula (when the ratio is not 1).
- Accumulate the final "zero-knowledge" constraint: set $\overline{\alpha}_{\mathsf{stru},A} = \overline{\alpha}_{\mathsf{stru},B} = \alpha^{d_{\mathsf{stru},AB}}$ (note that the density of A_{stru} and B_{stru} is equal to the number of rows of either) and

$$\overline{\varphi}_{\mathsf{row},\mathsf{stru},C} = \overline{\varphi}_{\mathsf{row},\mathsf{stru},C} + \overline{\alpha}_{\mathsf{stru},C}$$
$$\overline{\varphi}_{\mathsf{col},\mathsf{stru},A} = \overline{\varphi}_{\mathsf{col},\mathsf{stru},A} + \overline{\alpha}_{\mathsf{stru},A} \cdot \overline{\omega}_{\mathsf{inst},\mathsf{col}}$$
$$\overline{\varphi}_{\mathsf{col},\mathsf{stru},B} = \overline{\varphi}_{\mathsf{col},\mathsf{stru},B} + \overline{\alpha}_{\mathsf{stru},B} \cdot \omega_{c_{\mathsf{full}}} \cdot \overline{\omega}_{\mathsf{inst},\mathsf{col}}$$
$$\overline{\varphi}_{\mathsf{col},\mathsf{stru},C} = \overline{\varphi}_{\mathsf{col},\mathsf{stru},C} + \overline{\alpha}_{\mathsf{stru},C} \cdot \omega_{c_{\mathsf{full}}}^2 \cdot \overline{\omega}_{\mathsf{inst},\mathsf{col}}$$
$$\overline{\varphi}_{\mathsf{val},\mathsf{stru},M} = \overline{\varphi}_{\mathsf{val},\mathsf{stru},M} + \overline{\alpha}_{\mathsf{stru},M} \quad \text{for } M \in \{A, B, C\}.$$

  Observe that the zero-knowledge constraint part of the remaining fingerprints was added in the previous step.
- Pad the structural fingerprints and prepare them for addition to the instruction ones (cf. equuations (12_{row}) - (12_{val}) or definition 12 iii)) by setting

$$\overline{\varphi}_{\mathsf{row},\mathsf{stru},A} = \omega_{r_{\mathsf{full}}}^{r_{\mathsf{inst}}} \cdot \alpha^{d_{\mathsf{inst},M}} \cdot \overline{\varphi}_{\mathsf{row},\mathsf{stru},A} + \sum_{i=d_{\mathsf{inst},M}+d_{\mathsf{stru},M}}^{d_M-1} \alpha^i$$
$$\overline{\varphi}_{\mathsf{col},\mathsf{stru},A} = \alpha^{d_{\mathsf{inst},M}} \cdot \overline{\varphi}_{\mathsf{col},\mathsf{stru},A} + \sum_{i=d_{\mathsf{inst},M}+d_{\mathsf{stru},M}}^{d_M-1} \alpha^i$$

  for $M \in \{A, B, C\}$.

iv) **Computation of batched full-matrix fingerprint**: Batch the structural fingerprints with the same coefficients as their instruction analogues and combine the two by setting

$$\Phi = \overline{\varphi}_{\mathsf{batch},\mathsf{inst}}(\alpha) + \sum_{M \in \{A,B,C\}} \sum_{\mathsf{f} \in \mathcal{F}_{\mathsf{mat}}} \beta^M \cdot \beta^{\mathsf{f}} \cdot \overline{\varphi}_{\mathsf{f},\mathsf{stru},M},$$

where the first term in the right-hand side comes from the query phase. The resulting value Φ is precisely the (batched) full-program fingerprint

$$\sum_{M \in \{A,B,C\}} \sum_{\mathsf{f} \in \mathcal{F}_{\mathsf{mat}}} \beta^M \cdot \beta^{\mathsf{f}} \cdot \varphi_{\overline{\mathsf{vec}}_{\mathsf{f},M_{\mathsf{full}}}}(\alpha).$$

v) **Verification of the batched sumcheck proof**: verify that

$$\sum_{M\in\{A,B,C\}} \beta^M \cdot S_{K/K_M}(\gamma) \cdot \mathsf{pow}_{M,\alpha}(\gamma) \cdot \sum_{f\in\mathcal{F}_{\mathsf{mat}}} \beta^{\mathsf{f}} \cdot \mathsf{f}_M(\gamma) \stackrel{?}{=} h(\gamma)\cdot v_K(\gamma) + \gamma \cdot g(\gamma) + \frac{\Phi}{d_{\max}}, \tag{21}$$

where β^M and β^{f} are as in the query phase. Note all values therein come from the query phase or have been computed in previous steps.

4.5 Compilation into a preprocessed SNARG

Since $\mathsf{eAHP}_{\mathsf{vkproof}}$ is public coin, so is the preprocessing argument universal SRS compiled with any (separable homomorphic) PCS as described in subsection 2.4. Applying the Fiat-Shamir transform to the argument therefore results in a preprocessing SNARG.

We do not study the soundness of the latter, or indeed of the outcome of the double compilation process from a black-box eAHP into a SNARG ([Chi+20] does not define a notion of AHP soundness aimed at Fiat-Shamir security). Observe, however, that our soundness analysis for the eAHP in section 5.1 can be converted into an analysis of (intuitively) "round-by-round soundness", which is, in contexts where it is formally defined, a sufficient condition for the security of the Fiat-Shamir transform (cf. [BCS16], [Can+19], [Hol19], [Blo+23]).

5 Properties

In this section, we give a concrete soundness error for the eAHP presented above and study its dominant costs. We do not study (or indeed define) zero-knowledge or knowledge-soundness claims about the eAHP, since the witness (i. e. the Varuna proving key) can be recovered from the program P, which is part of the $\mathcal{R}_{\mathsf{VarunaVK}}$ instance.

5.1 Soundness of the eAHP

We study the soundness error of $\mathsf{eAHP}_{\mathsf{vkproof}}$. Together with the soundness error of the PCS used for compilation into a succinct argument, this yields an overall soundness-error bound for the latter by proposition 11.

Let $P = (N, (\nu_1, \rho_1), \ldots, (\nu_I, \rho_I))$ be a program as in section 3 and $(A_{\mathsf{full}}, B_{\mathsf{full}}, C_{\mathsf{full}})$ the R1CS produced from it by the modular compiler. We make extensive use of previously introduced notation without further notice. Furthermore, we use f^* to denote a polynomial in the Varuna verifying key (which is part of the instance) or sent by the possibly dishonest prover $\mathcal{P}^*$, purported to equal f – recall, however, that indexed polynomials are honest by the definition of eAHP soundness. For instance, row_A is the honest row polynomial of the matrix A_{full} whereas row^*_A is the corresponding polynomial in the instance, allegedly equal to the former; and val_A is the honest value polynomial of A_{full} whereas val^*_A is the one sent by the prover in the first round, allegedly equal to the former. As indicated in definition 2, only admissible dishonest provers need to be considered, that is, ones sending polynomials that satisfy the expected degree bounds. We recall

those bounds, stated at the start of section 4 and following from the definition Varuna proving keys and the $\mathsf{eAHP}_{\mathsf{vkproof}}$ prover:

- Each of the twelve polynomials in the instance, that is, f^*_M for $\mathsf{f} \in \mathcal{F}_{\mathsf{key}}$ and $M \in \{A, B, C\}$, should have degree $\leq d_M - 1$ (different for each M).
- Each of the three value polynomials val^*_M with $M \in \{A, B, C\}$ sent by $\mathcal{P}^*$ in the first round should have degree $\leq d_M - 1$ (different for each M).
- Each of the three power polynomials $\mathsf{pow}^*_{M,\alpha}$ with $M \in \{A, B, C\}$ sent by $\mathcal{P}^*$ in the second round should have degree $\leq d_M - 1$ (different for each M).
- Each of the remaining three polynomials sent by $\mathcal{P}^*$ in the second round, h^*, g^* and q^*, should have degree $\leq d_{\max} - 2$.

As the premise for the soundness analysis, suppose one of the polynomials whose oracles appear in the instance is incorrect, that is, $\overline{\mathsf{f}}^*_{\overline{M}} \neq \overline{\mathsf{f}}_{\overline{M}}$ for some $\overline{\mathsf{f}} \in \mathcal{F}_{\mathsf{key}}$ and $\overline{M} \in \{A, B, C\}$ (recall that the honest $\overline{\mathsf{f}}_{\overline{M}}$ is the interpolator of $\overline{\mathsf{vec}}_{\overline{\mathsf{f}},\overline{M}_{\mathsf{full}}}$ as explained at the start of section 4).

We conceptualise the probability of $\mathcal{V}$ accepting in terms of the following tree:

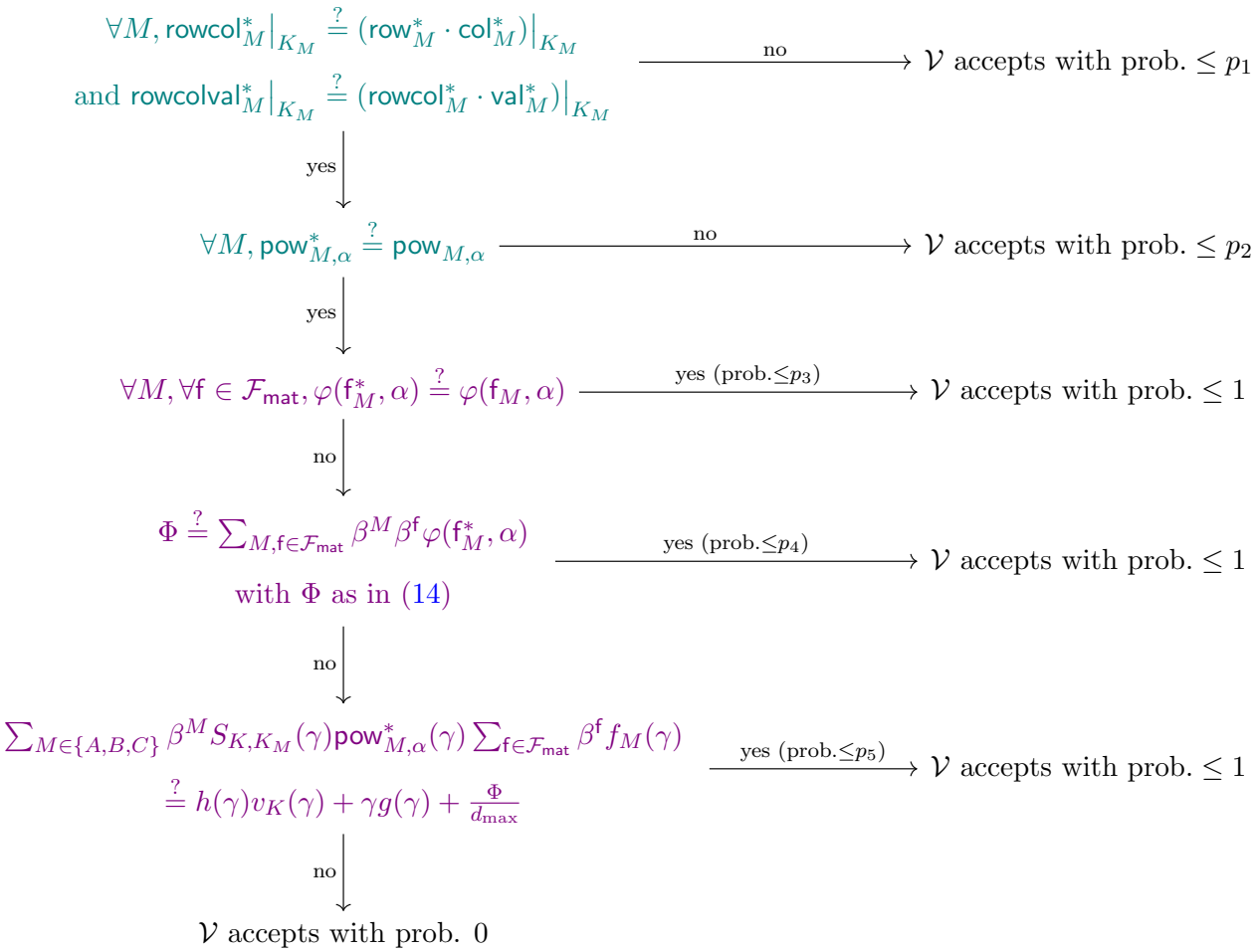

Figure 1: Structure of the soundness analysis of the eAHP.

(here $\forall M$ denotes $\forall M \in \{A, B, C\}$ and, throughout this section, $\sum_M$ denotes $\sum_{M\in\{A,B,C\}}$). The

tree has 5 internal nodes, which we refer to simply as nodes, and 6 leaves. At each node, we consider the probability of $\mathcal{V}$ accepting conditioned on all conditions in the path from the root to it holding. The bound for the overall probability that $\mathcal{V}$ accepts is then reconstructed from the leaves all the way to the root.

Note that the large union bound $\Pr(\mathcal{V} \text{ accepts}) \leq \sum_{i=1}^{5} p_i$ is, while not incorrect, unnecessarily loose. In the first two nodes (in teal), we have no control over the probability of each child path being taken: it is chosen by the possibly dishonest prover $\mathcal{P}^*$. However, we are able to bound the probability of $\mathcal{V}$ accepting in each child's subtree and hence the probability of $\mathcal{V}$ accepting (conditioned on that node itself) is the *maximum* of its children's probabilities. In the other three nodes (in violet), we can instead bound the probability of one of the two paths (the "bad" one to the right) being taken and, separately, the probability of $\mathcal{V}$ accepting in the other child's subtree. This is a classical union bound, where the bound for the node itself is the *sum* of the bad-path probability and the good-subtree one[47].

In other words, one has

$$\Pr(\mathcal{V} \text{ accepts}) \leq \max\{p_1, p_2, p_3 + p_4 + p_5\} \tag{22}$$

and all that remains is to find valid values for each of the p_i. The result can be found in proposition 17 at the end of this subsection.

Computing p_1

If, for some $M \in \{A, B, C\}$, rowcol^*_M and $\mathsf{row}^*_M \cdot \mathsf{col}^*_M$ do not coincide over K_M, one has $v_{K_M}(x) \nmid \mathsf{row}^*_M(x) \cdot \mathsf{col}^*_M(x) - \mathsf{rowcol}^*_M(x)$ or, equivalently, $v_K \nmid S_{K,K_M}(x) \cdot (\mathsf{row}^*_M(x) \cdot \mathsf{col}^*_M(x) - \mathsf{rowcol}^*_M(x))$. An analogous division failure occurs if $\mathsf{rowcolval}^*_M$ and $\mathsf{rowcol}^*_M \cdot \mathsf{val}^*_M$ do not coincide over K_M. By a classic result (cf. for instance [GWC19] claim 4.6), one has

$$\Pr_{\beta \xleftarrow{R} \mathbb{F}} \left(v_K(x) \text{ divides } \begin{array}{l} \displaystyle\sum_M \beta^M \cdot S_{K,K_M}(x) \cdot (\mathsf{row}^*_M(x) \cdot \mathsf{col}^*_M(x) - \mathsf{rowcol}^*_M(x)) \\ \displaystyle +\beta^3 \sum_M \beta^M \cdot S_{K,K_M}(x) \cdot (\mathsf{rowcol}^*_M(x) \cdot \mathsf{val}^*_M(x) - \mathsf{rowcolval}^*_M(x)) \end{array} \right) \leq \frac{5}{|\mathbb{F}|}.$$

If this does not happen, then no matter what q the prover sends in the second round (with the appropriate degree bound), the probability that (19) holds at $x = \gamma \xleftarrow{R} \mathbb{F}$ is $\leq (2 \cdot d_{\max} - 2)/|\mathbb{F}|$ by the Schwartz-Zippel lemma.

Let E_1 be the event that, for some $M \in \{A, B, C\}$, $\mathsf{rowcol}^*_M\big|_{K_M} \neq (\mathsf{row}^*_M \cdot \mathsf{col}^*_M)\big|_{K_M}$ or $\mathsf{rowcolval}^*_M\big|_{K_M} \neq$

[47] For the mathematically inclined: considering the law of conditional probability $\Pr(A) = \Pr(A \mid B) \cdot \Pr(B) + \Pr(A \mid \neg B) \cdot \Pr(\neg B)$, the two teal nodes are ones where we can control the first and third terms of the right-hand side (which results in a maximum bound for the left-hand side); and the three violet ones are those where we can control the first and last terms of the right-hand side (which results in the usual union-bound sum).

$(\mathsf{rowcol}^*_M \cdot \mathsf{val}^*_M)\big|_{K_M}$. By a union bound, one has

$$\begin{aligned}\Pr(\mathcal{V} \text{ accepts} \mid E_1) \leq & \Pr(\mathcal{V} \text{ accepts} \mid E_1 \text{ and } v_K(x) \nmid f(x)) + \\ & \Pr(v_K(x) \mid f(x) \mid E_1). \\ \leq & \Pr_{\gamma \xleftarrow{R} \mathbb{F}} (q(\gamma) \cdot v_K(\gamma) = f(\gamma) \mid E_1 \text{ and } v_K(x) \nmid f(x)) + \\ & \Pr_{\beta \xleftarrow{R} \beta} (v_K(x) \mid f(x) \mid E_1) \\ \leq & \frac{(2 \cdot d_{\max} - 2)}{|\mathbb{F}|} + \frac{5}{|\mathbb{F}|} = \frac{2 \cdot d_{\max} + 3}{|\mathbb{F}|}.\end{aligned}$$

where $f(x)$ denotes the polynomial inside the previous displayed equation. The second inequality is crucial: β can only be modelled as $\beta \xleftarrow{R} \mathbb{F}$ because it is sampled *after* all other elements (especially polynomials) involved in the condition $v_K(x) \mid f(x)$ have been sent by the prover or otherwise made available to $\mathcal{V}$ - this is a fundamental requisite for batching. Similarly, γ can be modelled as $\gamma \xleftarrow{R} \mathbb{F}$ because it is independent of the other elements appearing in the condition $q(\gamma) \cdot v_K(\gamma) = f(\gamma)$ (more concretely: all other elements, including polynomials and β, have been fixed before γ is sampled).

This yields the valid bound choice $p_1 = (2 \cdot d_{\max} + 3)/|\mathbb{F}|$.

Computing p_2

The computation of a valid choice for p_2 boils to an application of the Schwartz-Zippel lemma, but some subtleties arise depending on whether $\alpha \in K_M$ or not. We defer the details to appendix A. In particular, lemma 18 iv) therein shows[48] that, if $\mathsf{pow}^*_{M,\alpha} \neq \mathsf{pow}_{M,\alpha}$, the probability that the verifier's check (20) passes is at most $(d_M - 1)/|\mathbb{F}|$. Note that the lemma can be applied only because $\mathsf{pow}^*_{M,\alpha}$ is sent to the verifier before γ is sampled. One can therefore set

$$p_2 = \max_{M \in \{A,B,C\}} \left\{ \frac{d_M - 1}{|\mathbb{F}|} \right\} = \frac{d_{\max} - 1}{|\mathbb{F}|}.$$

Computing p_3

A valid p_3 can be deduced immediately from the Schwartz-Zippel lemma, as we show in a few lines. However, we first need to switch[49] from the original assumption $\bar{\mathsf{f}} \in \mathcal{F}_{\mathsf{key}}$ to a polynomial (label) in $\mathcal{F}_{\mathsf{mat}}$.

Claim: Conditioned on the first vertical arrow of figure 1, there exists an $\tilde{\mathsf{f}} \in \mathcal{F}_{\mathsf{mat}}$ such that $\tilde{\mathsf{f}}^*_{\overline{M}} \neq \tilde{\mathsf{f}}_{\overline{M}}$.

To see why, recall that $\bar{\mathsf{f}}^*_{\overline{M}} \neq \bar{\mathsf{f}}_{\overline{M}}$ by the overall assumption of the soundness analysis. If $\bar{\mathsf{f}}$ is row or col we are done, since those function labels also belong to $\mathcal{F}_{\mathsf{mat}}$ and one can set $\tilde{\mathsf{f}} = \bar{\mathsf{f}}$. Otherwise,

[48] The proof of lemma 18 shows something slightly stronger: one can drop the check that $\mathsf{pow}_{M,\alpha}(1) = 1$ from the protocol altogether (including the corresponding query) at the cost of an (additional) soundness loss of $d_{\max}/|\mathbb{F}|$ (caused by the possibility that $\alpha \in K_M$). We stress that this soundness loss is not restricted to p_2 (i. e. it does not get maximised away in (22)) but would manifest as an additional $d_{\max}/|\mathbb{F}|$ in the final ε from proposition 17.

[49] Indeed, q was introduced into the protocol with the express purpose of moving from fingerprints of rowcol and $\mathsf{rowcolval}$ to those of val (alongside row and col), which are cheaper to compute.

note first that $\bar{\mathsf{f}}$ cannot be rowcol as

$$\mathsf{rowcol}^*_{\overline{M}}\big|_{K_M} = (\mathsf{row}^*_{\overline{M}} \cdot \mathsf{col}^*_{\overline{M}})\big|_{K_M} = (\mathsf{row}_{\overline{M}} \cdot \mathsf{col}_{\overline{M}})\big|_{K_M} = \mathsf{rowcol}_{\overline{M}}\big|_{K_M}$$

and $\mathsf{rowcol}^*_{\overline{M}}$ and $\mathsf{rowcol}_{\overline{M}}$ both have degree $< d_M = |K_M|$, which yields $\mathsf{rowcolval}^*_{\overline{M}} = \mathsf{rowcol}^*_{\overline{M}}$. The first equality in the displayed equation follows from the conditioning on the first vertical arrow in the diagram and the second one from $\bar{\mathsf{f}} \notin \{\mathsf{row}, \mathsf{col}\}$. One therefore necessarily has $\bar{\mathsf{f}} = \mathsf{rowcolval}$, i. e. $\mathsf{rowcolval}^*_{\overline{M}} \neq \mathsf{rowcolval}_{\overline{M}}$ or equivalently by degree considerations $\mathsf{rowcolval}^*_{\overline{M}}\Big|_{K_M} \neq \mathsf{rowcolval}_{\overline{M}}\big|_{K_M}$. But then $\mathsf{val}^*_{\overline{M}}\Big|_{K_M} \neq \mathsf{val}_{\overline{M}}\big|_{K_M}$, since otherwise we would reach the contradiction

$$\begin{aligned}\mathsf{rowcolval}^*_{\overline{M}}\big|_{K_M} = (\mathsf{rowcol}^*_{\overline{M}} \cdot \mathsf{val}^*_{\overline{M}})\big|_{K_M} &= (\mathsf{row}^*_{\overline{M}} \cdot \mathsf{col}^*_{\overline{M}} \cdot \mathsf{val}^*_{\overline{M}})\big|_{K_M} \\ &= (\mathsf{row}_{\overline{M}} \cdot \mathsf{col}_{\overline{M}} \cdot \mathsf{val}_{\overline{M}})\big|_{K_M} = \mathsf{rowcolval}_{\overline{M}}\big|_{K_M}\end{aligned}$$

for analogous reasons to the above. This shows $\tilde{\mathsf{f}} = \mathsf{val}$ (note that $\mathsf{val}^*_{\overline{M}} = \mathsf{val}_{\overline{M}}$ if and only if $\mathsf{val}^*_{\overline{M}}\Big|_{K_M} = \mathsf{val}_{\overline{M}}\big|_{K_M}$ by the same degree considerations as earlier) and concludes the proof of the claim.

In order to compute p_3 we simply note that, letting E_3 denote the event that the two vertical arrows in figure 1 occur, one has

$$\begin{aligned}&\Pr\left(\forall M, \forall \mathsf{f} \in \mathcal{F}_{\mathsf{mat}}, \varphi(\mathsf{f}^*_M, \alpha) = \varphi(\mathsf{f}^*_M, \alpha) \;\middle|\; E_3\right) \\ \leq &\Pr\left(\varphi(\tilde{\mathsf{f}}^*_{\overline{M}}, \alpha) = \varphi(\tilde{\mathsf{f}}_{\overline{M}}, \alpha) \;\middle|\; E_3\right) \\ = &\Pr\left(\alpha \text{ is a root of the non-zero polynomial } \sum_{i=0}^{d_M - 1} (\tilde{\mathsf{f}}^*_{\overline{M}}(k_i) - \tilde{\mathsf{f}}_{\overline{M}}(k_i)) \cdot x^i \;\middle|\; E_3\right) \\ \leq &\frac{d_{\overline{M}} - 1}{|\mathbb{F}|} \\ \leq &\frac{d_{\max} - 1}{|\mathbb{F}|},\end{aligned}$$

where k_i denotes the i-th element of $K_{\overline{M}}$ (in the predefined order) and the next-to-last inequality follows from the Schwartz-Zippel lemma together with the fact that α is sampled *after* the verifier has received $\tilde{\mathsf{f}}_{\overline{M}}(k_i)$ (which is true whether $\tilde{\mathsf{f}}$ is row, col or val). This yields the valid bound choice $p_3 = (d_{\max} - 1)/|\mathbb{F}|$.

Computing p_4

The probability that

$$\Phi = \sum_M \sum_{\mathsf{f} \in \mathcal{F}_{\mathsf{mat}}} \beta^M \cdot \beta^{\mathsf{f}} \cdot \varphi_{\mathsf{f},M}(\alpha) = \sum_M \sum_{\mathsf{f} \in \mathcal{F}_{\mathsf{mat}}} \beta^M \cdot \beta^{\mathsf{f}} \cdot \varphi(\mathsf{f}_M, \alpha),$$

coincides with $\sum_M \sum_{\mathsf{f} \in \mathcal{F}_{\mathsf{mat}}} \beta^M \cdot \beta^{\mathsf{f}} \cdot \varphi(\mathsf{f}^*_M, \alpha)$ conditioned on the third vertical arrow (which entails $\varphi(\tilde{\mathsf{f}}_{\overline{M}}, \alpha) \neq \varphi(\tilde{\mathsf{f}}^*_{\overline{M}}, \alpha)$ for some $\overline{M} \in \{A, B, C\}$, $\tilde{\mathsf{f}} \in \mathcal{F}_{\mathsf{mat}}$) is $\leq 8/|\mathbb{F}|$ by the Schwartz-Zippel lemma. Note here that β is sampled after all f^*_M are fixed, and independently from α. We can therefore set $p_4 = 8/|\mathbb{F}|$.

Computing p_5

Observe that, for all $M \in \{A, B, C\}$ and $\mathsf{f} \in \mathcal{F}_{\mathsf{mat}}$, one has

$$\sum_{\tau \in K} S_{K,K_M}(\tau) \cdot \mathsf{pow}^*_{M,\alpha}(\tau) \cdot \mathsf{f}^*_M(\tau) = \sum_{\tau \in K_M} \mathsf{pow}^*_{M,\alpha}(\tau) \cdot \mathsf{f}^*_M(\tau) = \varphi(\mathsf{f}^*_M, \alpha)$$

and therefore, for any $h^* \in \mathbb{F}[x]$, $g^* \in \mathbb{F}^{<d_{\max}}[x]$ and $\widetilde{\Phi} \in \mathbb{F}$ satisfying

$$\sum_{M \in \{A,B,C\}} \sum_{\mathsf{f} \in \mathcal{F}_{\mathsf{mat}}} \beta^M \cdot \beta^{\mathsf{f}} \cdot S_{K,K_M}(x) \cdot \mathsf{pow}_{M,\alpha}(x) \cdot \mathsf{f}^*_M(x) = h^*(x) \cdot v_K(x) + x \cdot g^*(x) + \frac{\widetilde{\Phi}}{d_{\max}}, \quad (23)$$

one necessarily has $\widetilde{\Phi} = \sum_{M,\mathsf{f}} \beta^M \cdot \beta^{\mathsf{f}} \cdot \varphi(\mathsf{f}^*_M, \alpha)$ by the standard univariate-sumcheck result.

Therefore, conditioned on the fourth vertical arrow in figure 1, the h^* and g^* sent by the prover cannot possibly satisfy (23) with $\widetilde{\Phi} = \Phi = \sum_{M,\mathsf{f}} \beta^M \cdot \beta^{\mathsf{f}} \cdot \varphi(\mathsf{f}^*_M, \alpha)$, as tested by the verifier at the random point γ. The Schwartz-Zippel lemma thus yields the valid bound choice $p_5 = (d_{\max} - 1)/|\mathbb{F}|$.

Substituting the bounds p_i found above into the right-hand side of (22) yields the following:

Proposition 17. *The extended algebraic holographic proof* $\mathsf{eAHP}_{\mathsf{vkproof}} = (\mathsf{k}, \mathsf{s}, \mathsf{d}, \mathbf{I}, \mathbf{P}, \mathbf{V})$ *has soundness error*

$$\varepsilon = \frac{3 \cdot d_{\max} + 5}{|\mathbb{F}_{\mathbb{i}}|},$$

where $d_{\max} \leq \mathsf{N}_{\mathbb{i}}$ *is* $\max\{\lceil {}^2 d_{A_{\mathsf{full}}} \rceil, \lceil {}^2 d_{B_{\mathsf{full}}} \rceil, \lceil {}^2 d_{C_{\mathsf{full}}} \rceil\}$*, i. e. the maximum among the three padded densities of the R1CS* $(A_{\mathsf{full}}, B_{\mathsf{full}}, C_{\mathsf{full}})$ *compiled from the program* P *in the instance.*

5.2 Costs

In this subsection, we analyse the costs (including round complexity, communication complexity and runtime) of three different protocol scenarios:

- The preprocessing argument compiled from an abstract extended algebraic holographic proof and polynomial commitment scheme.
- The preprocessing argument compiled from the vkproof eAHP and an abstract PCS.
- The specific preprocessing argument compiled from the vkproof eAHP and the Marlin PCS.

Note that we only study the costs of the *interactive* argument without the applying the final transformation to a preprocessing SNARG, that is, the Fiat-Shamir transform. Since vkproof has constant communication complexity (thanks largely to the use of a homomorphic PCS) and many PCS have small, constant-size commitments (in particular, so does the Marlin PCS), the Fiat-Shamir transform has a minimal impact on performance.

Abstract eAHP and PCS: The cost analysis when the eAHP and PCS are treated as a black-box is very similar to that in [Chi+20] theorem 8.1. Let $\mathsf{eAHP} = (\mathsf{k}, \mathsf{s}, \mathsf{d}, \mathbf{I}, \mathbf{P}, \mathbf{V})$, $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ and $\mathcal{R}$ be as in definition 9, which are compiled into a preprocessing argument $\mathsf{eAHP}_{\mathsf{PC}} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$.

Let $(\mathbb{i}, \mathsf{ck}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{PC}_{\mathsf{NH}}}$.

- **Round complexity**: the interaction between proceeds in $\mathsf{k}(|\mathbb{i}|) + 2$ rounds, as detailed in definition 9: $\mathsf{k}(|\mathbb{i}|)$ rounds for the interaction between $\mathbf{P}$ and $\mathbf{V}$, one for the querying randomness $\rho_{\mathsf{k}(|\mathbb{i}|)+1}$ and the tuple v of replies to the queries, and an additional one for the opening challenge $\xi = \rho_{\mathsf{k}(|\mathbb{i}|)+2}$ and the opening proof π.
- **Communication complexity**: $\mathcal{P}$ sends
  - $\sum_{i=1}^{\mathsf{k}(|\mathbb{i}|)} \mathsf{s}(\mathbb{i}, i)$ commitments.
  - $|Q| = \left|\mathbf{Q}^{\mathbb{i}}(\widetilde{\mathbb{x}}, \rho_1, \ldots, \rho_{\mathsf{k}(|\mathbb{i}|)}, \rho_{\mathsf{k}(|\mathbb{i}|)+1})\right|$ field elements (one evaluation per query).
  - One batched PCS opening proof π for Q as above (with polynomial degrees depending on Q and the non-oracle part of the instance $\widetilde{\mathbb{x}}$).

  $\mathcal{V}$ sends $\sum_{i=1}^{\mathsf{k}(|\mathbb{i}|)+2} |\rho_i|$ field elements, where $\left|\rho_{\mathsf{k}(|\mathbb{i}|)+2}\right| = 1$ (the opening challenge ξ).

  Assuming the size of field elements in $\mathbb{F}_{\mathbb{i}}$ is $\mathrm{poly}(\lambda)$, commitments have size $\mathrm{poly}(\lambda)$ (for instance, they consist of a constant number of field elements), every polynomial sent by $\mathbf{P}$ is queried at least once by $\mathbf{V}$ (so $|Q| \geq \sum_{i=1}^{\mathsf{k}(|\mathbb{i}|)} \mathsf{s}(\mathbb{i}, i)$) and the opening proof has size $|Q| \cdot \mathrm{poly}(\lambda)$ (in particular, it is independent of the degree bounds it refers to), then the communication complexity from $\mathcal{P}$ to $\mathcal{V}$ is $O_\lambda(|Q|)$.
- **Indexer time**: $\mathcal{I}$ performs the following operations:
  - Runs $\mathbf{I}(\mathbb{i})$.
  - Trims the SRS to the degree bound $\mathsf{N}_{\mathbb{i}}$ (which can be considered to take negligible time).
  - Commits to $\mathsf{s}(\mathbb{i}, 0)$ polynomials with the same degree bound $\mathsf{d}(\mathbb{i}, 0)$ and empty randomness.
  - Computes any desired preprocessed multi-scalar-multiplication data on the $\mathsf{s}(\mathbb{i}, 0)$ commitments.
- **Prover time**: $\mathcal{P}$ performs the following operations:
  - Runs all $\mathsf{k}(|\mathbb{i}|)$ interactive rounds of $\mathbf{P}$.
  - Makes $\mathsf{k}(|\mathbb{i}|)$ calls to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}$.
  - Computes the linear combinations of indexed polynomials queried in the query set Q. Each such polynomial has degree $\leq \mathsf{d}(\mathbb{i}, 0)$.
  - Evaluates all queried polynomials and linear combinations at the relevant points in Q.
  - Performs a single call to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Open}$ with $|Q|$ queries.
- **Verifier time**: The verifier $\mathcal{V}$ performs the following main operations, assuming sampling randomness ρ_i and computing the query set Q takes negligible time:
  - Computes the linear combinations of commitments corrresponding to linear-combination queries in Q using the indexed MSM information, if any.
  - Verifies the opening proof corresponding to the $|Q|$ claimed evaluations with a single call to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Check}$.
  - Runs the decision algorithm of $\mathbf{V}$.

vkproof eAHP and abstract PCS: The specific $\mathsf{eAHP}_{\mathsf{vkproof}}$ parameters and algorithms from section 4 result in the following specific costs for $(\mathsf{eAHP}_{\mathsf{vkproof}})_{\mathsf{PC}} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$, where we denote by mul multiplication or inversion[50] in $\mathbb{F}_{\mathbb{i}}$.

Let $(\mathbb{i}, \mathsf{ck}, \mathbb{x}, \mathbb{w}) \in (\mathcal{R}_{\mathsf{VarunaVK}})_{\mathsf{PC}_{\mathsf{NH}}}$, where $\mathbb{x} = ((c_i, \mathsf{d}_i)_{i=1}^{12}, P)$ for some program P over the language defined in $\mathbb{i}$. Let $d_{\max}$ be as in section 4, that is, the maximum among the densities $\{d_A, d_B, d_C\}$ of the three full-program matrices compiled from P.

- **Round complexity**: $\mathsf{eAHP}_{\mathsf{vkproof}}$ has constant round complexity $\mathsf{k}(|\mathbb{i}|) = 2$ rounds, and therefore $\mathcal{P}$ and $\mathcal{V}$ interact throughout 4 rounds.
- **Communication complexity**: $\mathcal{P}$ sends the verifier $3 + 6 = 9$ commitments, $|Q| = 28$ field elements and one batched PCS opening proof. $\mathcal{V}$ sends the prover $\rho_1 + \rho_2 + \rho_3 + \rho_4 = 0 + 2 + 1 + 1 = 4$ field elements.
- **Indexer time**: $\mathcal{I}$ performs the following operations:
  - Runs $\mathbf{I}$ on the index $\mathbb{i}$ containing a list of I instructions and a maximum depth $t_{\mathbb{i}}$, where $\mathsf{N}_{\mathbb{i}} = 2^{t_{\mathbb{i}}}$. For each instruction ν, $\mathbf{I}$ computes $\mathsf{s}(\mathbb{i}, 0) \leq 3 + 6 \cdot t_{\mathbb{i}}$ matrix-fingerprint polynomials of degree $< d_{M_\nu}$ each (where $M \in \{A, B, C\}$ is the corresponding matrix). This amounts to $\sim 2 \cdot t_{\mathbb{i}} \cdot \sum_\nu d_{A_\nu} + d_{B_\nu} + d_{C_\nu}$ mul.
  - Trims the SRS to the degree bound $\mathsf{N}_{\mathbb{i}}$.
  - Commits to the $\mathsf{s}(\mathbb{i}, 0)$ polynomials from the first step with a single call to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}$.
  - Computes any desired preprocessed multi-scalar-multiplication data on the $\mathsf{s}(\mathbb{i}, 0)$ commitments, which we do not detail here. It should be noted that only indexed polynomials corresponding to the same depth t will ever be multiplied together by the verifier, which may reduce the size and computational cost of MSM precomputations.
- **Prover time**: $\mathcal{P}$ performs the following operations:
  - First round of $\mathbf{P}$, i. e. computing $\mathsf{val}_A(x)$, $\mathsf{val}_B(x)$ and $\mathsf{val}_C(x)$: $O(d_{\max} \cdot \log(d_{\max}))$ mul.
  - Second round of $\mathbf{P}$, i. e. computation of $\mathsf{pow}_{M,\alpha}(x)$ for $M \in \{A, B, C\}$, $q(x)$, $h(x)$ and $g(x)$: $O(d_{\max} \cdot \log(d_{\max}))$ mul. The superlinear factor essentially comes from the $\mathsf{row}_M(x) \cdot \mathsf{col}_M(x)$ and $\mathsf{rowcol}_M(x) \cdot \mathsf{val}(x)$ products, which can be computed using FFTs[51].
  - Makes 2 calls to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}$.
  - Computes the unique linear combination $\overline{\varphi}_{\mathsf{batch},\mathsf{inst}}(x)$ (equation (18)) of indexed polynomials in the query set: $O(d_{\max})$ mul.
  - Evaluates all queried polynomials at the relevant points according to the 28 queries: $\leq 28 \cdot d_{\max}$ mul.
  - Performs a single call to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Open}$ with 28 queries and degree bounds as specified in $\mathsf{eAHP}_{\mathsf{vkproof}}$.
- **Verifier time**: The verifier $\mathcal{V}$ performs the following main operations. Note that sampling the $\sum_{i=1}^{4} |\rho_i| = 4$ random field elements takes negligible time.

[50] The number of field inversions in the protocol is constant and, in fact, very low.

[51] Since all involved polynomial are typically obtained in evaluation form first and then converted to the monomial basis using FFTs, one of the two FFTs in each of the relevant products can be avoided if $\mathcal{P}$ is also provided with the polynomials in evaluation form.

- Computes the commitment to $\overline{\varphi}_{\mathsf{batch},\mathsf{inst}}(x)$ as the linear combination of the relevant indexed commitments. Letting $I' \leq I$ denote the number of distinct instructions in the program P, this amounts to one fixed-base MSM of $9 \cdot I'$ elements *of the $\mathbb{F}_{\mathbb{i}}$-vector space of commitments.*
- Verifies the opening proof corresponding to the 28 claimed evaluations with a single call to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Check}$.
- Runs the decision algorithm of $\mathbf{V}$. Steps i), ii), iv) and v) perform a small number of checks involving the queried evaluations and values of several vanishing polynomials, requiring $O(\log(d_{\max}))$ mul. Step iii) performs a single pass through the program and essentially computes the structural part of the full-program matrix fingerprints. It requires $O(n_{\mathsf{stru}})$ field operations, where n_{stru} is the number of structural constraints in P. Assuming the number of input/outpus of all instructions in the indexed language is bounded by a constant C, step iii) requires $O(C \cdot I)$ mul.

vkproof **eAHP and Marlin PCS:** We now incorporate the costs resulting from compilation with the specific PCS $\mathsf{PC}_{\mathsf{Marlin}}$ from [Chi+20] (cf. appendices C and D for the definition and table 2 for the efficiency parameters[52]). The main operations contributing to the computational complexity of the polynomial commitment scheme are pairings on $\mathbb{G}_1 \times \mathbb{G}_2$, which we denote by pair; and variable-and fixed-base multi-scalar multiplications in $\mathbb{G}_1$, which we denote by $v\text{-}\mathsf{msm}(n)$ and $f\text{-}\mathsf{msm}(n)$ for a given number of group elements n, respectively.

We resort to the same notation as above and repeat eAHP-related costs for reference, highlighting the new parts in blue. Note that all commitments in $(\mathsf{eAHP}_{\mathsf{vkproof}})_{\mathsf{PC}_{\mathsf{Marlin}}}$ (including those in the instances of $(\mathcal{R}_{\mathsf{VarunaVK}})_{\mathsf{PC}_{\mathsf{Marlin}}}$) have empty hiding randomness.

- **Round complexity**: $\mathsf{eAHP}_{\mathsf{vkproof}}$ has constant round complexity $\mathsf{k}(|\mathbb{i}|) = 2$ rounds, and therefore $\mathcal{P}$ and $\mathcal{V}$ interact throughout 4 rounds.
- **Communication complexity**: $\mathcal{P}$ sends the verifier $3 + 6 = 9$ commitments, i. e. 18 elements of $\mathbb{G}_1$; $|Q| = 28$ field elements as query replies; and one PCS opening proof consisting of 6 elements of $\mathbb{G}_1$ (as there are 6 query points). This results in a prover-to-verifier communication complexity of 24 elements of $\mathbb{G}_1$ and 28 field elements[53].

  $\mathcal{V}$ sends the prover $\rho_1 + \rho_2 + \rho_3 + \rho_4 = 0 + 2 + 1 + 1 = 4$ field elements.
- **Indexer time**: $\mathcal{I}$ performs the following operations:
  - Runs $\mathbf{I}$ on the index $\mathbb{i}$ containing a list of I instructions and a maximum depth $t_{\mathbb{i}}$, where $\mathsf{N}_{\mathbb{i}} = 2^{t_{\mathbb{i}}}$. For each instruction ν, $\mathbf{I}$ computes $\mathsf{s}(\mathbb{i}, 0) \leq 3 + 6 \cdot t_{\mathbb{i}}$ matrix-fingerprint polynomials of degree $< d_{M_\nu}$ each (where $M \in \{A, B, C\}$ is the corresponding matrix). This amounts to $\sim 2 \cdot t_{\mathbb{i}} \cdot \sum_\nu d_{A_\nu} + d_{B_\nu} + d_{C_\nu}$ mul.
  - Trims the SRS to the degree bound $\mathsf{N}_{\mathbb{i}}$.

[52] We consider the algebraic-group-model variant therein.

[53] This corresponds to the size of a SNARK proof after application of the Fiat-Shamir transform. If the base and scalar fields of the curve have ~ 256-bit elements and $\mathbb{G}_1$ points are sent (i. e. absorbed into the sponge) in compressed form, the resulting proof size is ~ 1.7 KB.

- Commits to the $\mathsf{s}(\mathbb{i}, 0)$ polynomials from the first step:

$$\sim 4 \cdot t_{\mathbb{i}} \cdot \sum_{\nu} v\text{-}\mathsf{msm}(d_{A_\nu}) + v\text{-}\mathsf{msm}(d_{B_\nu}) + v\text{-}\mathsf{msm}(d_{C_\nu}).$$

- Computes any desired preprocessed multi-scalar-multiplication data on the $\mathsf{s}(\mathbb{i}, 0)$ commitments, which we do not detail here. It should be noted that only indexed polynomials corresponding to the same depth t will ever be multiplied together by the verifier, which may reduce the size and computational cost of MSM precomputations.

- **Prover time**: $\mathcal{P}$ performs the following operations:
  - First round of $\mathbf{P}$, i. e. computing $\mathsf{val}_A(x)$, $\mathsf{val}_B(x)$ and $\mathsf{val}_C(x)$: $O(d_{\max} \cdot \log(d_{\max}))$ mul.
  - Second round of $\mathbf{P}$, i. e. computation of $\mathsf{pow}_{M,\alpha}(x)$ for $M \in \{A, B, C\}$, $q(x)$, $h(x)$ and $g(x)$: $O(d_{\max} \cdot \log(d_{\max}))$ mul. The superlinear factor essentially comes from the $\mathsf{row}_M(x) \cdot \mathsf{col}_M(x)$ and $\mathsf{rowcol}_M(x) \cdot \mathsf{val}(x)$ products, which can be computed using FFTs[54].
  - Makes 2 calls to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Commit}$, one per round of $\mathbf{P}$: $18 \cdot v\text{-}\mathsf{msm}(\leq d_{\max})$ in total (recall there is no hiding randomness).
  - Computes the unique linear combination $\overline{\varphi}_{\mathsf{batch},\mathsf{inst}}(x)$ (equation (18)) of indexed polynomials in the query set: $O(d_{\max})$ mul.
  - Evaluates all queried polynomials at the relevant points according to the 28 queries: $\leq 28 \cdot d_{\max}$ mul.
  - Performs a single call to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Open}$, which translates to one MSM per query: $28 \cdot v\text{-}\mathsf{msm}(\leq d_{\max})$. Note that the MSM in $\mathsf{PC}_{\mathsf{Marlin}}.\mathsf{Open}$ has linear costs in the number of non-zero coefficients of the polynomial being opened (in particular, in its degree) and therefore the bound $d_{\max}$ is somewhat loose. For instance, the hybrid fingerprint polynomial $\overline{\varphi}_{\mathsf{row},\mathcal{M},2^t,\alpha}(x)$ has degree less than the density of the largest instruction in the program P, and the power polynomial $\mathsf{pow}_{M,\alpha}(x)$ has degree less than the full-matrix density d_M.

  These costs are dominated by $O(d_{\max} \cdot \log(d_{\max}))$ mul and 46 $v\text{-}\mathsf{msm}(\leq d_{\max})$.

- **Verifier time**: The verifier $\mathcal{V}$ performs the following main operations. Note that sampling the $\sum_{i=1}^{4} |\rho_i| = 4$ random field elements and computing the 6 points in the query set Q takes negligible time. On the other hand, computing the coefficients of the unique linear combination in Q requires one pass through the program, which is merged with the structural-fingerprint computation in the decision phase below.
  - Computes the commitment to $\overline{\varphi}_{\mathsf{batch},\mathsf{inst}}(x)$ as the linear combination of the relevant indexed commitments. Letting $I' \leq I$ denote the number of distinct instructions in the program P, this amounts to 2 $f\text{-}\mathsf{msm}(9 \cdot I')$, since $\mathsf{PC}_{\mathsf{Marlin}}$ commitments contain two elliptic-curve points.
  - Verifies the opening proof corresponding to the 28 claimed evaluations with a single call to $\mathsf{PC}_{\mathsf{NH}}.\mathsf{Check}$. The numbers of polynomials queried at the 6 query points[55] are

[54] Since all involved polynomial are typically obtained in evaluation form first and then converted to the monomial basis using FFTs, one of the two FFTs in each of the relevant products can be avoided if $\mathcal{P}$ is also provided with the polynomials in evaluation form.

[55] Note that, if two or more of the padded densities $\{d_A, d_B, d_C\}$ happen to be equal, so are corresponding query points $\{\omega_A^{-1} \cdot \gamma, \omega_B^{-1} \cdot \gamma, \omega_C^{-1} \cdot \gamma\}$. This can be used to saive two or four pairings in that case.

21, 3, 1, 1, 1 and 1, respectively. The total cost to check the corresponding opening proof is 1 v-msm$(2 \cdot 21)$, 1 v-msm$(2 \cdot 3)$ and 4 v-msm(2), as well as 12 pair.

- Runs the decision algorithm of $\mathbf{V}$. Steps i), ii), iv) and v) perform a small number of checks involving the queried evaluations and values of several vanishing polynomials, requiring $O(\log(d_{\max}))$ mul. Step iii) performs a single pass through the program and essentially computes the structural part of the full-program matrix fingerprints. It requires $O(n_{\mathsf{stru}})$ field operations, where n_{stru} is the number of structural constraints in P. Assuming the number of input/outpus of all instructions in the indexed language is bounded by a constant C, step iii) requires $O(C \cdot I)$ mul.

These costs are dominated by $O(C \cdot I)$ mul, 2 f-msm(42) and 12 pair.

In order to reduce the amount of verifier pairings, a different PCS (or a modification of $\mathsf{PC}_{\mathsf{Marlin}}$) could be used where the number of pairings were independent of the number of queried points (cf. [Chi+20] section 9.2, paragraph "Batching pairing equations"). Separately from that, it is likely that the query point 1 and the three queries at it (on the power polynomials $\mathsf{pow}_{M,\alpha}(x)$) can be avoided with the trick mentioned in footnote 42.

6 Future work

We conclude by outline some natural directions to explore in future work:

1) Implement the protocol – for instance, in the setting of snarkVM. This will, on one hand, offer a more concrete idea of the speedup achieved by verifying proofs of key synthesis compared to the current option of key re-generation; and, on the other, illustrate the challenges of adapting a complex, real-world language such as Leo (or Aleo instructions) to the clean, idealised formalism of modular compilation.

2) One could introduce into the protocol a separate class of "super-sparse instructions" which, due to their simplicity, can be more efficiently treated analogously to structural constraints (that is, having the verifier compute the value of their fingerprints on the fly) than to full-blown instructions (where the verifier has to operate on the indexed commitments to their fingerprint polynomials). A prime candidate for this is, for instance, the field-multiplication instruction mul, containing a single constraint.

3) Determine to what extent modular compilation enables faster *proving-key generation*[56]. More specifically, just like the vkproof indexer precomputes fingerprint polynomials for each individual-instruction R1CS, it could also expose the plain row, col and val (possibly rowcol and rowcolval) polynomials of said R1CS[57]. A party wishing to generate the proving key for a specific program P could then combine the relevant precomputed polynomials using the

[56] Here we mean direct key generation rather than verification by means of a proof system such as the one described in the previous point

[57] Just like in the case of fingerprint polynomials, this approach requires a set of precomputed $\mathsf{row}_{M,2^i}$, $\mathsf{col}_{M,2^i}$ and $\mathsf{val}_{M,2^i}$ polynomials for each possible full-circuit size $1, 2, 2^2, \ldots, 2^{t_{\mathrm{i}}}$. The reason is that, in order for those polynomials to combine nicely under matrix concatenation, they need to interpolate the value zero in a large part of the density domain K_{2^i} of the full-program R1CS.

simple equations describing their behaviour under diagonal matrix concatenation, thus obtaining their full A_{inst}, B_{inst} and C_{inst} counterparts (in the notation of subsection 3.2). Only a small FFT would be needed in order to incorporate the information about A_{stru}. One may optimistically expect a speedup from the $O(d \cdot \log(d))$ of current proving-key generation to $O(d)$ using modular compilation and precomputation, but more careful analysis is needed in order to substantiate any complexity claims.

Note that this pursuit is independent of the vkproof eAHP and should instead be regarded as a positive side-effect of switching to modular compilation.

4) Extend the protocol to attest to the correctness of not only verifying keys, but also *proving* ones. This would open the door to proving-key generation as a service, where a party interested in a program P's proving keys may delegate their synthesis to a different entity and simply verify a succinct proof of honest key generation. On one hand, the usefulness of this is somewhat unclear as proving keys are of interest mainly to proving parties which typically have the necessary resources to directly synthesise the proving key by themselves. On the other hand, the changes to the vkproof protocol needed to encompass this functionality are minimal: simply move the full Varuna-key polynomials into the instance and replace the opening of their commitments in the original vkproof by direct evaluation. This has a cost of $O(d)$, with d denoting the density of the full-program R1CS, as opposed to the $O(d \cdot \log(d))$ asymptotics of proving-key synthesis.

5) Study which other existing languages and R1CS proof systems our techniques can be applied to without substantially departing from the current design. As already mentioned, vkproof encompasses both the compilation step and the proof system itself, which means it is likely only extendable to scenarios where both components exist. Marlin-based proof systems are good candidates for exploration.

6) Formally define a notion of knowledge soundness and zero knowledge for the eAHP primitive and study its behaviour under compilation with a PCS, which should likely follow the template in [Chi+20]. This has not been treated in this text since it is not directly relevant to vkproof, as already explained.

References

[Chi+20] Alessandro Chiesa et al. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *Advances in Cryptology – EUROCRYPT 2020*. Ed. by Anne Canteaut and Yuval Ishai. Cham: Springer International Publishing, 2020, pp. 738–768. ISBN: 978-3-030-45721-1.

[Bow+20] Sean Bowe et al. "ZEXE: Enabling Decentralized Private Computation". In: *2020 IEEE Symposium on Security and Privacy (SP)*. 2020, pp. 947–964. DOI: 10.1109/SP40000.2020.00050.

[AST24] Arasu Arun, Srinath Setty, and Justin Thaler. "Jolt: SNARKs for Virtual Machines via Lookups". In: *Advances in Cryptology – EUROCRYPT 2024*. Ed. by Marc Joye and Gregor Leander. Springer Nature Switzerland, 2024, pp. 3–33.

[STW24] Srinath Setty, Justin Thaler, and Riad Wahby. "Unlocking the Lookup Singularity with Lasso". In: *Advances in Cryptology – EUROCRYPT 2024*. Ed. by Marc Joye and Gregor Leander. Springer Nature Switzerland, 2024, pp. 180–209.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. "Global-Scale Secure Multiparty Computation". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Association for Computing Machinery, 2017, pp. 39–56. URL: https://doi.org/10.1145/3133956.3133979.

[Buc+25] Thomas Buchsteiner et al. *webSPDZ: Versatile MPC on the Web*. Cryptology ePrint Archive, Paper 2025/487. 2025. URL: https://eprint.iacr.org/2025/487.

[Chi+23] Alessandro Chiesa et al. "EOS: efficient private delegation of zkSNARK provers". In: *Proceedings of the 32nd USENIX Conference on Security Symposium*. SEC '23. USENIX Association, 2023.

[Yan+24] Yunbo Yang et al. *Siniel: Distributed Privacy-Preserving zkSNARK*. Cryptology ePrint Archive, Paper 2024/1803. 2024. URL: https://eprint.iacr.org/2024/1803.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Paper 2019/953. 2019. URL: https://eprint.iacr.org/2019/953.

[Che+23] Binyi Chen et al. "HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates". In: *Advances in Cryptology – EUROCRYPT 2023*. Springer Nature Switzerland, 2023, pp. 499–530.

[Mal+19] Mary Maller et al. "Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. Association for Computing Machinery, 2019, pp. 2111–2128. URL: https://doi.org/10.1145/3319535.3339817.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. "LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. Association for Computing Machinery, 2019, pp. 2075–2092. URL: https://doi.org/10.1145/3319535.3339820.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *Theory of Cryptography*. Ed. by Martin Hirt and Adam Smith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 31–60.

[Can+19] Ran Canetti et al. "Fiat-Shamir: from practice to theory". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2019. Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 1082–1090.

[Hol19] Justin Holmgren. *On Round-By-Round Soundness and State Restoration Attacks*. Cryptology ePrint Archive, Paper 2019/1261. 2019. URL: https://eprint.iacr.org/2019/1261.

[Blo+23] Alexander R. Block et al. "Fiat-Shamir Security of FRI and Related SNARKs". In: *Advances in Cryptology – ASIACRYPT 2023*. Ed. by Jian Guo and Ron Steinfeld. Singapore: Springer Nature Singapore, 2023, pp. 3–40.

Appendix A Power polynomials

Let $n \in \mathbb{N} \setminus \{0\}$ and $\mathbb{F}$ be a field whose unit group $\mathbb{F}^*$ contains a subgroup H of order n, for which we fix a distinguished generator ω. Given $\alpha \in \mathbb{F}$, let $\mathsf{pow}_\alpha(x) \in \mathbb{F}^{<n}[x]$ be the unique polynomial of degree less than n which maps ω^i to α^i for $0 \leq i < n$ with the convention that $\mathsf{pow}_0(1) = 1$.

We denote by $\hat{v}_H(x) = (x^n - 1)/(x - 1) = \sum_{i=0}^{n-1} x^i \in \mathbb{F}^{=n-1}[x]$ the vanishing polynomial over $H \setminus \{1\}$.

Lemma 18.

i) The relation

$$\alpha \cdot f(\omega^{-1} \cdot x) - f(x) = \frac{\alpha^n - 1}{n} \cdot \hat{v}_H(x), \tag{24}$$

is satisfied by $f = \mathsf{pow}_\alpha$.

ii) If $\alpha \notin H$, then[58] $\mathsf{pow}_\alpha(x)$ is the unique polynomial $f \in \mathbb{F}^{<n}[x]$ of degree less than n satisfying (24) and it is given by

$$\mathsf{pow}_\alpha(x) = \frac{\alpha^n - 1}{n} \cdot \sum_{i=0}^{n-1} \frac{1}{\alpha \cdot \omega^{-i} - 1} \cdot x^i. \tag{25}$$

iii) If $\alpha \in H$, then $\mathsf{pow}_\alpha(x) = x^j$, where j is the unique integer $0 \leq j < n$ satisfying $\alpha = \omega^j$. Furthermore, the polynomials satisfying (24) are exactly the polynomials of the form $t \cdot x^j$ with $t \in \mathbb{F}$.

iv) If $f \in \mathbb{F}^{<n}[x]$ is different from $\mathsf{pow}_\alpha(x)$, then

$$\Pr_{\gamma \xleftarrow{R} \mathbb{F}} \left(\alpha \cdot f(\omega^{-1} \cdot \gamma) - f(\gamma) = \frac{\alpha^n - 1}{n} \cdot \hat{v}_H(x) \text{ and } f(1) = 1\right) \leq \frac{n-1}{|\mathbb{F}|}.$$

Proof. i) Since $\alpha \cdot \mathsf{pow}_\alpha(\omega^{i-1}) - \mathsf{pow}_\alpha(\omega^i) = 0$ for $0 < i \leq n-1$, one has

$$\frac{\alpha \cdot \mathsf{pow}_\alpha(\omega^{-1} \cdot x) - \mathsf{pow}_\alpha(x)}{\hat{v}_H(x)} = q(x)$$

for some $q(x) \in \mathbb{F}[x]$. The numerator has degree at most equal to the denominator (i. e. $n-1$) and hence $q(x)$ is in fact a constant $k \in \mathbb{F}$. Substituting $x = 1$ in the above expression yields

$$k = \frac{\alpha^n - 1}{\hat{v}_H(1)} = \frac{\alpha^n - 1}{n}.$$

ii) When $\alpha \notin H$, the homomorphism of $\mathbb{F}$-vector spaces

$$\begin{aligned} T_\alpha \colon \mathbb{F}^{<n}[x] &\xrightarrow{\sim} \mathbb{F}^{<n}[x] \\ f(x) &\mapsto \alpha \cdot f(\omega^{-1} \cdot x) - f(x) \\ \sum_{i=0}^{n-1} c_i \cdot x^i &\mapsto \sum_{i=0}^{n-1} (\alpha \cdot \omega^{-i} - 1) \cdot c_i \cdot x^i \end{aligned} \tag{26}$$

is injective as $(\alpha \cdot \omega^{-i} - 1)$ does not vanish for any i. In particular, pow_α is the unique preimage under it of $T_\alpha(\mathsf{pow}_\alpha)$, which is precisely $(\alpha^n - 1)/n \cdot \hat{v}_H(x)$ by i). The explicit expression now follows from (26) and $\hat{v}_H(x) = \sum_{i=0}^{n-1} x^i$.

[58] All statements in the lemma also apply to $\alpha = 0$ (which never belongs to H).

iii) If $\alpha \in H$ equals ω^j for some $0 \leq j < n$, then x^j and $\mathsf{pow}_\alpha(x)$ coincide over all of H and therefore also as polynomials since they both have degree less than n. (26) shows that, when $\alpha = \omega^j$, the images under T_α of two polynomials coincide if and only if their coefficients of all terms other than x^j do, which shows iii).

iv) If $\alpha \notin H$, then $f \neq \mathsf{pow}_\alpha(x)$ implies $T_\alpha(f) \neq T_\alpha(\mathsf{pow}_\alpha)$ by ii). In particular,

$$\begin{aligned}
&\Pr_{\gamma \xleftarrow{R} \mathbb{F}} \left(\alpha \cdot f(\omega^{-1} \cdot \gamma) - f(\gamma) = \frac{\alpha^n - 1}{n} \cdot \hat{v}_H(x) \text{ and } f(1) = 1\right) \\
\leq &\Pr_{\gamma \xleftarrow{R} \mathbb{F}} \left(\alpha \cdot f(\omega^{-1} \cdot \gamma) - f(\gamma) = \frac{\alpha^n - 1}{n} \cdot \hat{v}_H(x)\right) \\
= &\Pr_{\gamma \xleftarrow{R} \mathbb{F}} \left(T_\alpha(f)(\gamma) = T_\alpha(\mathsf{pow}_\alpha)(\gamma)\right) \\
\leq &\frac{n-1}{|\mathbb{F}|}
\end{aligned}$$

by the Schwartz-Zippel lemma (note that $\deg(T_\alpha(f)) \leq \deg(f)$ in general).

If $\alpha \in H$ is of the form ω^j for some $0 \leq j < n$, we distinguish two cases:

- If f is not of the form $t \cdot x^j$ for any $t \in \mathbb{F}$, then $T_\alpha(f) \neq T_\alpha(\mathsf{pow}_\alpha(x))$ by iii) and the same argument as above yields

$$\Pr_{\gamma \xleftarrow{R} \mathbb{F}} \left(\alpha \cdot f(\omega^{-1} \cdot \gamma) - f(\gamma) = \frac{\alpha^n - 1}{n} \cdot \hat{v}_H(x) \text{ and } f(1) = 1\right) \leq \frac{n-1}{|\mathbb{F}|}$$

- If f is of the form $t \cdot x^j$ for some $t \in \mathbb{F}$, then $t \neq 1$ (since by assumption $f \neq \mathsf{pow}_\alpha(x) = x^j$) and thus

$$\Pr_{\gamma \xleftarrow{R} \mathbb{F}} \left(\alpha \cdot f(\omega^{-1} \cdot \gamma) - f(\gamma) = \frac{\alpha^n - 1}{n} \cdot \hat{v}_H(x) \text{ and } f(1) = 1\right) \leq \Pr_{\gamma \xleftarrow{R} \mathbb{F}} \left(f(1) = 1\right) = 0.$$

■

Appendix B Compilation with hiding polynomial commitment schemes

In the body of the text, we have restricted our attention to compilation of extended algebraic holographic proofs with polynomial commitments that are not hiding. This appendix briefly illustrates the motivation behind that choice and informally discusses the modifications needed to incorporate hiding PCS. There are multiple reasons one may be interested in allowing hiding polynomial commitment schemes in the eAHP paradigm. For instance, one may want to add a notion of zero-knowledge to eAHPs and the preprocessing arguments compiled from them. As a second, subtler reason, one may want to use the eAHP + PCS compilation process to obtain a preprocessing argument for an existing relation capturing an *already-deployed* proof system whose instance commitments come from a hiding PCS[59] (and, in particular, whose SRS includes information for producing hiding commitments).

Suppose given an indexed polynomial-oracle relation (cf. definition 1), where instances can contain oracles to witness polynomials; and a polynomial commitment scheme PC as in [Chi+20] section 6.1, where commitments can incorporate hiding randomness. There are at least two natural ways to approach the definition of the compiled relation $\mathcal{R}_{\mathsf{PC}}$, by contrast with the non-hiding case (definition 8):

i) Require that the commitments $(c_i)_{i=1}^n$ in the instance have the form

$$(c_i)_{i=1}^n = \mathsf{PC.Commit}(\mathsf{ck}, (f_i)_{i=1}^n, (\mathsf{d}_i)_{i=1}^n, (\bot)_{i=1}^n),$$

where $\bot$ denotes empty randomness.

ii) Require that the commitments $(c_i)_{i=1}^n$ in the instance have the form

$$(c_i)_{i=1}^n = \mathsf{PC.Commit}(\mathsf{ck}, (f_i)_{i=1}^n, (\mathsf{d}_i)_{i=1}^n, (\omega_i)_{i=1}^n)$$

for *some* randomness $(\omega_i)_{i=1}^n$.

Option ii) results in a largely trivial relation as the hiding property means that commitments carry no information *per se*. For instance, in the Marlin PCS ([Chi+20] appendices B-D), fixing a polynomial $f \in \mathbb{F}[x]$ and sampling hiding randomness $\omega \xleftarrow{R} \mathbb{F}$ uniformly at random yields a uniformly random distribution on the resulting commitment (as long as the toxic waste γ from the second portion of the SRS is non-zero). This means that even the definition of plain soundness for the resulting preprocessing argument would need to incorporate a knowledge-soundness-like condition on the witness polynomials f.

We favour option i), where instance commitments do carry information about witness polynomials by themselves and the unmodified definition of argument soundness remains meaningful. Furthermore, it is this option that captures the $\mathsf{VarunaVK}$ relation in practice, where the Varuna (not $\mathsf{vkproof}$) indexer produces (Varuna) verifying keys by committing to proving-key polynomials with empty randomness.

One possible approach is to separate the PCS in instance commitments from that used for compilation of the rest of the eAHP (i. e. for indexed and round polynomials). However, there can

[59] This is, indeed, the case in the Varuna implementation underpinning the Aleo blockchain.

be practical (and formal) reasons to abide by the single-PCS model. This introduces the need for some formal changes on the PCS primitive, since the PC.Check method gives the verifier no guarantees that the commitments were produced with no randomness. For instance, the current paradigm could be modified as follows:

- The definition of a PCS (with hiding randomness) incorporates two[60] new methods: OpenNH and CheckNH (where NH stands for "non-hiding"). These are added to the original quintuple (Setup, Trim, Commit, Open, Check). The PCS completeness definition is expanded with the condition that commitments produced with empty randomness $\perp$ and opened with OpenNH result in accepting proofs under CheckNH.

- Commitment extractability (definition 4) is tweaked by requiring that, if PC.Check passes, then *there exists* some randomness $(\omega_{i,j})_{i,j}$ such that

$$(c_{i,j})_{i,j} = \mathsf{PC.Commit}(\mathsf{ck}, (f_{i,j})_{i,j}, (\mathsf{d}_{i,j})_{i,j}, (\omega_{i,j})_{i,j}).$$

  (alternatively, one can require the extractor to output the randomness in addition to the polynomials). More importantly, a second analogous extractability condition is added requiring that, in the same game, if PC.CheckNH passes, then the recovered polynomials satisfy

$$(c_{i,j})_{i,j} = \mathsf{PC.Commit}(\mathsf{ck}, (f_{i,j})_{i,j}, (\mathsf{d}_{i,j})_{i,j}, (\perp)_{i,j}).$$

- The single call to PC.Open in the compiled preprocessing argument is replaced with two: one to PC.OpenNH to open instance commitments at the queried points, and a second one to PC.Open to open the rest (corresponding to indexed polynomials, their linear combinations, and round polynomials). The call to PC.Check in the argument verifier's decision phase is split analogously into one call to PC.CheckNH and one to PC.Check. If the metrics of PC.OpenNH and PC.CheckNH are more desirable than those of PC.Open and PC.Check with empty randomness (which they may not be, as they are proving and verifying a stronger statement), then the indexed polynomials and their linear combinations can be opened and checked with the former instead.

With the above modifications, the proof of soundness for the compiled argument (proposition 11) easily carries over to the new setting since the polynomials obtained by the extractor[61] at the end of the proof result in the commitments in the instance with no hiding randomness, as required in the definition of the compiled relation $\mathcal{R}_{\mathsf{PC}}$.

The definition of PC.OpenNH and PC.CheckNH are completely scheme dependent. We illustrate this with the Marlin PCS. For simplicity, we focus on the single-degree-bound, single-polynomial case in the algebraic group model ([Chi+20] appendix B.3.1 with $n = 1$ polynomial). It is important to note that, if an SRS for the hiding variant has been generatred (i. e. one containing both $\{G, \beta G, \ldots, \beta^D G\}$ and $\{\gamma \cdot G, \gamma \cdot \beta G, \ldots, \gamma \cdot \beta^D G\}$), *it is not suffcient* to define $\mathsf{PC}_{\mathsf{Marlin}}$.CheckNH as the equation $e(C - vG, H) = e(\mathsf{w}, \beta H - zH)$ in the notation of paragraph "Check" therein – i. e. simply remove the term $-\gamma \cdot \overline{v} G$ from the first operand of the left-hand pairing. For instance, if the point z is not randomly sampled from a large space after the commitment has been sent (say, z is a

[60] Depending of the specifics of the PCS, it might be useful to also introduce a CommitNH method separate from Commit$(\ldots, (\perp))$.

[61] In some more detail, one has to construct two commitment-extractability adversaries and argue about their two separate extractors.

fixed element of $\mathbb{F}$ established by the eAHP), a malicious prover could choose a hiding polynomial $\overline{p}$ which vanishes at z, resulting in $\overline{v} = \overline{p}(z) = 0$ and making our strawman $\mathsf{PC.CheckNH}$ accept despite operating on a commitment with hiding randomness.

It would be acceptable to restrict the definition of extractability for non-hiding commitments to the game where the committed polynomials are queried at (at least) a uniformly random point in the field; and to require the eAHP to query any instance oracles it does query at at least one such point. Indeed, $\mathsf{eAHP}_{\mathsf{vkproof}}$ is of this form. However, even then, it does not seem possible to reduce the resulting extractability game to the strong Diffie-Hellman assumption ([Chi+20] section A.2). Instead, when opening a commitment C with $\mathsf{PC}_{\mathsf{Marlin}}.\mathsf{OpenNH}$, the prover could also be asked to supply $\widetilde{C}$ satisfying $e(\widetilde{C}, H) = e(C, \gamma H)$ (i. e. $\widetilde{C} = \gamma C$), which would be checked by the verifier. An algebraic adversary producing a commitment $C = p(\beta)G + \gamma \cdot \overline{p}(\beta)G$ could only compute $\widetilde{C} = \gamma \cdot p(\beta)G + \gamma^2 \cdot \overline{p}(\beta)G$ using the public SRS if $\overline{p}(\beta) = 0$. Note this approach requires a new element γH to be generated during setup and added to the verifying key.

We leave exploration of this space to future work.